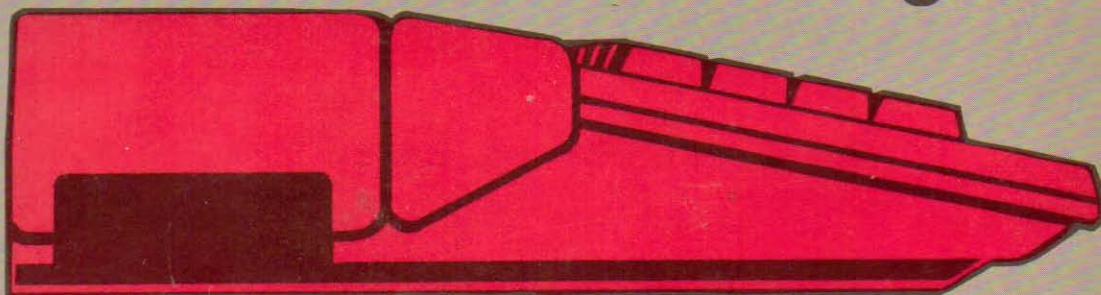


Third Revision—Includes blitter chip information

# ATARI® ST™

## INTERNALS

The authoritative insider's guide



A Data Becker book published by

You Can Count On  Software

# ATARI<sup>1</sup>ST INTERNALS

The authoritative insider's guide

By K. Gerits, L. Englisch, R. Bruckmann

A Data Becker Book

Published by

Abacus  Software

Third Printing, July 1986  
Printed in U.S.A.  
Copyright © 1985

Copyright © 1985

Data Becker GmbH  
Merowingerstr.30  
4000 Dusseldorf, West Germany  
Abacus Software, Inc.  
P.O. Box 7219  
Grand Rapids, MI 49510

This book is copyrighted. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Abacus Software or Data Becker, GmbH.

Every effort has been made to insure complete and accurate information concerning the material presented in this book. However Abacus Software can neither guarantee nor be held legally responsible for any mistakes in printing or faulty instructions contained in this book. The authors will always appreciate receiving notice of subsequent mistakes.

ATARI, 520ST, ST, TOS, ST BASIC and ST LOGO are trademarks or registered trademarks of Atari Corp.

GEM, GEM Draw and GEM Write are trademarks or registered trademarks of Digital Research Inc.

IBM is a registered trademark of International Business Machines.

ISBN 0-916439-46-1

---

## Table of Contents

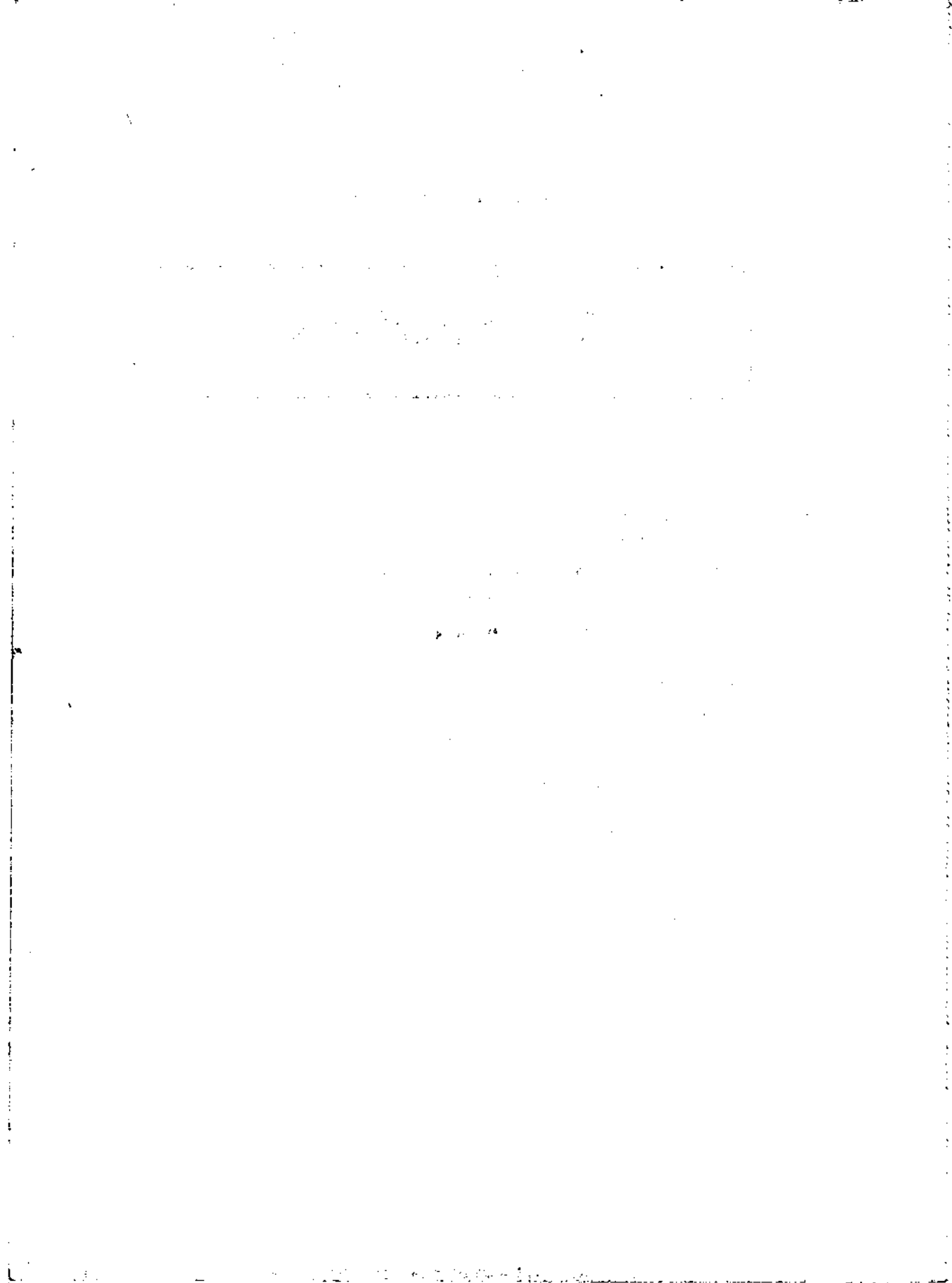
1	The Integrated Circuits	1
1.1	The 68000 Processor	3
1.1.1	The 68000 Registers	4
1.1.2	Exceptions on the 68000	7
1.1.3	The 68000 Connections	7
1.2	The Custom Chips	13
1.3	The WD 1772 Floppy Disk Controller	20
1.3.1	1772 Pins	20
1.3.2	1772 Registers	24
1.3.3	Programming the FDC	25
1.4	The MFP 68901	28
1.4.1	68901 Connections	28
1.4.2	The MFP Registers	32
1.5	The 6850 ACIAs	41
1.5.1	The Pins of the 6850	41
1.5.2	The Registers of the 6850	44
1.6	The YM-2149 Sound Generator	48
1.6.1	Sound Chip Pins	50
1.6.2	The 2149 Registers and their Functions	52
1.7	I/O Register Layout of the ST	55
2	The Interfaces	65
2.1	The Keyboard	67
2.1.1	The mouse	71
2.1.2	Keyboard commands	74
2.2	The Video Connection	85
2.3	The Centronics Interface	88
2.4	The RS-232 Interface	90
2.5	The MIDI Connections	93



2.6	The Cartridge Slot	96
2.7	The Floppy Disk Interface	97
2.8	The DMA Interface	99
3	The ST Operating System	101
3.1	The GEMDOS	104
3.1.1	GEMDOS error codes and their meaning	139
3.2	The BIOS Functions of the Atari ST	140
3.3	The XBIOS	155
3.4	The Graphics	206
3.4.1	An overview of the "line-A" variables	226
3.4.2	Examples for using line-A opcodes	229
3.5	The Exception Vectors	234
3.5.1	The interrupt structure of the ST	236
3.6	The ST VT52 Emulator	242
3.7	The ST System Variables	247
3.8	The 68000 Instruction Set	255
3.8.1	Addressing modes	256
3.8.2	The instructions	260
3.9	The BIOS listing	268
4	Appendix - The System Fonts	443
4.1	The System Fonts	445
4.2	Alphabetical listing of GEMDOS functions	447

## List of Figures

1.1-1	68000 Registers	5
1.2-1	GLUE	14
1.2-2	MMU	16
1.2-3	SHIFTER	17
1.2-4	DMA	19
1.3-1	FDC 1772	21
1.4-1	MFP 68901	29
1.5-1	ACIA 6850	42
1.6-1	Sound Chip YM-2149	49
1.6-2	Envelopes of the PSG	53
2.1-1	6850 Interface to 68000	68
2.1-2	Block Diagram of Keyboard Circuit	70
2.1.1-1	The Mouse	72
2.1.1-2	Mouse control port	74
2.1.2-1	Atari ST Key Assignments	84
2.2-1	Diagram of Video Interface	86
2.2-2	Monitor Connector	87
2.3-1	Printer Port Pins	88
2.3-2	Centronics Connection	89
2.4-1	RS-232 Connection	92
2.5-1	MIDI System Connection	95
2.6-1	The Cartridge Slot	96
2.7-1	Disk Connection	98
2.8-1	DMA Port	100
2.8-2	DMA Connections	100
3.4-1	Lo-Res-Mode	208
3.4-2	Medium-Res-Mode	210
3.4-3	Hi-Res-Mode	212



# Chapter One

## The Integrated Circuits

- 1.1 The 68000 Processor**
  - 1.1.1 The 68000 Registers**
  - 1.1.2 Exceptions on the 68000**
  - 1.1.3 The 68000 Connections**
- 1.2 The Custom Chips**
- 1.3 The WD 1772 Floppy Disk Controller**
  - 1.3.1 1772 Pins**
  - 1.3.2 1772 Registers**
  - 1.3.3 Programming the FDC**
- 1.4 The MFP 68901**
  - 1.4.1 68901 Connections**
  - 1.4.2 The MFP Registers**
- 1.5 The 6850 ACIAs**
  - 1.5.1 The Pins of the 6850**
  - 1.5.2 The Registers of the 6850**
- 1.6 The YM-2149 Sound Generator**
  - 1.6.1 Sound Chip Pins**
  - 1.6.2 The 2149 Registers and their Functions**
- 1.7 I/O Register Layout of the ST**





## The Integrated Circuits

### 1.1 The 68000 Processor

The 68000 microprocessor is the heart of the entire Atari ST system. This 16-bit chip is in a class by itself; programmers and hardware designers alike find the chip very easy to handle. From its initial development by Motorola in 1977 to its appearance on the market in 1979, the chip was to be a competitor to the INTEL 8086/8088 (the processor used in the IBM-PC and its many clones). Before the Atari ST's arrival on the marketplace, there were no affordable 68000 machines available to the home user. Now, though, with 16-bit computers becoming more affordable to the *common* man, the 8-bit machines won't be around much longer.

What does the 68000 have that's so special? Here's a very incomplete list of features:

- 16 data bits
- 24 address bits (16-megabyte address range!!)
- all signals directly accessible without multiplexer
- hassle-free operation of "old" 8-bit peripherals
- powerful machine language commands
- easy-to-learn assembler syntax
- 14 different types of addressing
- 17 registers each having 32-bit widths

These specifications (and many yet to be mentioned here) make the 68000 an incredibly good microprocessor for home and personal computers. In fact, as the price of memory drops, you'll soon be seeing 68000-based 64K machines for the same price as present-day 8-bit computers with the same amount of memory.

## 1.1.1 The 68000 Registers

Let's take a look at 68000 design. Figure 1.1-1 shows the 17 onboard 32-bit registers, the program counter and the status register.

The eight data registers can store and perform calculations, as well as the normal addressing tasks. Eight-bit systems use the accumulators for this, which limits the programmer to a total of 8 accumulators. Our 68000 data registers are quite flexible; data can be handled in 1-, 8-, 16- and 32- bit sizes. Even four-bit operations are possible (within the limits of Binary Coded Decimal counting). When working with 32-bit data, all 32 bits can be handled with a single operation. With 8- and 16-bit data, only the 8th or 16th bit of the data register can be accessed.

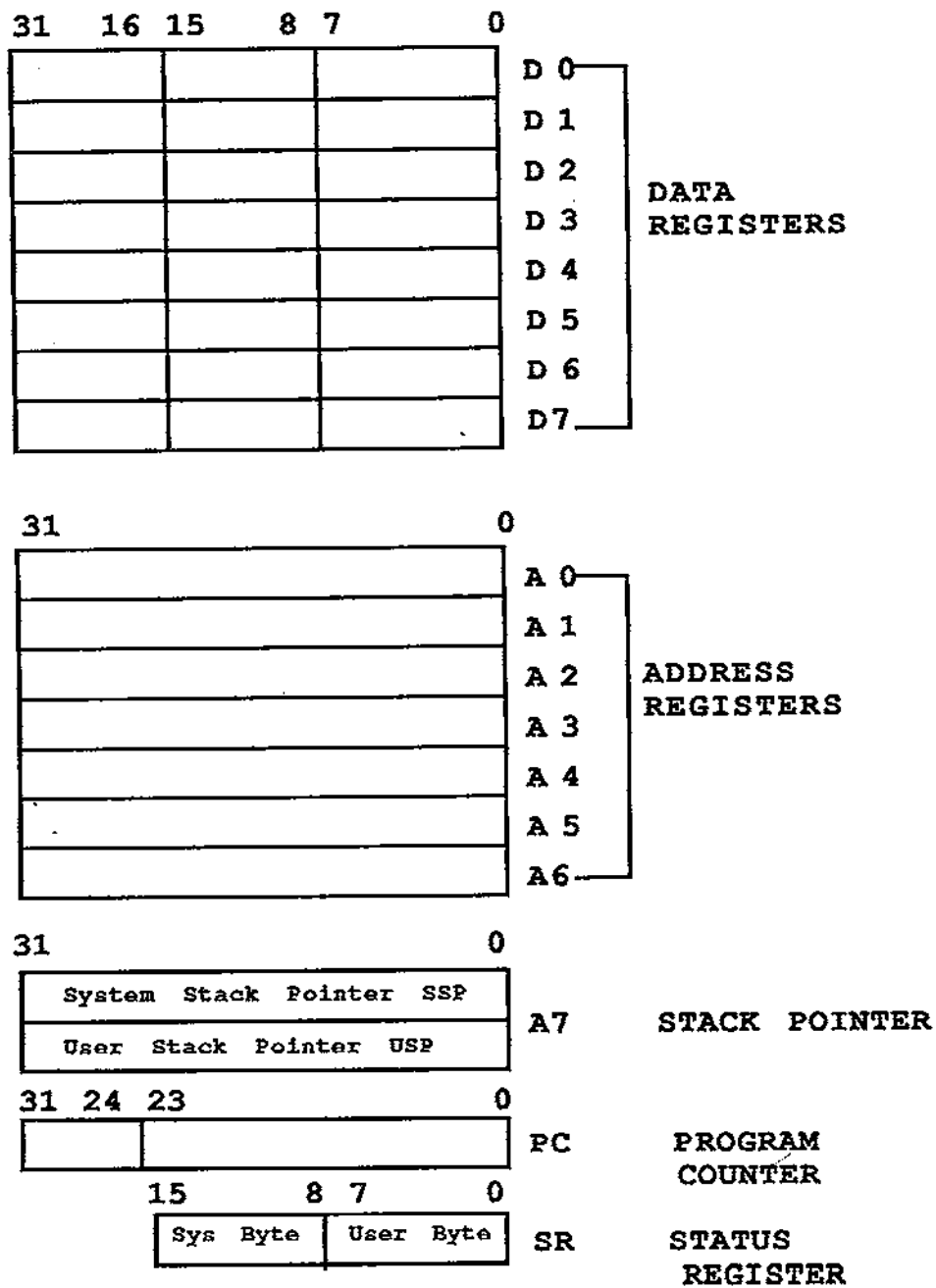
The address registers aren't as flexible for data access as are the data registers. These registers are for addressing, not calculation. Processing data is possible only with word (16-bit) and longword (32-bit) operations. The address registers must be looked at as two distinct groups, the most versatile being the registers A0-A6. Registers A7 and A7' fulfill a special need. These registers are used as the stack pointer by the processor. Two stack pointers are needed to allow the 68000 to run in USER MODE and SUPERVISOR MODE. Register A7 declares whether the system is in USER or SUPERVISOR mode. Note that the two registers work "under" A7, but the register contents are only available to the respective operating mode. We'll discuss these operating modes later.

The program counter is also considered a 32-bit register. It is theoretically possible to handle an address range of over 4 gigabytes. But the address bits A24-A31 aren't used, which "limits" us to 16 megabytes.

The 68000 status register comprises 16 bits, of which only 10 bits are used. This status register is divided into two halves: The lower eight bits (bits 0 to 4 proper) is the "user byte". These bits, which act as flags most of the time, show the results of arithmetical and comparative operations, and can be used for program branches hinging on those results. We'll look at the user byte in more detail later; for now, here is a brief list:

BIT 0 = Carry flag	BIT 1 = Overflow flag
BIT 2 = Zero flag	BIT 3 = Negative flag
BIT 4 = eXtend flag	

Figure 1.1-1 68000 Registers





Bits 8-10, 13 and 15 make up the status register's system byte. The remaining bits are unused. Bit 15 works as a trace bit, which lets you do a software controlled single-step execution of any program. Bit 13 is the supervisor bit. When this bit is set, the 68000 is in supervisor mode. This is the normal operating mode; all commands are executed in this mode. In user mode, in which programs normally run, privileged instructions are inoperative. A special hardware design allows access into the other memory range while in user mode (e.g., important system variables, I/O registers). The system byte of the status register can only be manipulated in supervisor mode; but there's a simple method of switching between modes.

Bits 8 and 10 show the interrupt mask, and run in connection with pins IPL0-IPL2.

The 68000 has great potential for handling interrupts. Seven different interrupt priorities exist, the highest being the "non-maskable interrupt"; NMI. This interrupt recognizes when all three IPL pins simultaneously read low (0). If, however, all three IPL pins read high, there is no interrupt, and the system operates normally. The other six priorities can be masked by appropriate setting of the system byte of the status register. For example, if bit I2 of the interrupt mask is set, while I0 and I1 are off, only levels 7, 6 and 5 (000, 001 and 010) are recognized. All other combinations from IPL0-IPL2 are ignored by the processor.

### 1.1.2 Exceptions on the 68000

We've spoken of interrupts as if the 68000 behaves like other microprocessors. Interrupts, according to Motorola nomenclature, are an external form of an exception (the machine can interrupt what it's doing, do something else, and return to the interrupted task if needed). The 68000 distinguishes between normal operation and exception handling, rather than between user and supervisor mode. One such set of exceptions is the interrupts. Other things which cause exceptions are undefined opcodes, and word or longword access to a prohibited address.

To make exception handling quicker and easier, the 68000 reserves the first 1K of memory (1024 bytes, \$000000-\$0003FF). The exception table is located here. Exceptions are all coded as one of four bytes of a longword. Encountering an exception triggers the 68000, and the address of the corresponding table entry is output.

A special exception occurs on reset, which requires 8 bytes (two longwords); the first longword contains the standard initial value of the supervisor stack pointer, while the second longword contains the address of the reset routine itself. See Chapter 3.3 for the design and layout of the exception table.

### 1.1.3 The 68000 Connections

The connections on the 68000 are divided into eight groups (see Figure 1.1-3 on page 11).

The first group combines data and address busses. The data bus consists of pins D0-D15, and the address bus A1-A23. Address bit A0 is not available to the 68000. Memory can be communicated with words rather than bytes (1 word=2 bytes=16 bits, as opposed to 1 byte=8 bits). Also, the 68000 can access data located on odd addresses as well as even addresses. The signals will be dealt with later.

It's important to remember in connection with this, that by word access to memory, the byte of the odd address is treated as the low byte, and the even

address is the high byte. Word access shouldn't stray from even addresses. That means that opcodes (whether all words or a single word) must always be located at an even addresses.

When the data and address bus are in "tri-state" condition, a third condition (in addition to high and low) exists, in which the pins offer high resistance, and thus are inactive on the bus. This is important in connection with Direct Memory Access (DMA).

The second group of connections comprise the signals for asynchronous bus control. This group has five signals, which we'll now look at individually:

**1) R/W (READ/WRITE)**

The R/W signal is a familiar one to all microprocessors. This indicates to memory and peripherals whether the processor is writing to or reading data from the address on the bus.

**2) AS (ADDRESS STROBE)**

Every processor has a signal which it sends along the data lines signaling whether the address is ready to be used. On the 68000, this is known as the ADDRESS STROBE (low active).

**3) UDS (UPPER DATA STROBE)**

**4) LDS (LOWER DATA STROBE)**

If the 68000 could only process an entire memory word (two bytes) simultaneously, this signal wouldn't be necessary. However, for individual access to the low-byte and high-byte of a word, the processor must be able to distinguish between the two bytes. This is the task performed by UDS and LDS. When a word is accessed, both strobes are activated simultaneously (active=low). Accessing the data at an odd address activates the Lower Data Strobe only, while accessing data at an even address activates the Upper Data Strobe.

Bit A0 from the address bus is used in this case. After every access when the system must distinguish between three conditions (word, even byte, odd byte), A0 determines how to complete the access.

LDS and UDS are tri-state outputs.

## 5) DTACK

The above signals (with the exception of UDS and LDS) are needed by an 8-bit processor. DTACK takes a different path; DTACK must be low for any write or read access to take place. If the signal is not low within a bus cycle, the address and data lines "freeze up" until DTACK turns low. This can also occur in a WAIT loop. This way, the processor can slow down memory and peripheral chips while performing other tasks. If no wait cycles are used on the ST, the processor moves "at full tilt".

The third group of connections, the signals VMA, VPA and E are for synchronous bus control. A computer is more than memory and a microprocessor; interfaces to keyboard, screen, printer, etc. must be available for communication. In most cases, interfacing is handled by special ICs, but the 68000 has a huge selection of interfaces chips onboard. For hardware designers we'll take a little time explaining these synchronous bus signals.

The signal E (also known as  $\Phi 2$  or phi 2) represents the reference count for peripherals. Users of 6800 and 6502 machines know this signal as the system counter. Whereas most peripheral chips have a maximum frequency of only 1 or 2 mHz, the 68000 has a working speed of 8 mHz, which can be increased to 10 by the E signal. The frequency of E in the ST is 800 kHz. The E output is always active; it is not capable of a TRI-STATE condition.

The signal VPA (Valid Peripheral Address) sends data over the synchronous bus, and delegates this transfer to specific sections of the chip. Without this signal, data transfer is performed by the asynchronous bus. VPA also plays a role in generating interrupts, as we'll soon see.

VMA (Valid Memory Address) works in conjunction with the VPA to produce the CHIP-select signal for the synchronous bus.

The fourth group of 68000 signals allows simple DMA operation in the 68000 system. DMA (Direct Memory Access) directly accesses the DMA controllers, which control computer memory, and which is the fastest method of data transfer within a computer system.

To execute the DMA, the processor must be in an inactive state. But for the processor to be signaled, it must be in a "sleep" state; the low BR signal



(Bus Request) accomplishes this. On recognizing the BR signal, the 68000's read/write cycle ends, and the BG signal (Bus Grant) is activated. Now the DMA-requested chip waits until the signals AS, DTACK and (when possible) BGACK are rendered inactive. As soon as this occurs, the BGACK (Bus Grant Acknowledge) is activated by the requested chip, and takes over the bus. All essential signals on the processor are made high; in particular, the data, address and control busses are no longer influenced by the processor. The DMA controller can then place the desired address on the bus, and read or write data. When the DMA chip is finished with its task, the BGACK signal returns to its inactive state, and the processor again takes over the bus.

The fifth group of signals on the 68000 control interrupt generation. The 68000's "user's choice" interrupt concept is one of its most extraordinary performing qualities; you have 199 (!) interrupt vectors from which to choose. These interrupt vectors are divided into 7 non-auto-vectors and 192 auto-vectors, plus 7 different priority lines.

Interrupts are triggered by signals from the three lines IPL0 to IPL2; these three lines give you eight possible combinations. The combination determines the priority of the interrupt. That is, if IPL0, IPL1 and IPL2 are all set high, then the lowest priority is set ("no interrupt"). However, if all three lines are low, then highest priority takes over, to execute a non-maskable interrupt. All the combinations in between affect special bits in the 68000's status register; these, in turn, affect program control, regardless of whether or not a chosen interrupt is allowable.

Wait -- what are auto-vectors and non-auto-vectors? What do these terms mean?

If requesting an interrupt on IPL0-IPL2 while VPA is active (low), the desired code is directly converted from the IPL pins into a vector number. All seven interrupt codes on the IPL pins have their own vectors, though. The auto-vector concept automatically gives the vector number of the IPL interrupt code needed.

When DTACK, instead of VPA, is active on an interrupt request, the interrupt is handled as a non-auto-vector. In this case, the vector number from the triggered chip is produced by DTACK on the 8 lowest bits of the data bus. Usually (though not important here), the vector number is placed into the user-vector range (\$40-\$FF).

The sixth set of connections are the three "function code" outputs FC0 to FC2. These lines handle the status display of the processor. With the help of these lines, the 68000 can expand to four times 16 megabytes (64 megabytes). This extension requires the MMU (Memory Management Unit). This MMU does more than handle memory expansion on the ST; it also recognizes whether access is made to memory in user or supervisor mode. This information is conveyed to a memory range only accessible in supervisor mode. Also, the interrupt verification uses this information on the FC line. The figure below shows the possible combinations of functions.

**Figure 1.1-3**

FC2	FC1	FC0	Status
0	0	0	unused
0	0	1	User-mode data access
0	1	0	User-mode program
0	1	1	unused
1	0	0	unused
1	0	1	Supervisor data access
1	1	0	Supervisor program
1	1	1	Interrupt verification

The seventh group contains system control signals. This group applies to the input CLK and BERR, as well as the bidirectional lines RESET and HALT.

The input CLK will generate the working frequency of the processor. The 68000 can operate at different speeds; but the operating frequency must be specified (4, 6, 8, 10, or even 12.5 MHz). The ST has 8 MHz built in, while the minimum operating frequency is 2 MHz. The ST's 8 MHz was chosen as a "middle of the road" frequency to avoid losing data at higher frequencies.

The RESET line is necessary to check for system power-up. The 68000's data page distinguishes between two different reset conditions. On power-up, RESET and HALT are switched low for at least 100 milliseconds, to set up a proper initialization. Every other initialization requires a low impulse of at least 4 "beats" on the 68K.

Here is what RESET does in detail. The system byte of the status register is loaded with the value \$27. Once the processor is brought into supervisor

status, the Trace flag in the status register is cleared, and the interrupt level is set to 7 (lowest priority, all lines allowable). Additionally, the supervisor stack pointer and program counter are loaded with the contents of the first 8 bytes of memory, whereby the value of the program counter is set to the beginning of the reset routine.

However, since the RESET line is bi-directional, the processor can also have RESET under program control during the time the line is low. The RESET instruction serves this purpose, when the connection is low for 124 "beats". It's possible to re-initialize the peripheral ICs at any time, without resetting the computer itself. RESET time puts the 68000 into a NOP state -- a reset is unstoppable once it occurs.

The HALT pin is important to the RESET line's existence (as we mentioned above), in order to initialize things properly. This pin has still more functions: when the pin is low while RESET is high, the processor goes into a halt state. This state causes the DMA pin to set the processor into the tri-state condition. The HALT condition ends when HALT is high again. This signal can be used in the design of single-step control.

HALT is also bi-directional. When the processor signals this line to become low, it means that a major error has occurred (e.g., doubled bus and address errors).

A low state on the BERR pin will call up exception handling, which runs basically like an external interrupt. In an orderly system, every access to the asynchronous bus quits with the DTACK signal. When DTACK is outputting, however, the hardware can produce a BERR, which informs the processor of any errors found. A further use for BERR is in connection with the MMU, to test for proper memory access of a specific range; this access is signaled by the FC pins. If protected memory is tried for in user mode, a BERR will turn up.

When both BERR and HALT are low, the processor will "re-execute" the instruction at which it stopped. If it doesn't run properly on the second "go-round", then it's called a *doubled* bus error, and the processor halts.

The eighth group of connections are for voltage and ground.

## 1.2 The Custom Chips

The Atari ST has four specially developed ICs. These chips (GLUE, MMU, DMA and SHIFTER) play a major role in the low price of the ST, since each chip performs several hundred overlapping functions. The first prototype of the ST was 5 X 50 X 30 cm. in size, mostly to handle all those TTL ICs. Once multiple functions could be crammed into four ICs, the ST became a saleable item. Then again, the present ST hasn't quite reached the ultimate goal -- it still has eight TTLs.

Naturally, since these chips were specifically designed by Atari for the ST, they haven't been publishing any spec sheets. Even without any data specs, we can give you quite a bit of information on the workings of the ICs.

An interesting fact about these ICs is that they're designed to work in concert with one another. For example, the DMA chip can't operate alone. It hasn't an address counter, and is incapable of addressing memory on its own (functions which are taken care of by the MMU). It's the same with SHIFTER -- it controls video screen and color, but it can't address video RAM. Again, MMU handles the addressing.

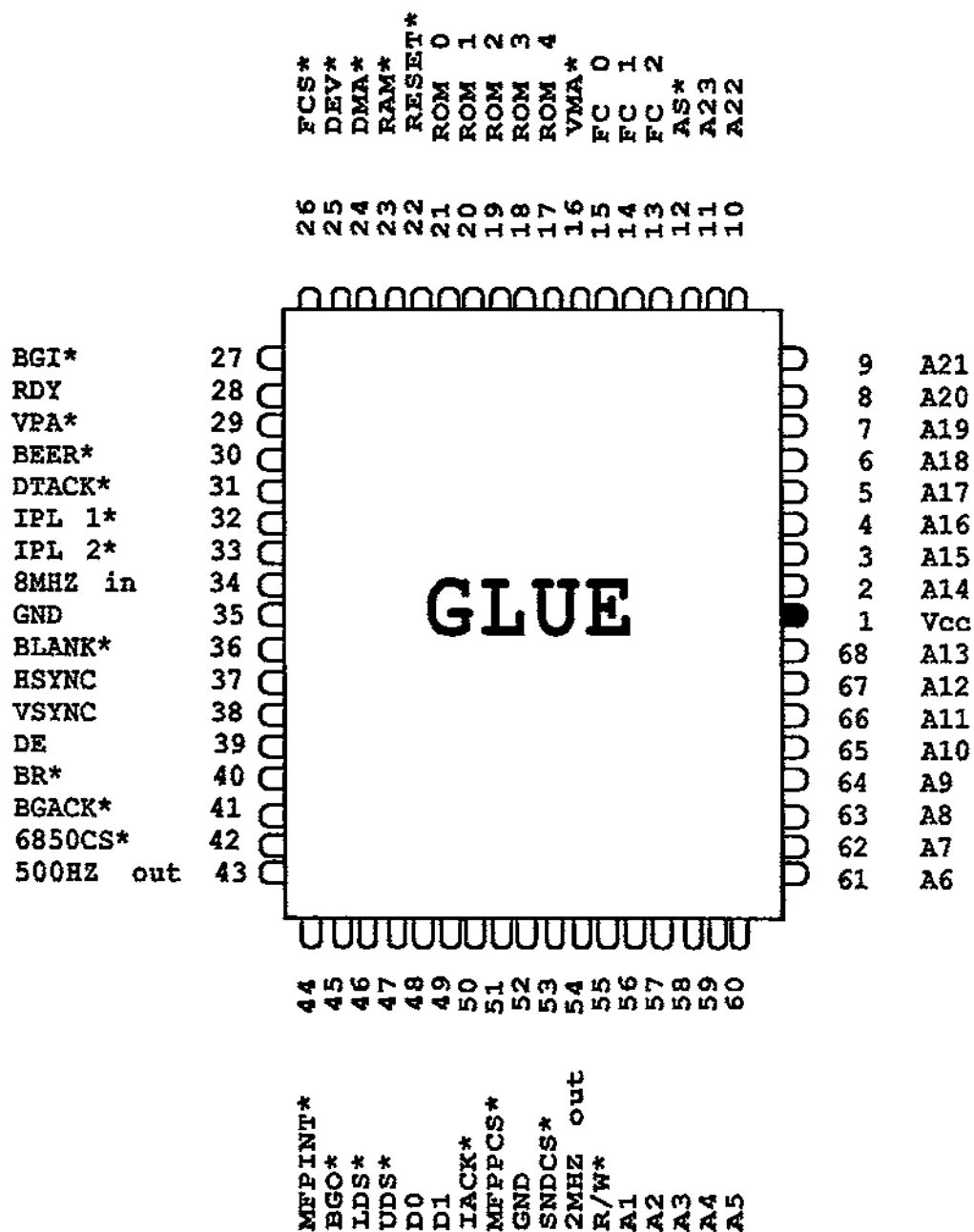
The system programmer can easily figure out which IC has which register. It is only essential to be able to recognize the address of the register, and how to control it. We're going to spend some time in this chapter exploring the pins of the individual ICs.

The most important IC of the "foursome" is GLUE. Its title speaks for the function -- a glue or paste. This IC, with its 68 pins, literally holds the entire system together, including decoding the address range and working the peripheral ICs.

Furthermore, the DMA handshake signals BR, BG and BGACK are produced/output by GLUE. The time point for DMA request is dictated by GLUE by the signal from the DMA controller. GLUE also has a BG (Bus Grant) input, as well as a BGO (Bus Grant Out).

The interrupt signal is produced by GLUE; in the ST, only IPL1 and IPL2 are used for this. Without other hardware, you can't use NMI (interrupt level 7). The pins MFPINT and IACK are used for interrupt control.

Figure 1.2-1 GLUE



The function code pins are guided by GLUE, where memory access tasks are performed (range testing and access authorization). Needless to say, the BERR signal is also handled by this chip. VPA is particularly important to the peripheral ICs and the appropriate select signals.

GLUE generates a timing frequency of 8 mHz. Frequencies between 2 mHz (sound chip's operating frequency) and 500 kHz (timing for keyboard and MIDI interface) can be produced.

HSYNC, VSYNC, BLANK and DE (Display Enable) are generated by GLUE for monitor operation. The synchronous timing can be switched on and off, and external sync-signals sent to the monitor. This will allow you to synchronize the ST's screen with a video camera.

The MMU also has a total of 68 pins. This IC performs three vital tasks. The most important task is coupling the multiplexed address bus of dynamic RAM with the processor's bus (handled by address lines A1 to A21). This gives us an address range totaling 4 megabytes. Dynamic RAM is controlled by RAS0, RAS1, CAS0L, CAS0H, CAS1L and CAS1H, as well as the multiplexed address bus on the MMU. DTACK, R/W, AS, LDS and UDS are also controlled by MMU.

We've already mentioned another important function of the MMU: it works with the SHIFTER to produce the video signal (the screen information is addressed in RAM, and SHIFTER conveys the information). Counters are incorporated in the MMU for this; a starting value is loaded, and within 500 nanoseconds, a word is addressed in memory and the information is sent over DCYC. The starting value of the video counter (and the screen memory position) can be shifted in 256-byte increments.

Another integrated counter in MMU, as mentioned earlier, is for addressing memory using the DMA. This counter begins with every DMA access (disk or hard disk), loading the address of the data being transferred. Every transfer automatically increments the counter.

The SHIFTER converts the information in video RAM into impulses readable on a monitor. Whether the ST is in 640 X 200 or 320 X 200 resolution, SHIFTER is involved.

Figure 1.2-2 MMU

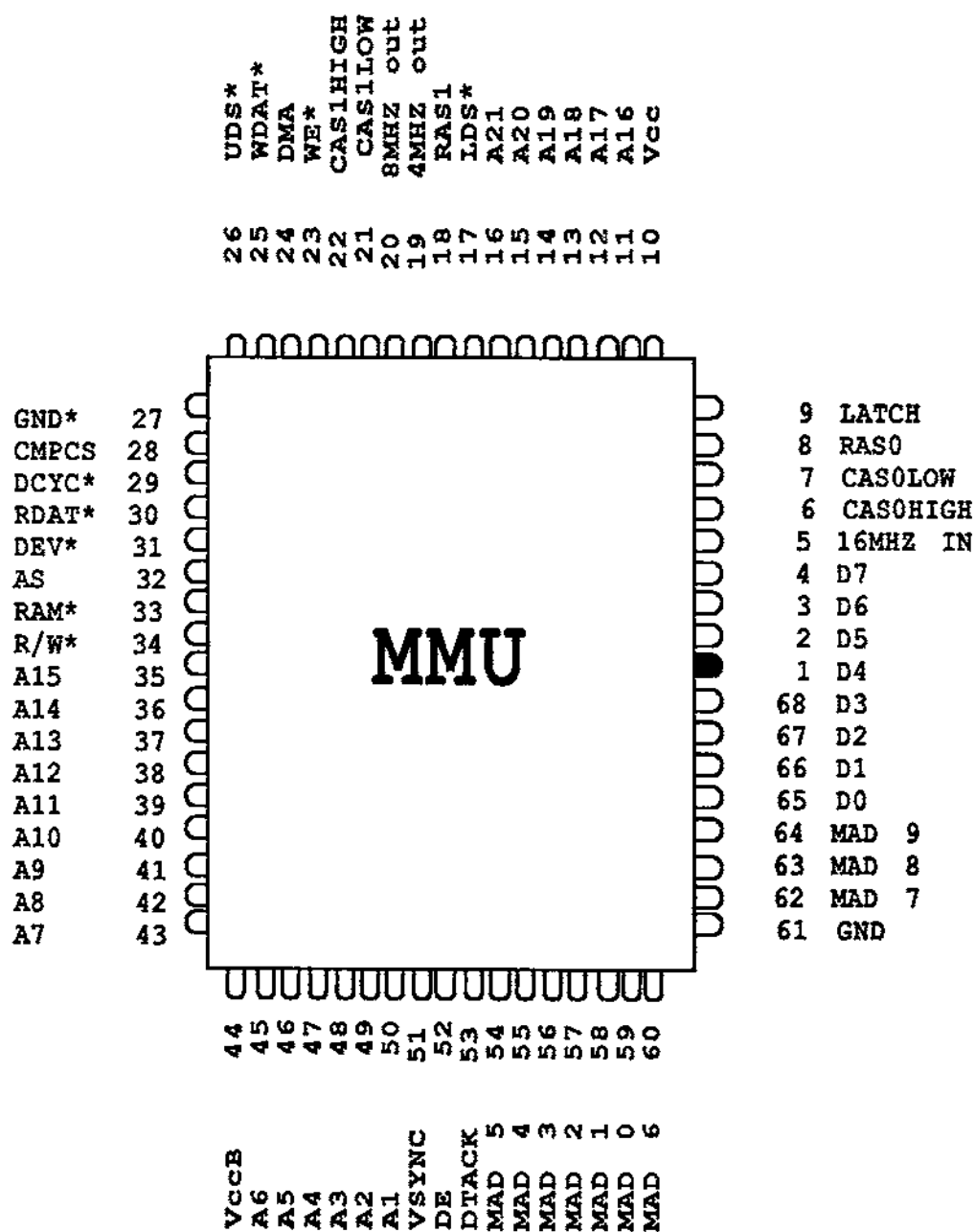
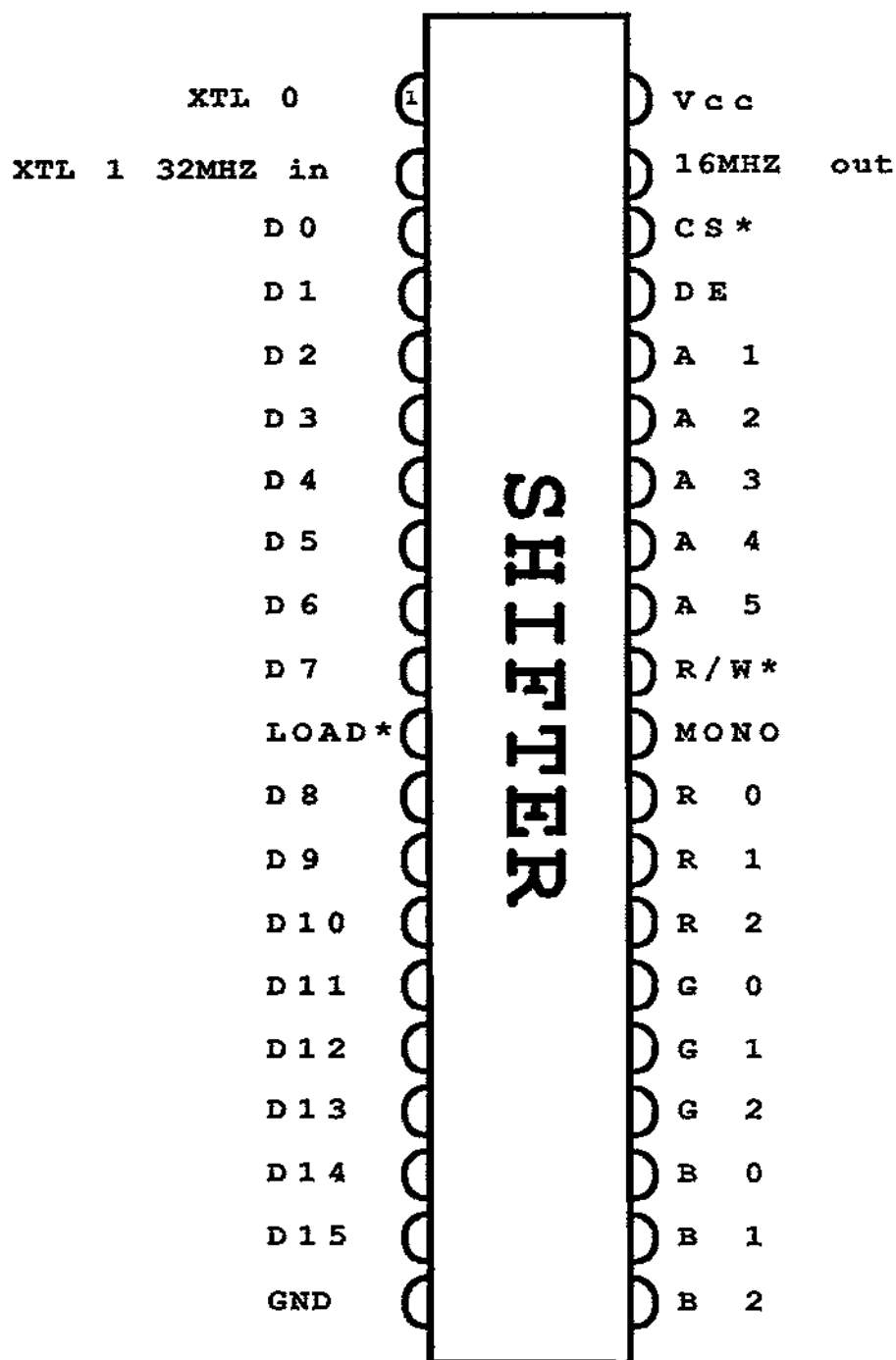


Figure 1.2-3 SHIFTER





The information from RAM is transferred to SHIFTER on the signal LOAD. A resolution of 640 X 400 points sends the video signal over the MONO connector. Since color is impossible in that mode, the RGB connection is rendered inactive. The other two resolutions set MONO output to inactive, since all screen information is being sent out the RGB connection in those cases.

The third color connection works together with external equipment as a digital/analog converter. Individual colors are sent out over different pins, to give us color on our monitor. Pins R1- R5 on the address bus make up the "palette registers". These registers contain the color values, which are placed in individual bit patterns. The 16 palette registers hold a total of 16 colors for 320 X 200 mode. Note, however, that since these are based on the "primary" colors red, green and blue, these colors can be adjusted in 8 steps of brightness, bringing the color total to 512.

The DMA controller is like SHIFTER, only in a 40-pin housing; it is used to oversee the floppy disk controller, the hard disk, and any other peripherals that are likely to appear.

The speed of data transfer using the floppy disk drive offers no problems to the processor. It's different with hard disks; data moves at such high speed that the 68000 has to send a "pause" over the 8 mHz frequency. This pace is made possible by the DMA.

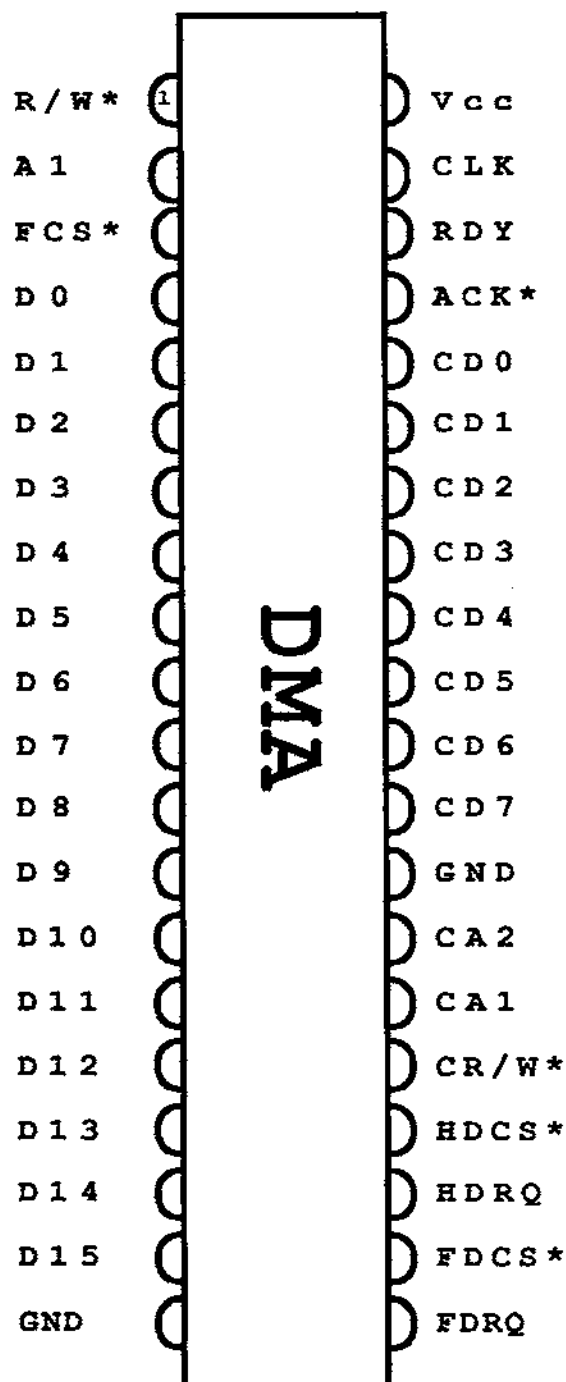
The DMA is joined to the processor's data bus to help transfer data. Two registers within the machine act as a bi-directional buffer for data through the DMA port; we'll discuss these registers later. One interesting point: The processor's 16-bit data bus is reduced to 8 bits for floppy/hard disk work. Data transfer automatically transfers two bytes per word.

The signals CA1, CA2, CR/W, FDCS and FDRQ manage the floppy disk controller. CA1 and CA2 are signals which the floppy disk controller (FDC) uses to select registers. CR/W determine the direction of data transfer from/to the FDC, and other peripherals connected to the DMA port.

The RDY signal communicated with GLUE (DMA-request) and MMU (address counter). This signal tells the DMA to transfer a word.

As you can see, these ICs work in close harmony with one another, and each would be almost useless on its own.

Figure 1.2-4 DMA



## 1.3 The WD 1772 Floppy Disk Controller

Although the 1772 from Western Digital has only 28 pins, this chip contains a complete floppy disk controller (FDC) with capabilities matching 40-pin controllers. This IC is software-compatible with the 1790/2790 series. Here are some of the 1772's features:

- Simple 5-volt current
- Built-in data separator
- Built-in copy compensation logic
- Single and double density
- Built-in motor controls

Although the user has his/her choice of disk format, e.g. sector length, number of sectors per track and number of tracks per diskette, the "normal" format is the optimum one for data transfer. So, Apple or Commodore diskettes can't be used.

Before going on to details of the FDC, let's take a moment to look at the 28 pins of this IC.

### 1.3.1 1772 Pins

These pins can be placed in three categories. The first group consists of the power connections.

**Vcc:**

+5 volts current.

**GND:**

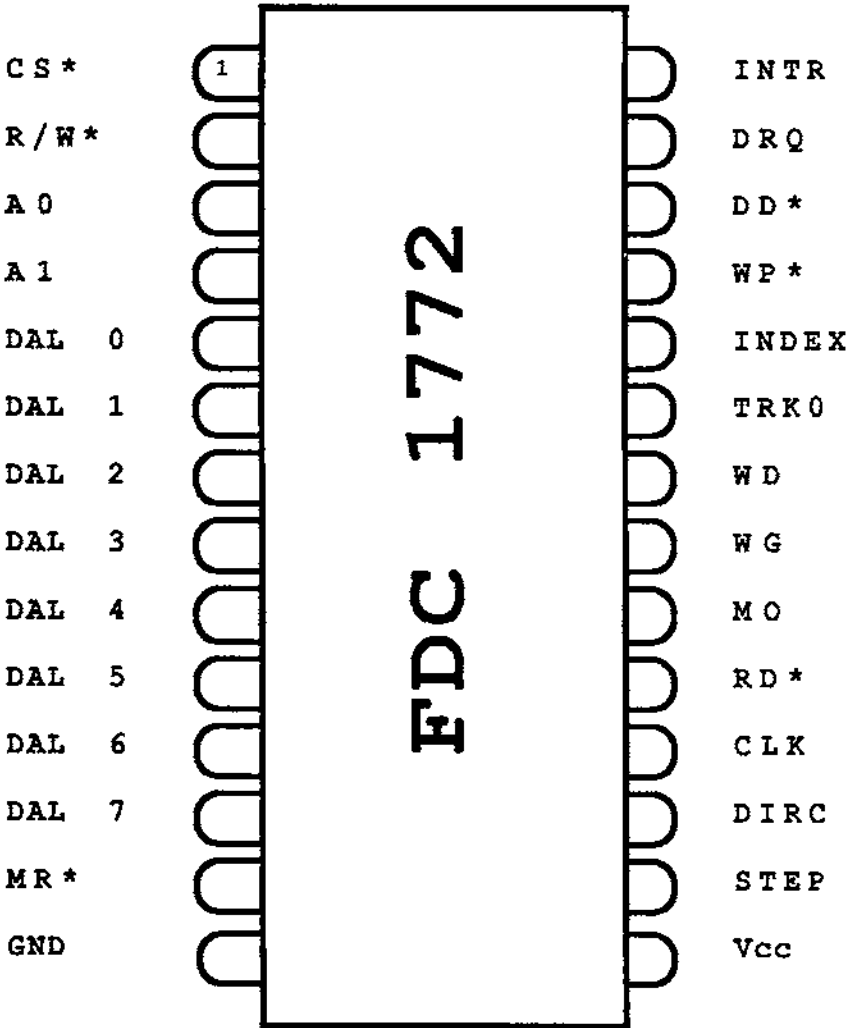
Ground connection.

**MR:**

Master reset. FDC reinitializes when this is low.

The second set are processor interface pins. These pins carry data between the processor and the FDC.

Figure 1.3-1 FDC 1772



**D0-D7:**

Eight-bit bi-directional bus; data, commands and status information go between FDC and system.

**CS:**

FDC can only access registers when this line is low.

**R/W:**

Read/Write. This pin states data direction. HIGH= read by FDC, LOW=write from FDC.

**A0,A1:**

These bits determine which register is accessed (in conjunction with R/W). The 1772 has a total of five registers which can both read and write to some degree. Other registers can only read OR write. Here is a table to show how the manufacturer designed them:

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.

**DRQ:**

Data Request. When this output is high, either the data register is full (from reading), and must be "dumped", or the data register is empty (writing), and can be refilled. This connection aids the DMA operation of the FDC.

**CLK:**

Clock. The clock signal counts only to the processor bus. An input frequency of 8 mHz must be on, for the FDC's internal timing to work.

The third group of signals make up the floppy interface.

**STEP:**

Sends an impulse for every step of the head motor.

**DIRC:**

Direction. This connection decides the direction of the head; high moves the head towards center of the diskette.

**RD:**

Read Data. Reads data from the diskette. This information contains both timing and data impulses -- it is sent to the internal data separator for division.

**MO:**

Motor On. Controls the disk drive motor, which is automatically started during read/write/whatever operations.

**WG:**

Write Gate. WG will be low before writing to diskette. Write logic would be impossible without this line.

**WD:**

Write Data. Sends serial data flow as data and timing impulses.

**TR00:**

Track 00. This moves read/write head to track 00. TR00 would be low in this case.

**IP:**

Index Pulse. The index pulses mark the physical beginnings of every track on a diskette. When formatting a disk, the FDC marks the start of each track before formatting the disk.

**WPRT:**

Write Protect. If the diskette is write-protected, this input will react.

**DDEN:**

Double Density Enable. This signal is confined to floppy disk control; it allows you to switch between single-density and double-density formats.

### 1.3.2 1772 Registers

**CR (Command Register):**

Commands are written in this 8-bit register. Commands should only be written in CR when no other command is under execution. Although the FDC only understands 11 commands, we actually have a large number of possibilities for these commands (we'll talk about those later).

**STR (Status Register):**

Gives different conditions of the FDC, coded into individual bits. Command writing depends on the meaning of each bit. The status register can only be read.

**TR (Track Register):**

Contains the current position of the read/write head. Every movement of the head raises or lowers the value of TR appropriately. Some commands will read the contents of TR, along with information read from the disk. The result affects the Status Register. TR can be read/written.

**SR (Sector Register):**

SR contains the number of sectors desired from read/write operations. Like TR, it can be used for either operation.

**DR (Data Register):**

DR is used for writing data to/ reading data from diskette.

### 1.3.3 Programming the FDC

Programming this chip is no big deal for a system programmer. Direct (and in most cases, unnecessary) programming is made somewhat harder AND drastically simpler by the DMA chip. The 11 FDC commands are divided into four types.

Type	Function
1	Restore, look for track 00
1	Seek, look for a track
1	Step, a track in previous direction
1	Step In, move head one track in (toward disk hub)
1	Step Out, move head one track out (toward edge of disk)
2	Read Sector
2	Write Sector
3	Read Address, read ID
3	Read Track, read entire track
3	Write Track, write entire track (format)
4	Force Interrupt

#### Type 1 Commands

These commands position the read/write head. The bit patterns of these five commands look like this:

	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



All five commands have several variable bits; bits R0 and R1 give the time between two step impulses. The possible combinations are:

R1	R0	STEP RATE
0	0	2 milliseconds
0	1	3 milliseconds
1	0	5 milliseconds
1	1	6 milliseconds

These bits must be set by the command bytes to the disk drive. The V-bit is the so-called "verify flag". When set, the drive performs an automatic verify after every head movement. The H-bit contains the spin-up sequence. The system delays disk access until the disk motor has reached 300 rpm. If the H-bit is cleared, the FDC checks for activation of the motor-on pins. When the motor is off, this pin will be set high (motor on), and the FDC waits for 6 index impulses before executing the command. If the motor is already running, then there will be no waiting time.

The three different step commands have bit 4 designated a U- bit. Every step and change of the head appears here.

### Type 2 Commands

These commands deal with reading and writing sectors. They also have individual bits with special meanings.

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

The H-bit is the previously described start-up bit. When the E-bit is set, the FDC waits 30 milliseconds before starting the command. This delay is important for some disk drives, since it takes time for the head to change tracks. When the E-bit reads null, the command will run immediately.

The M-bit determines whether one or several sectors are read one after another. On a null reading, only one sector will be read from/written to. Multi-sector reading sets the bit, and the FDC increments the counter at each new sector read.

Bits 0 and 1 must be cleared for sector reading. Writing has its own special meaning: the A0 bit conveys to bit 0 whether a cleared or normal data

address mark is to be written. Most operating systems don't use this option (a normal data address mark is written).

The P-bit (bit 1) dictates whether pre-compensation for writing data is turned on or off. Pre-compensation is normally set on; it supplies a higher degree of protection to the inner tracks of a diskette.

### Type 3 Commands

Read Address gives program information about the next ID field on the diskette. This ID field describes track, sector, disk side and sector length. Read Track gives all bytes written to a formatted diskette, and the data "between sectors". Write Track formats a track for data storage. Here are the bit patterns for these commands:

BIT	7	6	5	4	3	2	1	0
Read Address	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

The H- and E-bits also belong to the Type 2 command set (spin-up and head-settle time). The P-bit has the same function as in writing sectors.

### Type 4 Commands

There's only one command in this set: Force Interrupt. This command can work with individual bits during another FDC command. When this command comes into play, whatever command was currently running is ended.

BIT	7	6	5	4	3	2	1	0
Force Interrupt	1	1	0	1	I3	I2	I1	I0

Bits I0-I3 present the conditions under which the interrupt is pressed. I0 and I1 have no meaning to the 1772, and remain low. If I2 is set, an interrupt will be produced with every index impulse. This allows for software controlled disk rotation. If I3 is set, an interrupt is forced immediately, and the currently-running command ends. When all bits are null, the command ends without interruption.

## 1.4 The MFP 68901

MFP is the abbreviation for Multi-Function Peripheral. This name is no exaggeration; wait until you see what it can do! Here's a brief list of the most noteworthy features:

- 8-bit parallel port
- Data direction of every port bit is individually programmable
- Port bits usable as interrupt input
- 16 possible interrupt sources
- Four universal timers
- Built-in serial interface

### 1.4.1 The 68901 Connections

The 48 pins of the MFP are set apart in function groups. The first function group is the power connection set:

**GND, Vcc, CLK:**

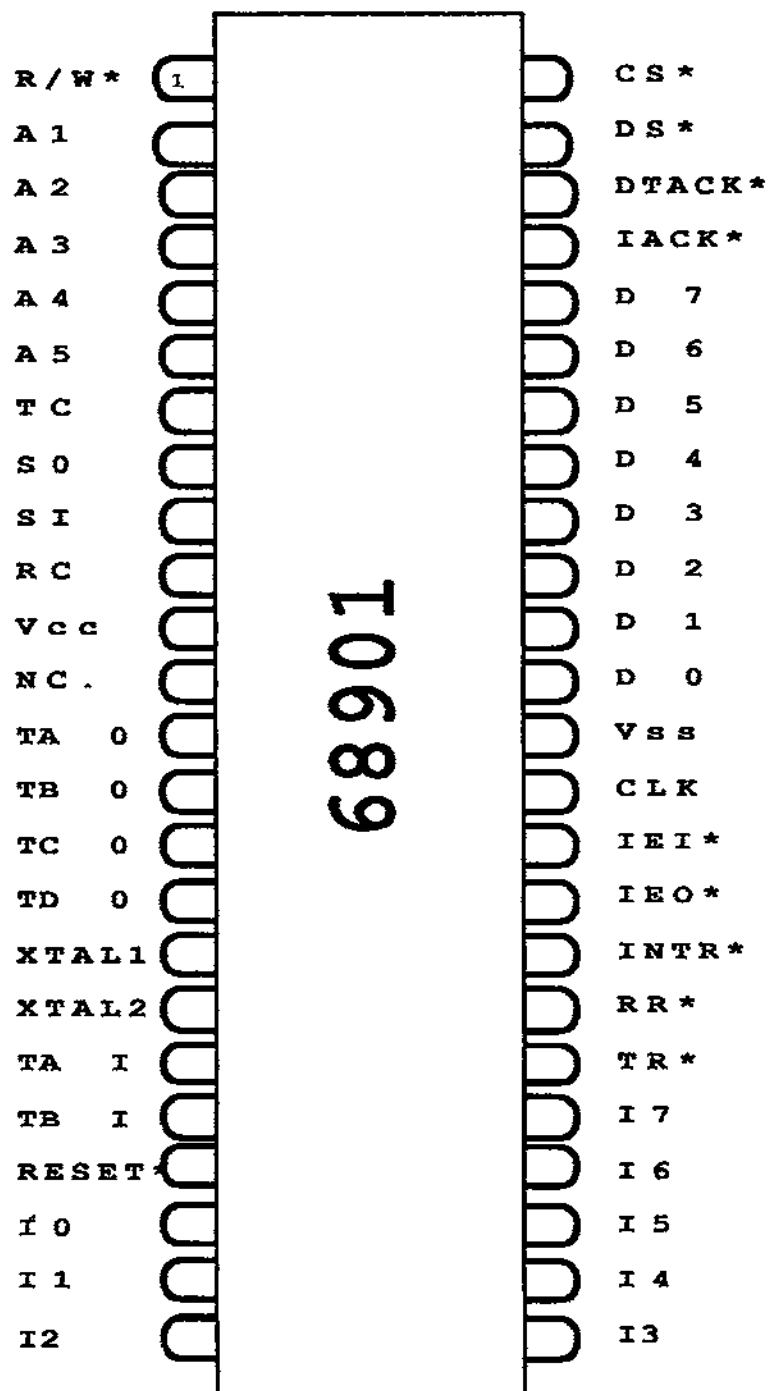
Vcc and GND carry voltage to and from the MFP. CLK is the clock input; this clock signal must not interfere with the system timer of the processor. The ST's MFP operates at a frequency of 4 mHz.

Communication with the data bus of the processor is maintained with D0-D7, DTACK, RS1-RS5 and RESET.

**D0-D7:**

These bi-directional pins normally work with the 8 lowest data bits of the 68000. It is also possible to connect with D8 through D15, but it's impossible to produce non-auto interrupts. Thus, interrupt vectors travel along the low order 8 data bits.

Figure 1.4-1 MFP 68901



**CS (Chip Select):**

This line is necessary to communication with the MFP. CS is active when low.

**DS (Data Strobe):**

This pin works with either LDS or UDS on the processor. Depending on the signal, MFP will operate either the lower or upper half of the data bus.

**DTACK (Data Transfer ACKnowledge):**

This signal shows the status of the bus cycle of the processor (read or write).

**RS1-RS5 (Register Select):**

These pins normally connect with to the bottom five address lines of the processor, and serve to choose from the 24 internal registers.

**RESET:**

If this pin is low for at least 2 microseconds, the MFP initializes. This occurs on power-up and a system reset.

The next group of signals cover interrupt connections (IRQ, IACK, IEI and IEO).

**IRQ (Interrupt ReQuest):**

IRQ will be low when an interrupt is triggered in the MFP. This informs the processor of interrupts.

**IACK (Interrupt ACKnowledge):**

On an interrupt (IRQ and IEI), the MFP sends a low signal over IACK and DS on the data lines. Since 16 different interrupt sources are available, this makes handling interrupts much simpler.

**IEI, IEO (Interrupt Enable In/ Out):**

These two lines permit daisy-chaining of several MFPs, and determine MFP priority by their positioning in this chain. IEI would work through the MFP with the highest priority. IEO of the second MFP would remain unswitched. On an interrupt, a signal is sent over IACK, and the first MFP in the chain will acknowledge with a high IEO.

Next, we'll look at the eight I/O lines.

**IO0-7 (Input/Output):**

These pins use one or all normal I/O lines. The data direction of each port bit is set up in a data direction register of its own. In addition, though, every port bit can be programmed to be an interrupt input.

The timer pins make up yet another group of connections:

**XTAL1,2 (Timer Clock Crystal):**

A quartz crystal can be connected to these lines to deliver a working frequency for the four timers.

**TAL,TBI (Timer Input):**

Timers A and B can not only be used as real counters differently from timers C and D with the frequency from XTAL1 and 2, but can also be set up for event counting and impulse width measurement. In both these cases, an external signal (Timer Input) must be used.

**TAO,TBO,TCO,TDO (Timer Output):**

Every timer can send out its status on each peg (from 01 to 00). Each impulse is equal to 01.

The second-to-last set of signals are the connections to the universal serial interface. The built-in full duplex of the MFP can be run synchronously or asynchronously, and in different sending and receiving baud rates.

**SI (Serial Input):**

An incoming bit current will go up the SI input.

**SO (Serial Output):**

Outgoing bit voltage (reverse of SI).

**RC (Receiver Clock):**

Transfer speed of incoming data is determined by the frequency of this input; the source of this signal can, for example, be one of the four timers.

**TC (Transmitter Clock):**

Similar to RC, but for adjusting the baud-rate of data being transmitted.

The final group of signals aren't used in the Atari ST. They are necessary when the serial interface is operated by the DMA.

**RR (Receiver Ready):**

This pin gives the status of the receiving data registers. If a character is completely received, this pin sends current.

**TR (Transmitter Ready):**

This line performs a similar function for the sender section of the serial interface. Low tells the DMA controller that a new character in the MFP must be sent.

## 1.4.2 The MFP Registers

As we've already mentioned, the 68901 has a total of 24 different registers. This large number, together with the logical arrangement, makes programming the MFP much easier.

**Reg 1 GPIP, General Purpose I/O Interrupt Port**

This is the data register for the 8-bit ports, where data from the port bits is sent and read.

**Reg 2 AER, Active Edge Register**

When port bits are used for input, this register dictates whether the interrupt will be a low-high- or high-low conversion. Zero is used in the high-low change, one for low-high.

**Reg 3 DDR, Data Direction Register**

We've already said that the data direction of individual port bits can be fixed by the user. When a DDR bit equals 0, the corresponding pin becomes an input, and 1 makes it an output. Port bit positions are influenced by AER and DDR bits.

**Reg 4,5 IERA,IERB, Interrupt Enable Register**

Every interrupt source of the MFP can be separately switched on and off. With a total of 16 sources, two 8-bit registers are needed to control them. If a 1 has been written to IERA or IERB, the corresponding channel is enabled (turned on). Conversely, a zero disables the channel. If it comes upon a closed channel caused by an interrupt, the MFP will completely ignore it. The following table shows which bit is coordinated with which interrupt occurrence:

**IERA**

Bit 7: I/O port bit 7 (highest priority)  
Bit 6: I/O port bit 6  
Bit 5: Timer A  
Bit 4: Receive buffer full  
Bit 3: Receive error  
Bit 2: Sender buffer empty  
Bit 1: Sender error  
Bit 0: Timer B

**IERB**

Bit 7: I/O port bit 5  
Bit 6: I/O port bit 4  
Bit 5: Timer C  
Bit 4: Timer D  
Bit 3: I/O port bit 3  
Bit 2: I/O port bit 2  
Bit 1: I/O port bit 1  
Bit 0: I/O port bit 0, lowest priority

This arrangement applies to the IP-, IM- and IS-registers discussed below.

**Reg 6,7 IPRA,IPRB, Interrupt Pending Register**

When an interrupt occurs on an open channel, the appropriate bit in the Interrupt Pending Register is set to 1. When working with a system that allows vector creation, this bit will be automatically cleared when the MFP puts the vector number on the data bus. If this possibility doesn't exist, the IPR must be cleared using software. To clear a bit, a byte in the MFP will show the location of the specific bit.

The bit arrangement of the IPR is shown in the table for registers 4 and 5 (see above).



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

The function of these registers is somewhat complicated, and depends upon bit 3 of register 12. This bit is an S-bit, which determines whether the 68901 is working in "Software End-of-Interrupt" mode (SEI) or in "Automatic End-of-Interrupt" mode (AEI). AEI mode clears the IPR (Interrupt Pending Bit), when the processor gets the vector number from the MFP during an IACK cycle. The appropriate In-Service bit is cleared at the same time. Now a new interrupt can occur, even when the previous interrupt hasn't finished its work.

SEI mode sets the corresponding ISR-bit when the vector number of the interrupt is requested by the processor. At the interrupt routine's end, the bit designated within the MFP must be cleared. As long as the Interrupt In-Service bit is set, all interrupts of lower priority are masked out by the MFP. Once the Pending-bit of the active channel is cleared, the same sort of interrupt can occur a second time, and interrupts of lesser priority can occur as well.

**Reg 10,11 IMRA,IMRB Interrupt Mask Register**

Individual interrupt sources switched on by IER can be masked with the help of this register. That means that the interrupt is recognized from within and is signalled in the IPR, even if the IRQ line remains high.

**Reg 12 VR Vector Register**

In the cases of interrupts, the 68901 can generate a vector number corresponding to the interrupt source requested by the processor during an Interrupt Acknowledge Cycle. All 16 interrupt channels have their own vectors, with their priorities coded into the bottom four bits of the vector number (the upper four bits of the vector are copied from the vector register). These bits must be set into VR, therefore.

Bit 3 of VR is the previously mentioned S-bit. If this bit is set (like in the ST), then the MFP operates in "Software End-of-Interrupt" mode; a cleared bit puts the system into "Automatic End-of-Interrupt" mode.

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

Before proceeding with these registers, we should talk for a moment about the timer. Timers A and B are both identical. Every timer consists of a data register, a programmable feature and an 8-bit count-down counter. Contents of the counters will decrease by one every impulse. When the counter stands at 01, the next impulse changes the corresponding timer to the output of its pins. At the same time, the value of the timer data register is loaded into the timer. If this channel is set by the IER bit, the interrupt will be requested. The source of the timer beats will usually be those quartz frequencies from XTAL1 and 2. This operating mode is called delay mode, and is available to timers C and D.

Timers A and B can also be fed external impulses using timer inputs TAI and TBI (in event count mode). The maximum frequency on timer inputs should not surpass 1/4 of the MFP's operating frequency (that is, 1 mHz).

Another peculiarity of this operating mode is the fact that the timer inputs for the interrupts are I/O pins 13 and 14. By programming the corresponding bits in the AER, a pin-jump can be used by the timer inputs to request an interrupt. TAI is joined with pin 13, TBI by pin 14. Pins 13 and 14 can also be used as I/O lines without interrupt capability.

Timers A and B have yet a third operating mode (pulse-length measurement). This is similar to Delay Mode, with the difference that the timer can be turned on and off with TAI and TBI. Also, when pins 13 and 14 are used, the AER-bits can determine whether the timer inputs are high or low. If, say, AER-bit 4 is set, the counter works when TAI is high. When TAI changes to low, an interrupt is created.

Now we come to TACR and TBCR. Both registers only use the fifth through eighth bits. Bits 0 to 3 determine the operating mode of each timer:

BIT 3 2 1 0      Function

0 0 0 0	Timer stop, no function executed
0 0 0 1	Delay mode, subdivider divides by 4
0 0 1 0	Delay mode, subdivider divides by 10
0 0 1 1	Delay mode, subdivider divides by 16
0 0 1 1	Delay mode, subdivider divides by 16
0 1 0 0	Delay mode, subdivider divides by 50
0 1 0 1	Delay mode, subdivider divides by 64
0 1 1 0	Delay mode, subdivider divides by 100
0 1 1 1	Delay mode, subdivider divides by 200
1 0 0 0	Event Count Mode
1 0 0 1	Pulse extension mode, subdivider divides by 4
1 0 1 0	Pulse extension mode, subdivider divides by 10
1 0 1 1	Pulse extension mode, subdivider divides by 16
1 1 0 0	Pulse extension mode, subdivider divides by 50
1 1 0 1	Pulse extension mode, subdivider divides by 64
1 1 1 0	Pulse extension mode, subdivider divides by 100
1 1 1 1	Pulse extension mode, subdivider divides by 200

Bit 4 of the Timer Control Register has a particular function. This bit can produce a low reading for the timer being used with it at any time. However, it will immediately go high when the timer runs.

### Reg 15 TCDCR Timers C and D Control Register

Timers C and D are available only in delay mode; thus, one byte controls both timers. The control information is programmed into the lower three bits of the nibbles (four-bit halves). Bits 0 and 2 arrange Timer D, Timer C is influenced by bits 4 and 6. Bits 3 and 7 in this register have no function.

Bit	2 1 0	Function - Timer D
Bit	6 5 4	Function - Timer C
	0 0 0	Timer Stop
	0 0 1	Delay Mode, division by 4
	0 1 0	Delay Mode, division by 10
	0 1 1	Delay Mode, division by 16
	1 0 0	Delay Mode, division by 50
	1 0 1	Delay Mode, division by 64
	1 1 0	Delay Mode, division by 100
	1 1 1	Delay Mode, division by 200

**Reg 16-19 TADR,TBDR,TCDR,TDDR Timer Data Registers**

The four Timer Data Registers are loaded with a value from the counter. When a condition of 01 is reached, an impulse occurs. A continuous countdown will stem from this value.

**Reg 20 SCR Synchronous Character Register**

A value will be written to this register by synchronous data transfer, so that the receiver of the data will be alerted. When synchronous mode is chosen, all characters received will be stored in the SCR, after first being put into the receive buffer.

**Reg 21 UCR,USART Control Register**

USART is short for Universal Synchronous/Asynchronous Receiver/Transmitter. The UCR allows you to set all the operating parameters for the interfaces. Parameters can also be coded in with the timers.

Bit 0 : unused

Bit 1 : 0=Odd parity  
1=Even parity

Bit 2 : 0=No parity (bit 1 is ignored)  
1=Parity according to bit 1

Bits 3,4 : These bits control the number of start- and stopbits and the format desired.

Bit	4	3	Start	Stop	Format
	0	0	0	0	Synchronous
	0	1	1	1	Asynchronous
	1	0	1	1,5	Asynchronous
	1	1	1	2	Asynchronous

Bits 5,6 : These bits give the "wordlength" of the data bits to be transferred.

Bits	6	5	Word length
	0	0	8 bits
	0	1	7 bits
	1	0	6 bits
	1	1	5 bits

Bit 7 : 0=Frequency from TC and RC  
directly used as transfer  
frequency (used only for  
synchronous transfer)  
1=Frequency in TC and RC  
internally divided by 16.

## **Reg 22 RSR Receiver Status Register**

The RSR gives information concerning the conditions of all receivers. Again, the different conditions are coded into individual bits.

### **Bit 0 Receiver Enable Bit**

When this bit is cleared, receipt is immediately turned off. All flags in RSR are automatically cleared. A set bit means that the receiver is behaving normally.

### **Bit 1 Synchronous Strip Enable**

This bit allows synchronous data transfer to determine whether or not a character in the SCR is identical to a character in the receive buffer.

### **Bit 2 Match/Character in Progress**

When in synchronous transfer format, this bit signals that a character identical with the SCR byte would be received. In asynchronous mode, this bit is set as soon as the startbit is recognized. A stopbit automatically clears this bit.

### **Bit 3 Found - Search/Break Detected**

This bit is set in synchronous transfer format, when a character received coincides with one stored in the SCR. This condition can be treated as an interrupt over the receiver's error channel. Asynchronous mode will cause the bit to set when a BREAK is received. The break condition is fulfilled when only zeroes are received following a startbit. To distinguish between a BREAK from a "real" null, this line should be low.

### **Bit 4 Frame Error**

A frame error occurs when a byte received is not a null, but the stopbit of the byte IS a null.

**Bit 5 Parity Error**

The condition of this bit gives information as to whether parity on the last received character was correct. If the parity test is off, the PE bit is untouched.

**Bit 6 Overrun Error**

This bit will be set when a complete character is in the receiver floating range but not read into the receive buffer. This error can be operated as an interrupt.

**Bit 7 Buffer Full**

This bit is set when a character is transferred from the floating register to the receive buffer. As soon as the processor reads the byte, the bit is cleared.

**Reg 23 TSR Transmitter Status Register**

Whereas the RSR sends receiver information, the TSR handles transmission information.

**Bit 0 Transmitter Enable**

The sending section is completely shut off when this bit is cleared. At the same time the End-bit is cleared and the UE-bit is set (see below). The output to the receiver is set in the corresponding H- and L-bits.

**Bits 1,2 High- and Low-bit**

These bits let the programmer decide which mode of output the switched-off transmitter will take on. If both bits are cleared, the output is high. High-bit only will create high output; low-bit, low output. Both bits on will switch on loop-back-mode. This state loops the output from the transmitter with receiver input. The output itself is on the high-pin.

**Bit 3 Break**

The break-bit has no function in synchronous data transfer. In asynchronous mode, though, a break condition is sent when the bit is set.

**Bit 4 End of Transmission**

If the sender is switched off during running transmission, the end-bit will be set as soon as the current character has been sent in its entirety. When no character is sent, the bit is immediately set.

**Bit 5 Auto Turnaround**

When this bit is set, the receiver is automatically switched on when the transmitter is off, and a character will eventually be sent.

**Bit 6 Underrun Error**

This bit is switched on when a character in the sender floating register will be sent, before a new character is written into the send buffer.

**Bit 7 Buffer Empty**

This bit will be set when a character from the send buffer will be transferred to the floating register. The bit is cleared when new data is written to the send buffer.

**Reg 24 UDR, USART Data Register**

Send/receive data is sent over this register. Writing sends data in the send buffer, reading gives you the contents of the receive buffer.

## 1.5 The 6850 ACIAs

ACIA is short for "Asynchronous Communications Interface Adapter". This 24-pin IC has all the components necessary for operating a serial interface, as well as error-recognizing and data-formatting capabilities. Originally for 6800-based computers, this chip can be easily tailored for 6502 and 68000 systems. The ST has two of these chips. One of them communicates with the keyboard, mouse, joystick ports, and runs the clock. Keyboard data travels over a serial interface to the 68000 chip. The second ACIA is used for operating the MIDI interface.

Parameter changes in the keyboard ACIA are not recommended: The connection between keyboard and ST can be easily disrupted. The MIDI interface is another story, though -- we can create all sorts of practical applications. Incidentally, nowhere else has it been mentioned that the MIDI connections can be used for other purposes. One idea would be to use the MIDI interfaces of several STs to link them together (for schools or offices, for example).

### 1.5.1 The Pins of the 6850

For those of you readers who aren't very well-acquainted with the principles of serial data transfer, we've included some fairly detailed descriptions in the pin layout which follows.

#### **Vss**

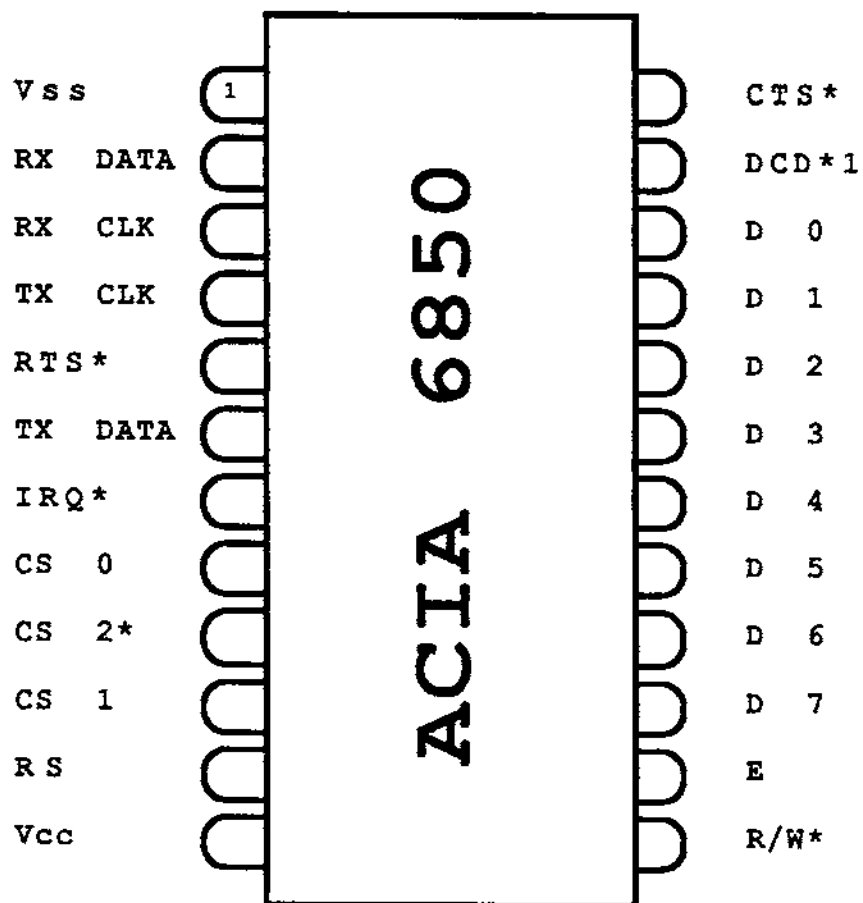
This connection is the "ground wire" of the IC.

#### **RX DATA Receive Data**

This pin receives data; a start-bit must precede the least significant data-bit before receipt.



Figure 1.5-1 ACIA 6850



**RX CLK Receive Clock**

This pin signal determines baud-rate (speed at which the data is received), and is synchronize to the incoming data. The frequency of RX CLK is patterned after the desired transfer speed and after the internally programmed division rate.

**TX CLK Transmitter Clock**

Like RX CLK, only used for transmission speed.

**RTS Request To Send**

This output signals the processor whether the 6850 is low or high; mostly used for controlling data transfer. A low output will, for example, signal a modem that the computer is ready to transmit.

**TX DATA Transmitter Data**

This pin sends data bit-wise (serially) from the computer.

**IRQ Interrupt Request**

Different circumstances set this pin low, signaling the 68000 processor. Possible conditions include completed transmission or receipt of a character.

**CS 0,1,2 Chip Select**

These three lines are needed for ACIA selection. The relatively high number of CS signals help minimize the amount of hardware needed for address decoding, particularly in smaller computer systems.

**RS Register Select**

This signal communicates with internal registers, and works closely with the R/W signal. We shall talk about these registers later.

**Vcc Voltage**

This pin is required of all ICs -- this pin gets an operating voltage of 5V.

**R/W Read/Write**

This tells the processor the "direction" of data traveling through the ACIA. A high signal tells the processor to read data, and low writes data in the 6850.

**E Enable**

The E-signal determines the time of reading/writing. All read/write processes with this signal must be synchronous.

**D0 - D7 Data**

These data lines are connected to those of the 68000. Until the ACIA is accessed, these bidirectional lines are all high.

**DCD Data Carrier Detect**

A modem control signal, which detects incoming data. When DCD is high, serial data cannot be received.

**CTS Clear To Send**

CTS answers the computer on the signal RTS. Data transmission is possible only when this pin is low.

## 1.5.2 The Registers of the 6850

The 6850 has four different registers. Two of these are read only. Two of them are write only. These registers are distinguished by R/W and RS, after the table below:

R/W	RS	Register	Access
0	0	Control Register	write
0	1	Sender Register	write
1	0	Status Register	read
1	1	Receive Register	read

The sender/receiver registers (also known as the RX- and TX- buffers) are for data transfer. When receiving is possible, the incoming bits are put in a shift register. Once the specified number of bits has arrived, the contents of the shift register are transferred to the TX buffer. The sender works in much the same way, only in the reverse direction (RX buffer to sender shift register).

## The Control Register

The eight-bit control register determines internal operations. To solve the problem of controlling diverse functions with one byte, single bits are set up as below:

### **CR 0,1**

These bits determine by which factor the transmitter and receiver clock will be divided. These bits also are joined with a master reset function. The 6850 has no separate reset line, so it must be accomplished through software.

CR1	CR0	
0	0	RXCLK/TXCLK without division
0	1	RXCLK/TXCLK by 16 (for MIDI)
1	0	RXCLK/TXCLK by 64 (for keyboard)
1	1	Master RESET

### **CR 2,3,4**

These so-called Word Select bits tell whether 7 or 8 data-bits are involved; whether 1 or 2 stop-bits are transferred; and the type of parity.

CR4	CR3	CR2	
0	0	0	7 databits, 2 stopbits, even parity
0	0	1	7 databits, 2 stopbits, odd parity
0	1	0	7 databits, 1 stopbit, even parity
0	1	1	7 databits, 1 stopbit, odd parity
1	0	0	8 databits, 2 stopbit, no parity
1	0	1	8 databits, 1 stopbit, no parity
1	1	0	8 databits, 1 stopbit, even parity
1	1	1	8 databits, 1 stopbit, odd parity

### **CR 6,5**

These Transmitter Control bits set the RTS output pin, and allow or prevent an interrupt through the ACIA when the send register is emptied. Also, BREAK signals can be sent over the serial output by this line. A BREAK signal is nothing more than a long sequence of 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 sent

**CR 7**

The Receiver Interrupt Enable bit determines whether the receiver interrupt will be on. An interrupt can be caused by the DCD line changing from low to high, or by the receiver data buffer filling. Besides that, an interrupt can occur from an **OVERRUN** (a received character isn't properly read from the processor).

CR7	
0	Interrupt disabled
1	Interrupt enabled

**The Status Register**

The Status Register gives information about the status of the chip. It also has its information coded into individual bytes.

**SR0**

When this bit is high, the RX data register is full. The byte must be read before a new character can be received (otherwise an **OVERRUN** happens).

**SR1**

This bit reflects the status of the TX data buffer. An empty register sets the bit.

**SR2**

A low-high change on pin DCD sets SR2. If the receiver interrupt is allowable, the IRQ will be cancelled. The bit is cleared when the status register and the receiver register are read. This also cancels the IRQ. SR2 register remains high if the signal on the DCD pin is still high; SR2 registers low if DCD becomes low.

**SR3**

This line shows the status of CTS. This signal cannot be altered by a master reset, or by ACIA programming.

**SR4**

Shows "Frame errors". Frame errors are when no stop-bit is recognized in receiver switching. It can be set with every new character.

**SR5**

This bit displays the previously mentioned OVERRUN condition. SR5 is reset when the RX buffer is read.

**SR6**

This bit recognizes whether the parity of a received character is correct. The bit is set on an error.

**SR 7**

This signals the state of the IRQ pins; this bit makes it possible to switch several IRQ lines on one interrupt input. In cases where an interrupt is program-generated, SR7 can tell which IC cut off the interrupt.

**The ACIAs in the ST**

The ACIAs have lots of extras unnecessary to the ST. In fact, CTS, DCD and RTS are not connected.

The keyboard ACIA lies at the addresses \$FFFC00 and \$FFFC02. Built-in parameters are: 8-bit word, 1 stopbit, no parity, 7812.5 baud (500 kHz/64).

The parameters are the same for the MIDI chip, EXCEPT for the baud rate, which runs at 31250 baud (500 kHz/16).

## 1.6 The YM-2149 Sound Generator

The Yamaha YM-2149, a PSG (programmable sound generator) in the same family as the General Instruments AY-3-8190, is a first-class sound synthesis chip. It was developed to produce sound for arcade games. The PSG also has remarkable capabilities for generating/altering sounds. Additionally, the PSG can be easily controlled by joysticks, the computer keyboard, or external keyboard switching. The PSG has two bidirectional 8-bit parallel ports. Here's some general data on the YM-2149:

- three independently programmable tone generators
- a programmable noise generator
- complete software-controlled analog output
- programmable mixer for tone/noise
- 15 logarithmically raised volume levels
- programmable envelopes (ASDR)
- two bidirectional 8-bit data ports
- TTL-compatible
- simple 5-volt power

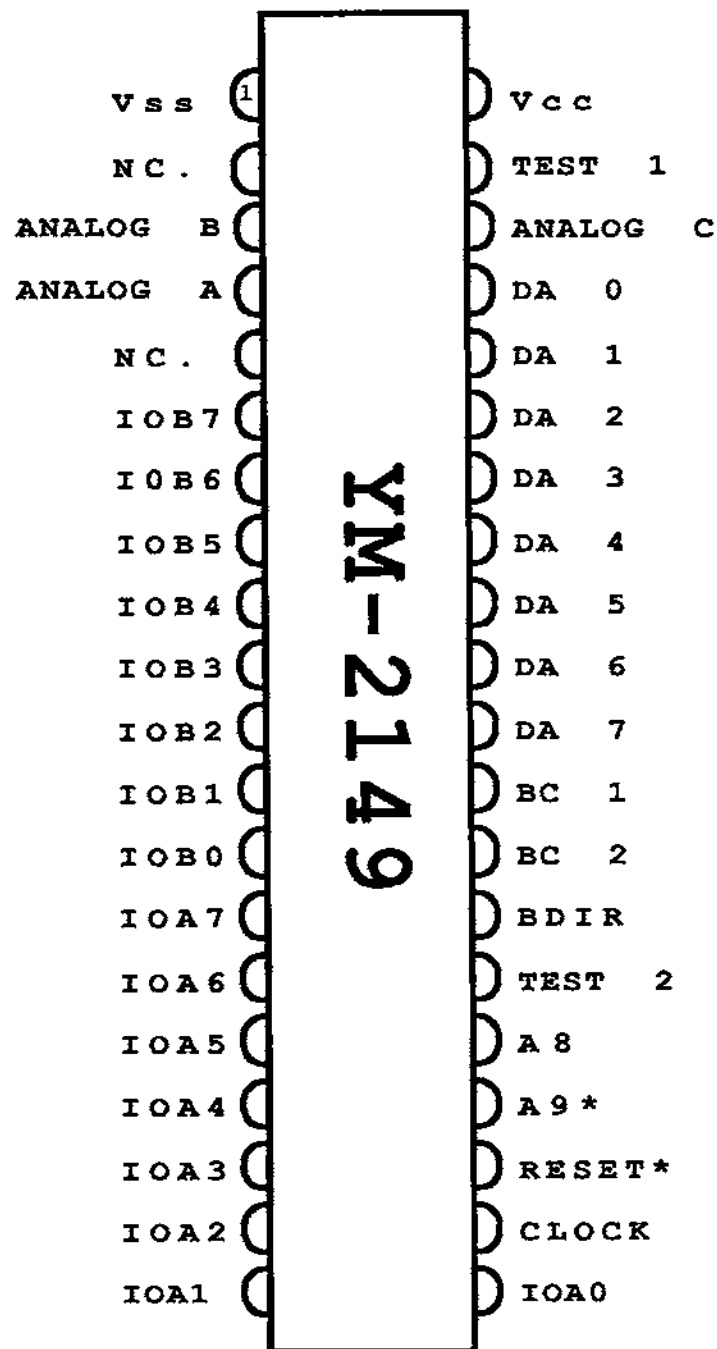
The YM-2149 has a total of 16 registers. All sound capabilities are controlled by these registers.

The PSG has several "functional blocks" each with its own job. The tone generator block produces a square-wave sound by means of a time signal. The noise generator block produces a frequency-modulated square-wave signal, whose pulse-width simulates a noise generator. The mixer couples the three tone generators' output with the noise signal. The channels may be coupled by programming.

The amplitude control block controls the output volume of the three channels with the volume registers; or creates envelopes (Attack, Decay, Sustain, Release, or ADSR), which controls the volume and alters the sound quality.

The D/A converter translates the volume and envelope information into digital form, for external use. Finally one function block controls the two I/O ports.

Figure 1.6-1 Sound chip YM-2149





### 1.6.1 Sound Chip Pins

**Vss:**

This is the PSG ground connection.

**NC.:**

Not used.

**ANALOG B:**

This is the channel B output. Maximum output voltage is 1 vss.

**ANALOG A:**

Works like pin 3, but for channel A.

**NC.:**

Not used.

**IOB7 - 0:**

The IOB connections make up one of the two 8-bit ports on the chip. These pins can be used for either input or output. Mixed operation (input and output combined) is impossible within one port, however both ports are independent of one another.

**IOA7 - 0:**

Like IOB, but for port A.

**CLOCK:**

All tone frequencies are divided by this signal. This signal operates at a frequency between 1 and 2 mHz.

**RESET:**

A low signal from this pin resets all internal registers. Without a reset, random numbers exist in all registers, the result being a rather unmusical "racket".

**A9:**

This pin acts as a chip select-signal. When it is low, the PSG registers are ready for communication.

**A8:**

Similar to A9, only it is active when high.

**TEST2:**

Test2 is used for testing in the factory, and is unused in normal operation.

**BDIR & BC1,2:**

The BDIR (Bus DIRection), BC1 and BC2 (Bus Control) pins control the PSG's register access.

<u>BDIR</u>	<u>BC2</u>	<u>BC1</u>	<u>PSG function</u>
0	0	0	Inactive
0	0	1	Latch address
0	1	0	Inactive
0	1	1	Read from PSG
1	0	0	Latch address
1	0	1	Inactive
1	1	0	Write to PSG
1	1	1	Latch address

Only four of these combinations are of any use to us; those with a 5+ voltage running over BC2. So, here's what we have left:

<u>BDIR</u>	<u>BC1</u>	<u>Function</u>
0	0	Inactive, PSG data bus high
0	1	Read PSG registers
1	0	Write PSG registers
1	1	Latch, write register number(s)

**DA0 - 7:**

These pins connect the sound chip to the processor, through the data bus. The identifier DA means that both data and (register) addresses can be sent over these lines.

**ANALOG C:**

Works with channel C (see ANALOG B, above).

**TEST1:**

See TEST2.

**Vcc:**

+5 volt pin.

## 1.6.2 The 2149 Registers and their Functions

Now let's look at the functions of the individual registers. One point of interest: the contents of the address register remain unaltered until reprogrammed. You can use the same data over and over, without having to send that data again.

### Reg 0,1:

These registers determine the period length, and the pitch of ANALOG A. Not all 16 bits are used here; the eight bits of register 0 (set frequency) and the four lowest bits of register 1 (control step size). The lower the 12-bit value in the register, the higher the tone.

### Reg 2,3:

Same as registers 0 and 1, only for channel B.

### Reg 4,5:

Same as registers 0 and 1, only for channel C.

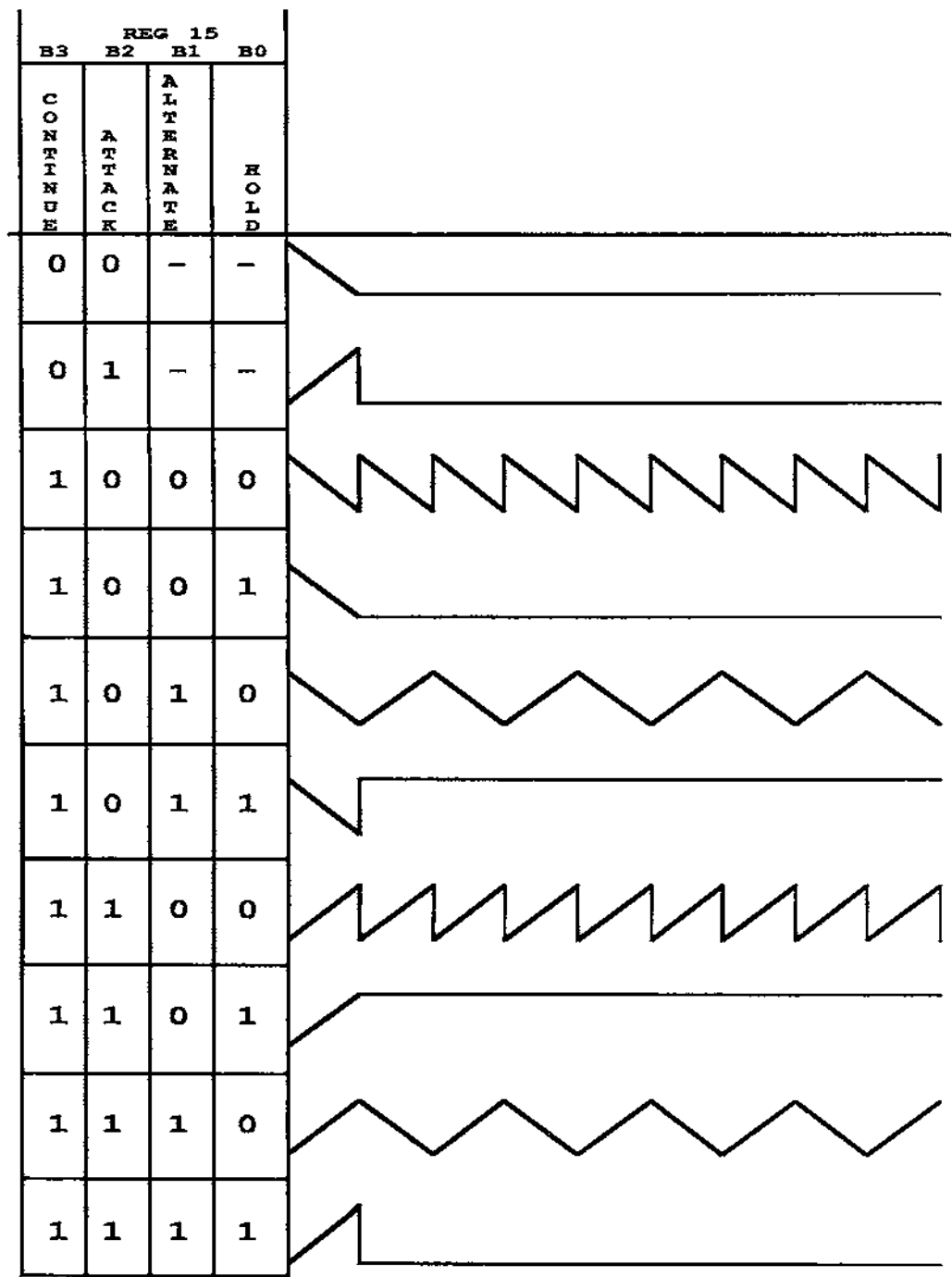
### Reg 6:

The five lowest bits of this register control the noise generator. Again, the smaller the value, the higher the noise "pitch".

### Reg 7:

Bit 0:	Channel A tone on/off	0=on /1=off
Bit 1:	Channel B tone on/off	0=on /1=off
Bit 2:	Channel C tone on/off	0=on /1=off
Bit 3:	Channel A noise on/off	0=on /1=off
Bit 4:	Channel B noise on/off	0=on /1=off
Bit 5:	Channel C noise on/off	0=on /1=off
Bit 6:	Port A in/output	0=in /1=out
Bit 7:	Port B in/output	0=in /1=out

Figure 1.6-2 Envelopes of the PSG



**Reg 8:**

Bits 0-3 of this register control the signal volume of channel A. When bit 4 is set, the envelope register is being used and the contents of bits 0-3 are ignored

**Reg 9:**

Same as register 8, but for channel B.

**Reg 10:**

Same as register 8, but for channel C.

**Reg 11,12:**

The contents of register 11 are the low-byte and the contents of register 12 are the high-byte of the sustain.

**Reg 13:**

Bits 0-3 determine the waveform of the envelope generator. The possible envelopes are pictured in Figure 1.6-2.

**Reg 14,15:**

These registers comprise the two 8-bit ports. Register 14 is connected to Port A and register 15 is connected to Port B. If these ports are programmed as output (bits 7 and 8 of register 7) then values may be sent through these registers.

## 1.7 I/O Register Layout in the ST

The entire I/O range (all peripheral ICs and other registers) is controlled by a 32K address register -- \$FF8000 - \$FFFFFF. Below is a complete table of the different registers. CAUTION: The I/O section can be accessed only in supervisor mode. Any access in user mode results in a bus-error.

\$FF8000	Memory configuration
\$FF8200	Video display register
\$FF8400	Reserved
\$FF8600	DMA/disk controller
\$FF8800	Sound chip
\$FFFA00	MFP 68901
\$FFFC00	ACIAs for MIDI and keyboard

The addresses given refer only to the start of each register, and supply no hint as to the size of each. More detailed information follows.

### \$FF8000 Memory Configuration

There is a single 8-bit register at \$FF8001 in which the memory configuration is set up (four lowest bits). The MMU-IC is designed for maximum versatility within the ST. It lets you use three different types of memory expansion chips: 64K, 256K, and the 1M chips. Since all of these ICs are bit-oriented instead of byte-oriented, 16 memory chips of each type are required for memory expansion. The identifier for 16 such chips (regardless of memory capacity) is BANK. So, expansion is possible to 128 Kbyte, 512 Kbyte or even 2 Megabytes.

MMU can control two banks at once, using the RAS- and CAS- signals. The table on the next page shows the possible combinations:

<u>\$FF8001</u>	<u>Bit</u>	<u>Memory configuration</u>	
	3-0	Bank 0	Bank 1
	0000	128K	128K
	0001	128K	512K
	0010	128K	2 M
	0011	reserved	
	0100	512K	128K
	0101	512K	512K
	0100	512K	2 M, normally reserved
	0100	reserved	
	1000	2M	128K
	1001	2M	512K
	1010	2M	2M
	1011	reserved	
	11XX	reserved	

The memory configuration can be read from or written to.

### \$FF8200 Video Display Register

This register is the storage area that determines the resolution and the color palette of the video display.

\$FF8201    8-bit    Screen memory position (high-byte)  
 \$FF8203    8-bit    Screen memory position (low-byte)

These two read/write registers are located at the beginning of the 32K video RAM.

In order to relocate video RAM, another register is used. This register is three bytes long and is located at \$FF8205. Video RAM can be relocated in 256-byte increments. Normally the starting address of video RAM is \$78000.

\$FF8205    8-bit    Video address pointer (high-byte)  
 \$FF8207    8-bit    Video address pointer (mid-byte)  
 \$FF8209    8-bit    Video address pointer (low-byte)

These three registers are read ONLY. Every three microseconds, the contents of these registers are incremented by 2.

---

```

$FF820A BIT      Synchronization mode
      1 0
      : :-- 0=internal,1=external synchronization
      :---- 0=60 Hz, 1=50Hz screen frequency

```

The bottom two bits of this register control synchronization mode; the remaining bits are unused. If bit 0 is set, the HSync and VSync impulses are shut off, which allows for screen synchronization from external sources (monitor jack). This offers new realm of possibilities in video, synchronization of your ST and a video camera, for example.

Bit 1 of the sync-mode register handles the screen frequency. This bit is useful only in the two "lowest" resolutions. High-res operation puts the ST at a 70 Hz screen frequency.

Sync mode can be read/written.

```

$FF8240    16-bit    Color palette register 0
$FF8242    16-bit    Color palette register 1
      :           :
      :           :    Color palette registers 2-13
      :           :
$FF825C    16-bit    Color palette register 14
$FF825E    16-bit    Color palette register 15

```

Although the ST has a total of 512 colors, only 16 different colors can be displayed on the screen at one time. The reason for this is that the user has 16 color pens on screen, and each can be one of 512 colors. The color palette registers represent these pens. All 16 registers contain 9 bits which affect the color:

```

FEDCBA9876543210
.....XXX.XXX.XXX

```

The bits marked X control the registers. Bits 0-2 adjust the shade of blue desired; 4-6, green hue; and 8-A, red. The higher the value in these three bits, the more intense the resulting color.

Middle resolution (640 X 200 points) offers four different colors; colors 4 through 15 are ignored by the palette registers.

When you want the maximum of 16 colors, it's best to zero-out the contents of the palette registers.



High-res (640 X 400 points) gives you a choice on only one "color"; bit 0 of palette register 0 is set to the background color. If the bit is cleared, then the text is black on a light background. A set bit reverses the screen (light characters, black background). The color register is a read/write register.

\$FF8260	Bit	Resolution
	1 0	
	0 0	320 X 200 points, four focal planes
	0 1	640 X 200 points, two focal planes
	1 0	640 X 400 points, one focal planes

This register sets up the appropriate hardware for the graphic resolution desired.

### \$FF8600 DMA/Disk Controller

\$FF8600	reserved
\$FF8602	reserved

\$FF8604	16-bit	FDC access/sector count
----------	--------	-------------------------

The lowest 8 bits access the FDC registers. The upper 8 bits contain no information, and consistently read 1. Which register of the FDC is used depends upon the information in the DMA mode control register at \$FF8606. The FDC can also be accessed indirectly.

The sector count-register under \$FF8604 can be accessed when the appropriate bit in the DMA control register is set. The contents of these addresses are both read/write.

\$FF8606	16-bit	DMA mode/status
----------	--------	-----------------

When this register is read, the DMA status is found in the lower three bits of the register.

Bit 0	0=no error, 1=DMA error
Bit 1	0=sector count = null, 1=sector count<>null
Bit 2	Condition of FDC DATA REQUEST signal

Write access to this address controls the DMA mode register.

---

Bit 0	unused
Bit 1	0=pin A0 is low 1=pin A0 is high
Bit 2	0=pin A1 is low 1=pin A1 is high
Bit 3	0=FDC access 1=HDC access
Bit 4	0=access to FDC register 1=access to sector count register
Bit 5	0, reserved
Bit 6	0=DMA on 1=no DMA
Bit 7	0=hard disk controller access (HDC) 1=FDC access
Bit 8	0=read FDC/HDC registers 1=write to FDC/HDC registers
\$FF8609	8-bit DMA basis and counter high-byte
\$FF860B	8-bit DMA basis and counter mid-byte
\$FF860D	8-bit DMA basis and counter low-byte

DMA transfer will tell the hardware at which address the data is to be moved. The initialization of the three registers must begin with the low-byte of the address, then mid-byte, then high-byte.

### \$FF8800 Sound Chip

The YM-2149 has 16 internal registers which can't be directly addressed. Instead, the number for the desired register is loaded into the select register. The chosen registers can be read/write, until a new register number is written to the PSG.

\$FF8800	8-bit	Read data/Register select
----------	-------	---------------------------

Reading this address gives you the last register used (normally port A), by which disk drive is selected. This can be accomplished with write-protect signals, although these protected contents can be accessed by another register. Port A is used for multiple control functions, while port B is the printer data port.

## PORT A

Bit 0	Page-choice signal for double-sided floppy drive
Bit 1	Drive select signal -- floppy drive 0
Bit 2	Drive select signal -- floppy drive 1
Bit 3	RS-232 RTS-output
Bit 4	RS-232 DTR output
Bit 5	Centronics strobe
Bit 6	Freely usable output (monitor jack)
Bit 7	reserved

When \$FF8800 is written to, the select register of the PSG is alerted. The information in the bottom four bits are then considered as register numbers. The necessary four-bit number serves for writing to the PSG.

\$FF8802    8-bit    Write data

Attempting to read this address after writing to it will give you \$FF only, while BDIR and BC1 are nulls.

Writing register numbers and data can be performed with a single MOVEP instruction.

**\$FFFA00    MFP 68901**

The MFP's 24 registers are found at uneven addresses from \$FFFA01-\$FFFA2F:

\$FFFA01	8-bit	Parallel port
\$FFFA03	8-bit	Active Edge register
\$FFFA05	8-bit	Data direction
\$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	Vector register
\$FFFA19	8-bit	Timer A control
\$FFFA1B	8-bit	Timer B control

---

\$FFFA1D	8-bit	Timer C & 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

See the chapter on the MFP for details on the individual registers.

#### I/O Port

Bit 0	Centronics busy
Bit 1	RS-232 data carrier detect - input
Bit 2	RS-232 clear to send - input
Bit 3	reserved
Bit 4	keyboard and MIDI interrupt
Bit 5	FDC and HDC interrupt
Bit 6	RS-232 ring indicator
Bit 7	Monochrome monitor detect

Timers A and B each have an input which can be used by external timer control, or send a time impulse from an external source. Timer A is unused in the ST, which means that the input is always available, but it isn't connected to the user port, so the Centronics busy pin is connected instead. You can use it for your own purposes.

Timer B is used for counting screen lines in conjunction with DE (Display Enable).

The timer outputs in A-C are unused. Timer D, on the other hand, sends the timing signal for the MFP's built-in serial interface.

**\$FFFC00 Keyboard and MIDI ACIAs**

The communications between the ST, the keyboard, and musical instruments are handled by two registers in the ACIAs.

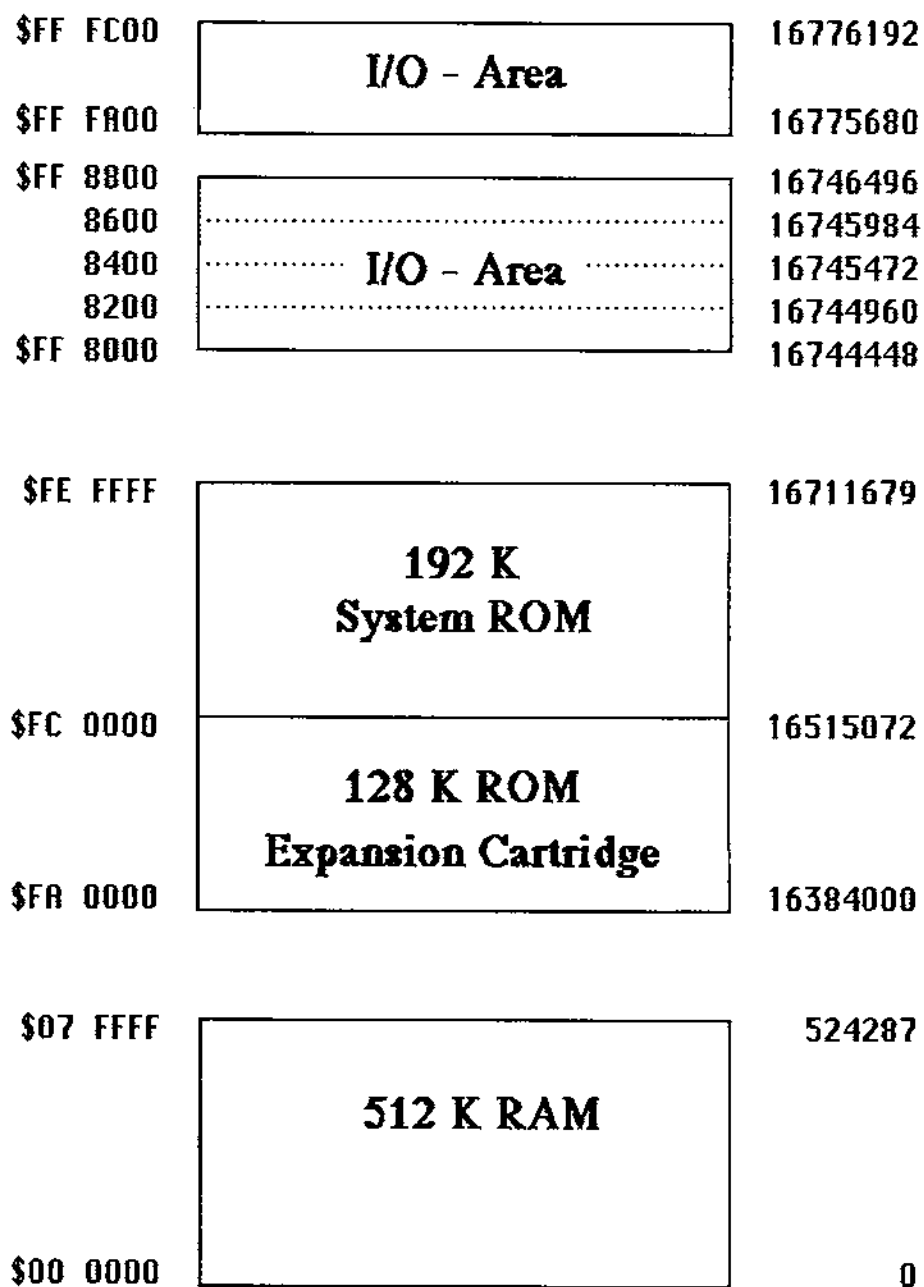
\$FFFC00	8-bit	Keyboard ACIA control
\$FFFC02	8-bit	Keyboard ACIA data
\$FFFC04	8-bit	MIDI ACIA control
\$FFFC06	8-bit	MIDI ACIA data

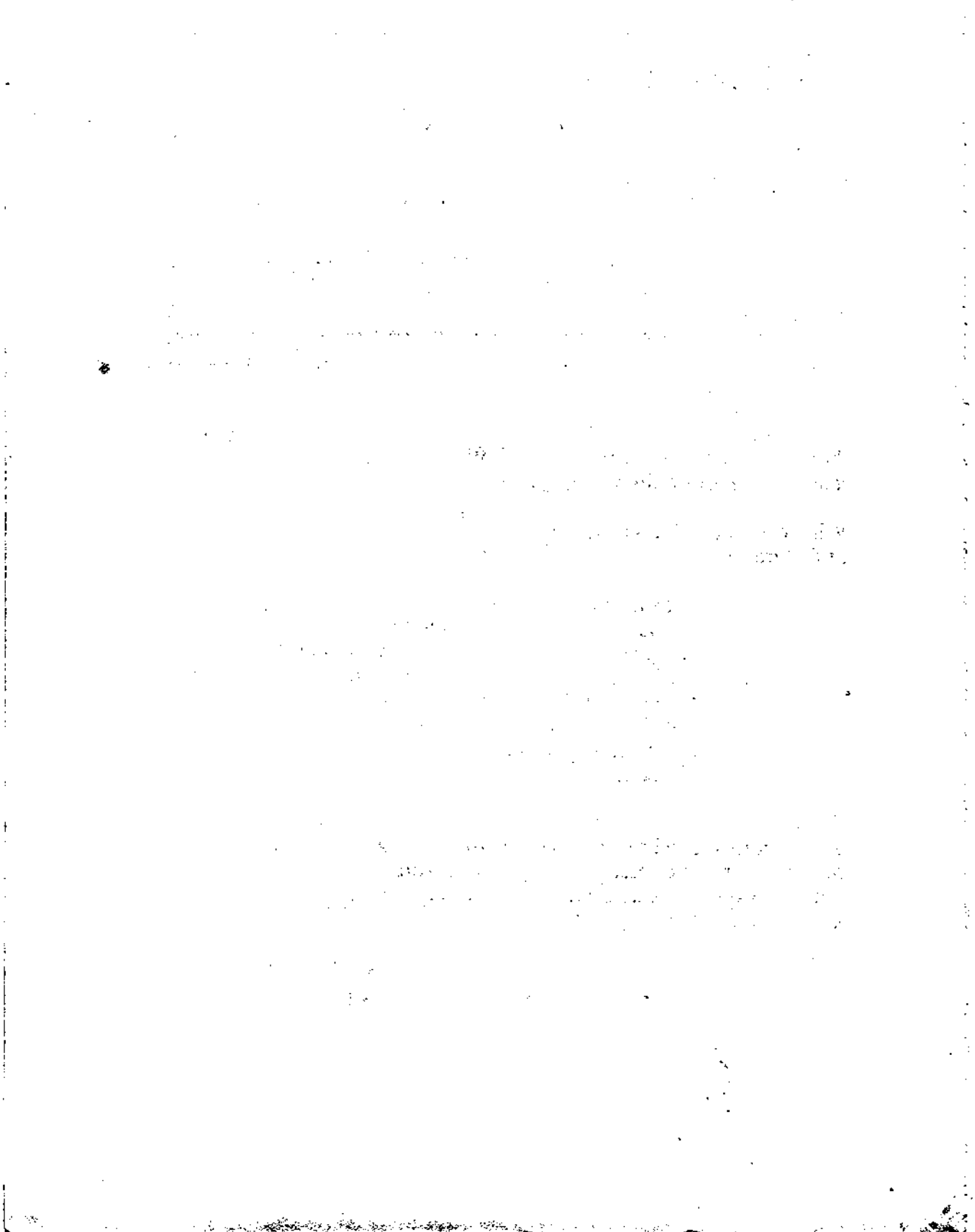
**Figure 1.7-1 I/O Assignments**

<b>\$FFFC00</b>	<b>2 ACIA's 6580</b>
<b>\$FFFA00</b>	<b>MFP 68901</b>

<b>\$FF8800</b>	<b>SOUND AY-3-8910</b>
<b>\$FF8600</b>	<b>DMA / WD 1770</b>
<b>\$FF8400</b>	<b>RESERVED</b>
<b>\$FF8200</b>	<b>VIDEO CONTROLLER</b>
<b>\$FF8000</b>	<b>DATA CONFIGURATION</b>

Figure 1.7-2 Memory Map of the ATARI ST





# Chapter Two

## The Interfaces

- 2.1 The Keyboard**
  - 2.1.1 The Mouse**
  - 2.1.2 Keyboard commands**
- 2.2 The Video Connection**
- 2.3 The Centronics Interface**
- 2.4 The RS-232 Interface**
- 2.5 The MIDI Connections**
- 2.6 The Cartridge Slot**
- 2.7 The Floppy Disk Interface**
- 2.8 The DMA Interface**



## The Interfaces

### 2.1 The Keyboard

Do you think it's really necessary to give a detailed report on something as trivial as the keyboard, since keyboards all function the same way? Actually the title should read "Keyboard Systems" or something similar. The keyboard is controlled by its own processor. You will soon see how this affects the assembly language programmer.

The keyboard processor is single-chip computer (controller) from the 6800 family, the 6301. Single chip means that everything needed for operation is found on a single IC. In actuality, there are some passive components in the keyboard circuit along with the 6301.

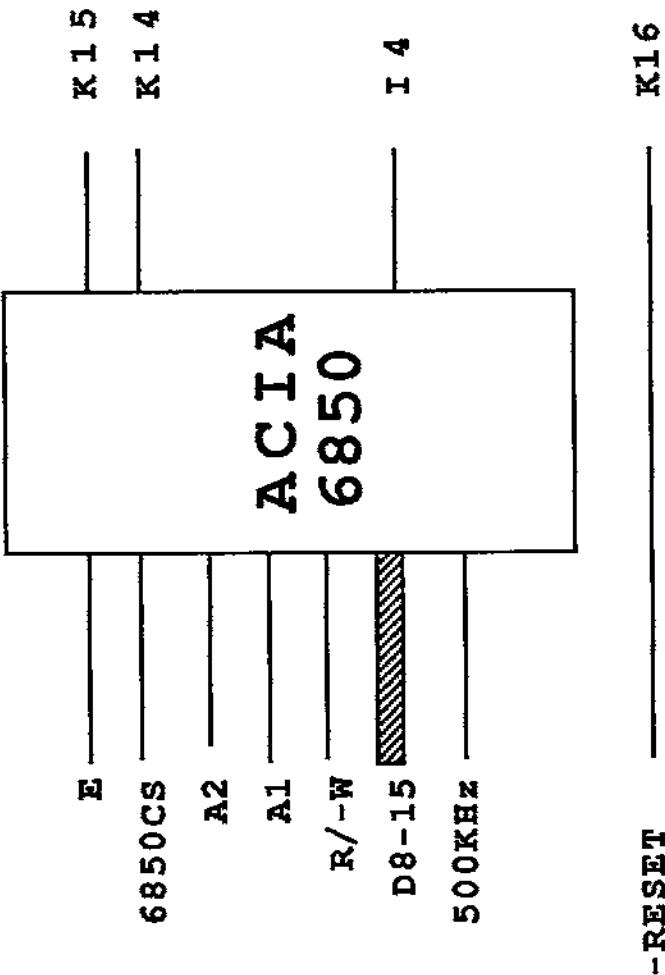
The 6301 has ROM, RAM, some I/O lines, and even a serial interface on the chip. The serial interface handles the traffic to and from the main board.

The advantage of this design is easy to see. The main computer is not burdened by having to continually poll the keyboard. Instead it can dedicate itself completely to processing your programs. The keyboard processor notifies the system if an event occurs that the operating system should be aware of.

The 6301 is not only responsible for the relatively boring task of reading the keyboard, however. It also takes care of the rather complicated tasks required in connection with the mouse. The main processor is then fed simply the new X and Y coordinates when the mouse is moved. Naturally, anything to do with the joysticks is also taken care of by the keyboard controller.

In addition, this controller contains a real-time clock which counts in one-second increments.

Figure 2.1-1 6850 Interface to 68000



In Figure 2.1-1 is an overview of the interface to the 68000. As you see, the main processors is burdened as little as possible. The ACIA 6850 ensures that it is disturbed only when a byte has actually been completely received from the keyboard. The ACIA, by the way, can be accessed at addresses \$FFFC00 (control register) and \$FFFC02 (data register). The individual connection to the keyboard takes place over lines K14 and K15. K indicates the plug connection by which the keyboard is connected to the main board.

The signal that the ACIA has received a byte is first sent over line 14 to the MFP 68901 which then generates an interrupt to the 68000. The clock frequency of 500KHz comes from GLUE. From this results the "odd" transfer rate of 7812.5 baud.

In case you were surprised that data can also be sent to the keyboard processor, you will find the solution to the puzzle in Chapter 2.1.2.

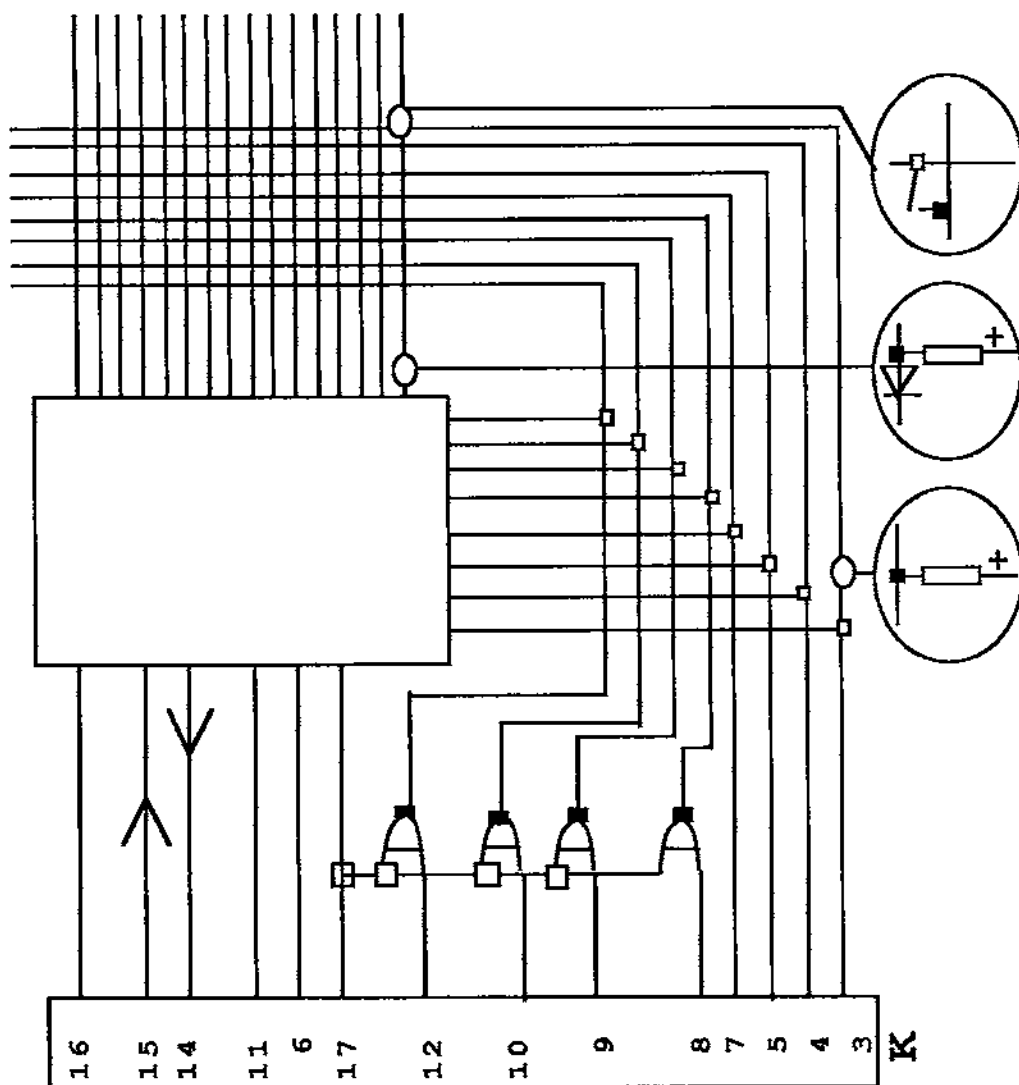
The block diagram of the keyboard circuit is found in Figure 2.1-2. The function is as simple as the figure is easy to read. The processor has 4K of ROM available. The 128 bytes of RAM is comparatively small, but it is used only as a buffer and for storing pointers and counters.

The lines designated with K are again the plug connections assigned to the main board. With few exceptions, the connections for the joystick and mouse are also put through. K16 is the reset line from the 68000. K15 carries the send data from the 6850, K14 the send data from the 6301.

The I/O ports 1(0-7), 3(1-7), and 4(0-7) are responsible for reading the keyboard matrix. One line from ports 3 and 4 is pulled low in a cycle. The state of port 1 is the checked. If a key is pressed, the low signal comes through on port 1.

Each key can be identified from the combination of value placed on ports 3 and 4 and the value read from port 1.

If none of the lines of Port 3 and 4 are placed low and a bit of port 1 still equals zero, a joystick is active on the outer connector 1. The data from outer connector 0, to which a mouse or a joystick can be connected, does not come through by chance since it must first be switched through the NAND gate with port 2 (bit 0). The buttons on the mouse or the joystick then arrive at port 2 (1 and 2).

**Figure 2.1-2 Block Diagram of Keyboard Circuit**

The assignments of the K lines to the signal names on the outer connector are found in the next section.

The processor 6301 is completely independent, but it can also be configured so that it works with an external ROM. Some of the port lines are then reconfigured to act as address lines. The configuration the processor assumes (one of eight possibilities) depends on the logical signal placed on port 2 (bits 0-2) during the reset cycle. All three lines high puts the processor in mode 7, the right one for the task intended here. But bits 1 and 2 depend on the buttons on the mouse. If you leave the mouse alone while powering-up, everything will be in order. If you hold the two buttons down, however, the processor enters mode 1 and makes a magnificent belly-flop, since the hardware for this operating mode is not provided. You notice this by the fact that the mouse cursor does not move on the screen if you move the mouse. Only the reset button will restore the processor.

### 2.1.1 The Mouse

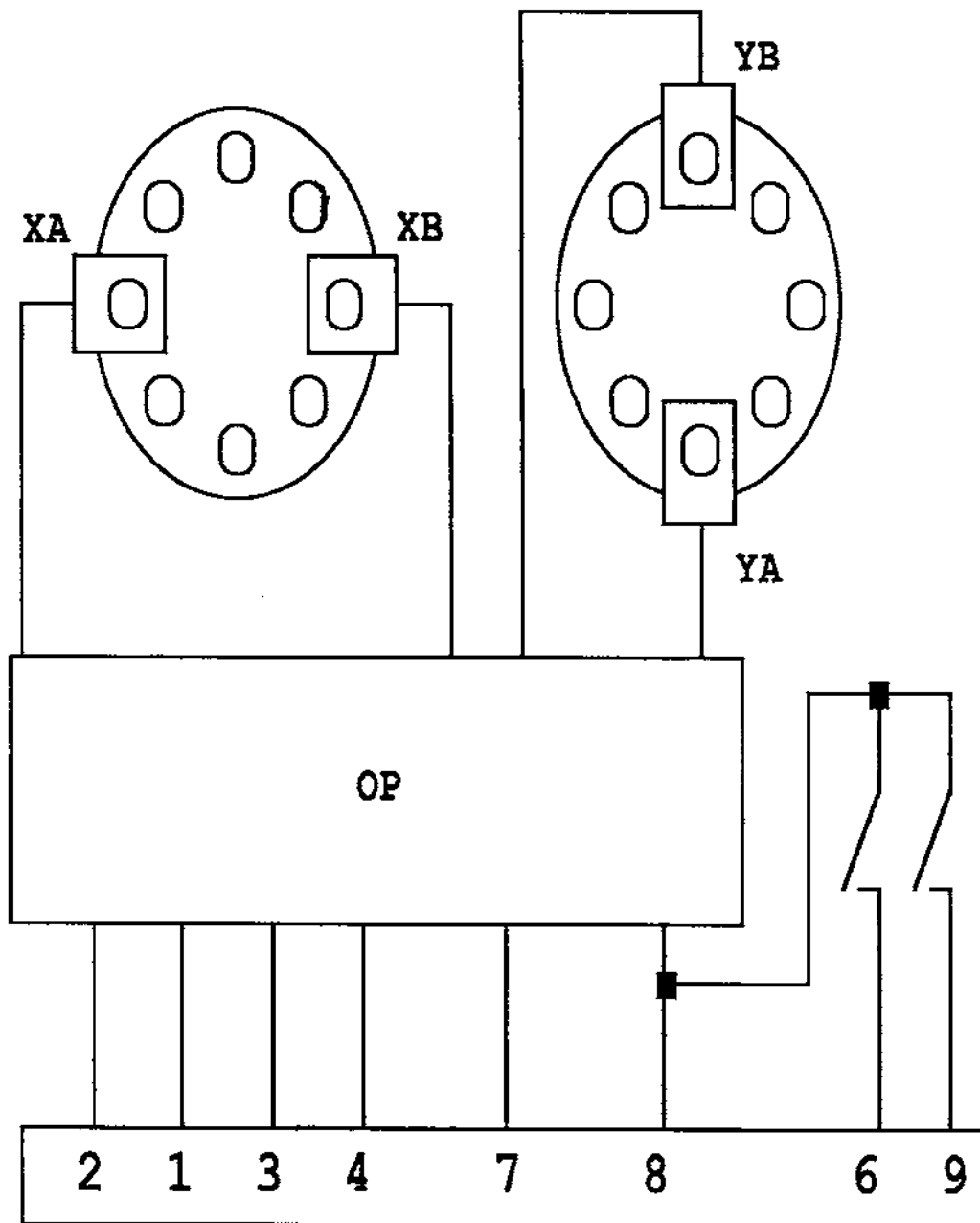
The construction of this little device is quite simple, but effective. Essentially, it consists of four light barriers, two encoder wheels, and a drive mechanism.

The task of the mouse is to give the computer information about its movements. This information consists of the components: direction on the X-axis, direction on the Y-axis, and the path traveled on each axis.

In order to do this, the rubber-covered ball visible from the outside drives two encoder wheels whose drive axes are at angle of 90 degrees to each other. The one or the other axis rotates more or less, forwards or backwards, depending on the direction the mouse is moved.

It is no problem to determine the absolute movement on each axis. The encoder wheels alternately interrupt the light barriers. One need only count the pulses from each wheel to be informed about the path traveled on each axis.

Figure 2.1.1-1 The Mouse



It is more difficult when the direction of movement is also required. The designers of the mouse used a convenient trick for this. There are not one, but two light barriers on each encoder wheel. They are arranged such that they are not shielded by the wheel at precisely the same time, but one shortly after the other. This arrangement may not be so clear in Figure 2.1.1-1, so we'll explain it in more detail. The direction can be determined by noticing which of the two light barriers is interrupted first. This is why the pulses from both light barriers are sent out, making a total of four. Corresponding to their significance they carry the names XA, XB, YA, YB.

The two contacts which you see on the picture represent the two buttons.

The large box on the picture is a quad operational amplifier which converts the rather rough light-barrier pulses into square wave signals.

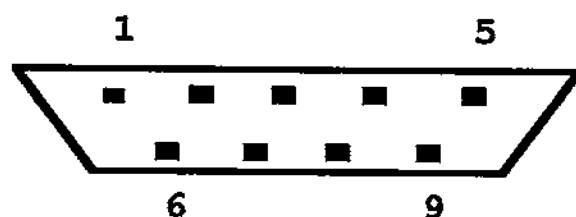
In Figure 2.1.1-2 is the layout of the control port on the computer, as you see it when you look at it from the outside. The designation behind the slash applies when a joystick is connected and the number in parentheses is the pin number of the keyboard connector.

#### 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)

**Figure 2.1.1-2 Mouse control port**

## 2.1.2 Keyboard commands

The keyboard processor "understands" some commands pertaining to such things as how the mouse is to be handled, etc. You can set the clock time, read the internal memory, and so on. You can find an application example in the assembly language listing on page 80 (after command \$21).

The "normal" action of the processor consists of keeping an eye on the keyboard and announcing each keypress. This is done by outputting the number of the key when the key is pressed. When the key is released the number is set again, but with bit 7 set. The result of this is that no key numbers greater than 127 are possible. You can find the assignment of the key numbers to the keys at the end of this section in figure 2.1.2-1. In reality these numbers only go up to 117 because values from \$F6 up are reserved for other purposes. There must be a way to pass more information than just key numbers to the main processor, information such as the clock time or the current position of the mouse. This cannot be handled in a single byte but only in something called a package, so the bytes at \$F6 signal the start of a package. Which header comes before which package is explained along with the individual commands.

A command to the keyboard processor consists of the command code (a byte) and any parameters required. The following description is sorted according to command bytes.

### **\$07**

Returns the result of pressing one of the two mouse buttons. A parameter byte with the following format is required:



Bit 0 =1: The absolute position is returned when a mouse button is pressed. Bit 2 must =0.  
Bit 1 =1: The absolute position is returned when a mouse button is released. Bit 2 must =0.  
Bit 2 =1: The mouse buttons are treated like normal keys. The left button is key number \$74, the right is \$75.  
Bits 3-7 must always be zero.

### \$08

Returns the relative mouse position from now on. This command tells the keyboard processor to automatically return the relative position (the distance from the previous position) whenever the mouse is moved. A movement is given when the number of encoder wheel pulses has reached a given threshold. See also \$0B. A relative mouse package looks like this:

1 byte Header in range \$F8-\$FB. The two lowest bits of the header indicate the condition of the two mouse buttons.  
1 byte Relative X-position (signed!)  
1 byte Relative Y-position (signed!)

If the relative position changes substantially between two packages so that the distance can no longer be expressed in one byte, another package is automatically created which makes up for the remainder.

### \$09

Returns the absolute mouse position from now on. This command also sets the coordinate maximums. The internal coordinate pointers are at the same time set to zero. The following parameters are required:

1 word Maximum X-coordinate  
1 word Maximum Y-coordinate

Mouse movements under the zero point or over the maximums are not returned.

### \$0A

With this command it is possible to get the key numbers of the cursor keys instead of the coordinates. A mouse movement then appears to the operating system as if the corresponding cursor keys had been pressed. These parameters are necessary:

- 1 byte Number of pulses (X) after which the key number for cursor left (or right) will be sent.
- 1 byte Number of pulses (Y) after which the key number for cursor up (or down) will be sent.

**\$0B**

This command sets the trigger threshold, above which movements will be announced. A certain number of encoder pulses elapse before a package is sent. This functions only in the relative operating mode. The following are the parameters:

- 1 byte Threshold in X-direction
- 1 byte Threshold in Y-direction

**\$0C**

Scale mouse. Here is determined how many encoder pulses will go by before the coordinate counter is changed by 1. This command is valid only in the absolute. The following parameters are required:

- 1 byte X scaling
- 1 byte Y scaling

**\$0D**

Read absolute mouse position. No parameters are required, but a package of the following form is sent:

- 1 byte Header = \$F7
- 1 byte Button status
  - Bit 0 = 1: Right button was pressed since the last read
  - Bit 1 = 1: Right button was not pressed
  - Bit 2 = 1: Left button was pressed since the last read
  - Bit 3 = 1: Left button was not pressed

From this strange arrangement you can determine that the state of a button has changed since the last read if the two bits pertaining to it are zero.

- 1 word Absolute X-coordinate
- 1 word Absolute Y-coordinate

**\$0E**

Set the internal coordinate counter. The following parameters are required:

1 byte      =0 as fill byte  
1 word      X-coordinate  
1 word      Y-coordinate

**\$0F**

Set the origin for the Y-axis is down (next to the user).

**\$10**

Set the origin for the Y-axis is up.

**\$11**

The data transfer to the main processor is permitted again (see \$13).  
Any command other than \$13 will also restart the transfer.

**\$12**

Turn mouse off. Any mouse-mode command (\$08, \$09, \$0A) turns the mouse back on. If the mouse is in mode \$0A, this command has no effect.

**\$13**

Stop data transfer to main processor.

NOTE: Mouse movements and key presses will be stored as long as the small buffer of the 6301 allows. Actions beyond the capacity of the buffer will be lost.

**\$14**

Every joystick movement is automatically returned. The packages sent have the following format:

1 byte      Header = \$FE or \$FF for joystick 0/1  
1 byte      Bits 0-3 for the position (a bit for each direction), bit 7 for the button

**\$15**

End the automatic-return mode for the joystick. When needed, a package must be requested with \$16.

**\$16**

Read joystick. After this command the keyboard sends a package as described above.



**\$1A**

Turn off joysticks. Any other joystick command turns them on again.

**\$1B**

Set clock time. This command sets the internal real-time clock in the keyboard processor. The values are passed in packed BCD, meaning a digit 0-9 for each half byte, yielding a two-digit decimal number per byte. The following parameters are necessary:

1 byte	Year, two digit (85, 86, etc.)
1 byte	Month, two digit (12, 01, etc.)
1 byte	Day, two digit (31, 01, 02, etc.)
1 byte	Hours, two digit
1 byte	Minutes, two digit
1 byte	Seconds, two digit

Any half byte which does not contain a valid BCD digit (such as F) is ignored. This makes it possible to change just part of the date or clock time.

**\$1C**

Read clock time. After receiving this command the keyboard processor returns a package having the same format as the one described above. A header is added to the package, however, having the value \$FC.

**\$20**

Load memory. The internal memory of the keyboard processor (naturally only the RAM in the range \$80 to \$FF makes sense) can be written with this command. It is not clear to us of what use this is since according to our investigations (we have disassembled the operating system of the 6301), no RAM is available to be used as desired. Perhaps certain parameters can be changed in this manner which are not accessible through "legal" means. Here are the parameters:

1 word	Start address
1 byte	Number of bytes (max. 128)
Data bytes (corresponding to the number)	

The interval at which the data bytes will be sent must be less than 20 msec.

**\$21**

Read memory. This command is the opposite of \$20. These parameters are required:

1 word      Address at which to read

A package having the following format is returned:

1 byte      Header 1 = \$F6. This is the status header which precedes all packages containing any operating conditions of the keyboard processor. We will come to the general status messages shortly.

1 byte      Header 2 = \$20 as indicator that this package carries the memory contents.

6 bytes     Memory contents starting with the address given in the command.

Here is a small program which we used to read the ROM in the 6301 and output it to a printer. Here you also see how the status packages arrive from the keyboard. These are normally thrown away by the 68000 operating system. Section 3.1 contains information about the GEMDOS and XBIOS calls used.

```

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 49F900000000F9      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 028000000000F      andi.l  #15,d0
45 00000068 47F900000000E8      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 028000000000F      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**

Execute routine. With this command you can execute a subroutine in the 6301. Naturally, you must know exactly what it does and where it is located, so long as you have not transferred it yourself to RAM with \$20 (assuming you found some free space). The only required parameters are:

1 word    Start address

**Status messages**

You can at any time read the operating parameters of the keyboard by simply adding \$80 to the command byte with which you would to set the operating mode (whose parameters you want to know). You then get a status package back (header=\$F6), whose format corresponds exactly to those which would be necessary for setting the operating mode.

An example makes it clearer: you want to know how the mouse is scaled. So you send as the command the value \$8C (since \$0C sets the scaling). You get the following back:

```
1 byte Status header = $F6
1 byte X-scaling
1 byte Y-scaling
```

This is the same format which would be necessary for the command \$0C. For commands which do not require parameters, you get the evoked command back as such. For example, say you want to know what operating mode the joystick is in (\$14 or \$15). You send the value \$94 (or \$95, it makes no difference). As status package you receive, in addition to the header, either \$14 or \$15 depending on the operating mode of the joystick handler.

Allowed status checks are: \$87, \$88, \$89, \$8A, \$8B, \$8C, \$8F, \$90, \$92, \$94, \$99, and \$9A.

In conclusion we have a tip for those for whom the functions of the keyboard are too meager and who want to give it more "intelligence". The processor 6301 is also available in "piggy-back" version, the 63P01 (Hitachi). This model does not have ROM built in, but has a socket on the top for an EPROM of type 2732 or 2764 (8K!). You can then realize your own ideas and, for example, use the two joystick connections as universal 4-bit I/O ports, for which you can also extend the command set in order to access the new functions from the XBIOS as well.

Figure 2.1.2-1 ATARI ST Key Assignments

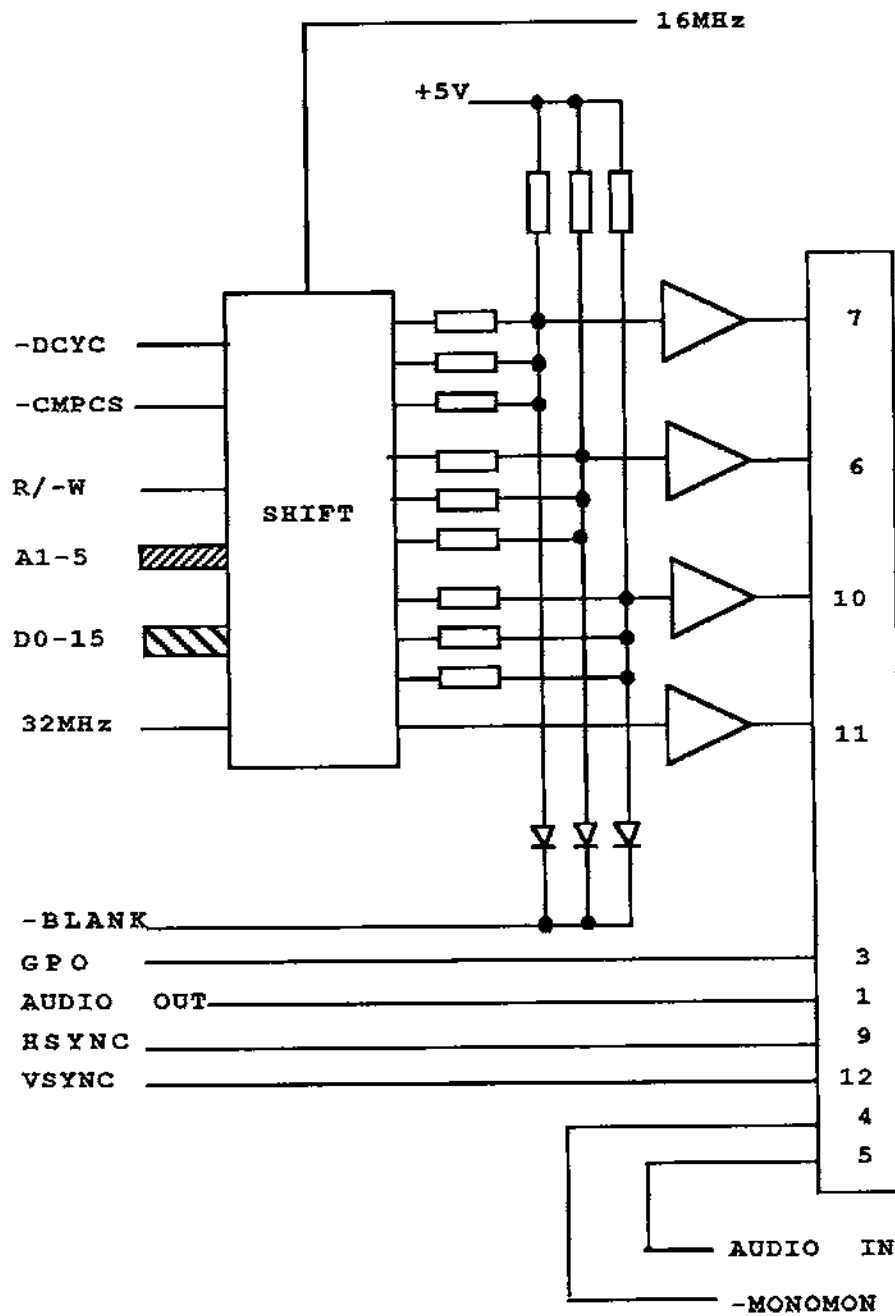
																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

## 2.2 The Video Connection

Without this, nothing would be displayed. You would be tapping in the dark in the truest sense of the word. Conspicuous are the many pins on the connection. Naturally more lines are required for hooking up an RGB monitor than for a monochrome screen, but seven would be enough. There is also something special about the remaining lines. In Figure 2.2-1 you find a block diagram in which you can see how the video connection is tied to the system. The numbering of the pins is given on the figure on the next page, as you can see, when you look at the connector from the outside. Here is the pin layout:

- 1 AUDIO OUT. This connection comes from the amplifier connected to the output of the sound chip. A high-impedance earphone can be attached here if you do not use the original monitor.
- 2 Not used
- 3 GPO, General Purpose Output. This connection is available for your use. The line has TTL levels and comes from I/O port A bit 6 of the sound chip.
- 4 MONOCHROME DETECT. If this line, which leads to the I7 input of the MFP 68901, is low, the computer enters the high-resolution monochrome mode. If the state of the line changes during operation, a cold start is generated.
- 5 AUDIO IN leads to the input of the amplifier described in 1 and is there mixed with the output of the sound chip.
- 6 GREEN is the analog green output of the video shifter.
- 7 RED. Red output.
- 8 GROUND.
- 9 HORIZONTAL SYNC is responsible for the horizontal beam return of the monitor.

Figure 2.2-1 Diagram of Video Interface



10 BLUE is the analog blue output of the video shifter.

11 MONOCHROME provides a monochrome monitor with the intensity signal.

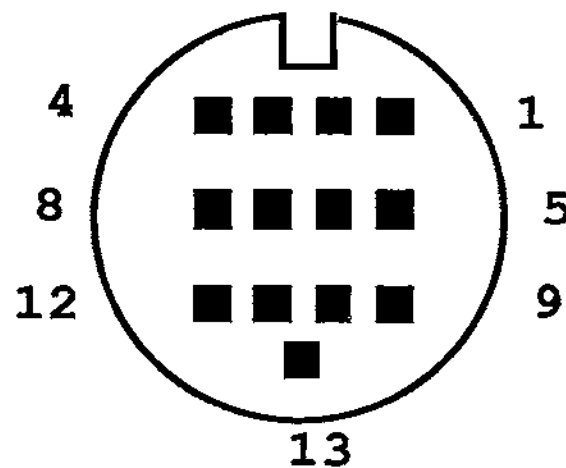
12 VERTICAL SYNC takes care of the beam return at the end of the screen.

13 GROUND.

A tip for the hardware hobbyist:

A plug to fit this connector is not available. If you want to make a plug for connecting other monitors, simply use a piece of perf board in which you have soldered pins, since the pins are fortunately organized in a 1/10" array. Pin 13 is out of order, but it is not needed since pin 8 is also available for ground.

**Figure 2.2-2 Monitor Connector**



## 2.3 The Centronics Interface

A standard Centronics parallel printer can be connected to this interface, provided that you have the proper cable. As you can see in Figure 2.3-2, the connection to the system is somewhat unusual. The data lines and the strobe of the universal port of the sound chip are used. So you find these too on the picture, in which the other lines, which will not be described in the section, will not disturb you. They belong to the disk drive and RS-232 interface and are handled there.

Here is the pin description:

- 1        -STROBE indicates the validity of the byte on the data lines to the connected device by a low pulse.
- 2-9     DATA
- 11      BUSY is always placed high by the printer when it is not able to receive additional data. This can have various causes. Usually the buffer is full or the device is off line.
- 18-15   GROUND.

All other pins are unused.

A tip for making a cable. Get flat-cable solderless connectors. You need a type D25-subminiature, a Cinch 36-pin (3M,AMP) and the appropriate length of 25-conductor flat ribbon cable. You squeeze the connectors on the cable so that pins 1 match up on both sides (they are connected together). The other connections then match automatically. Note that there will naturally be some pins free on the printer side.

**Figure 2.3-1 Printer Port Pins**

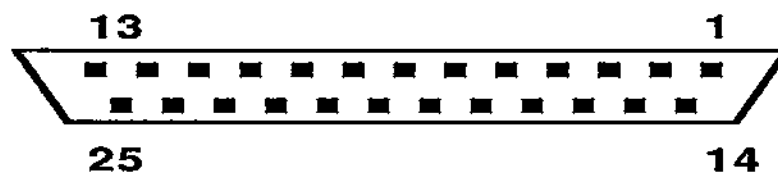
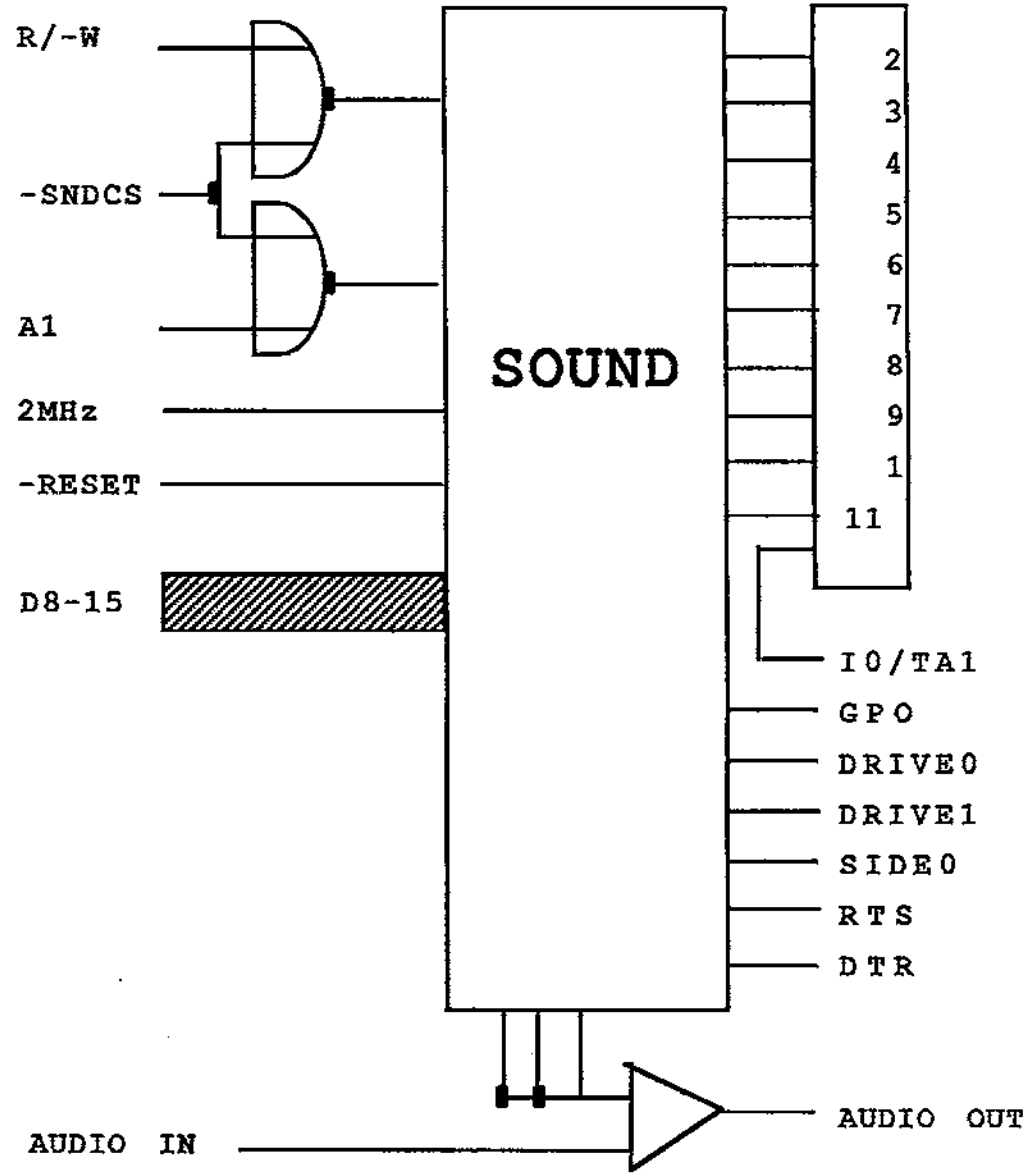


Figure 2.3-2 Centronics Connection



## 2.4 The RS-232 Interface

This interface usually serves for communication with other computers and modems. You can also connect a printer here. Note the description of pin 5!

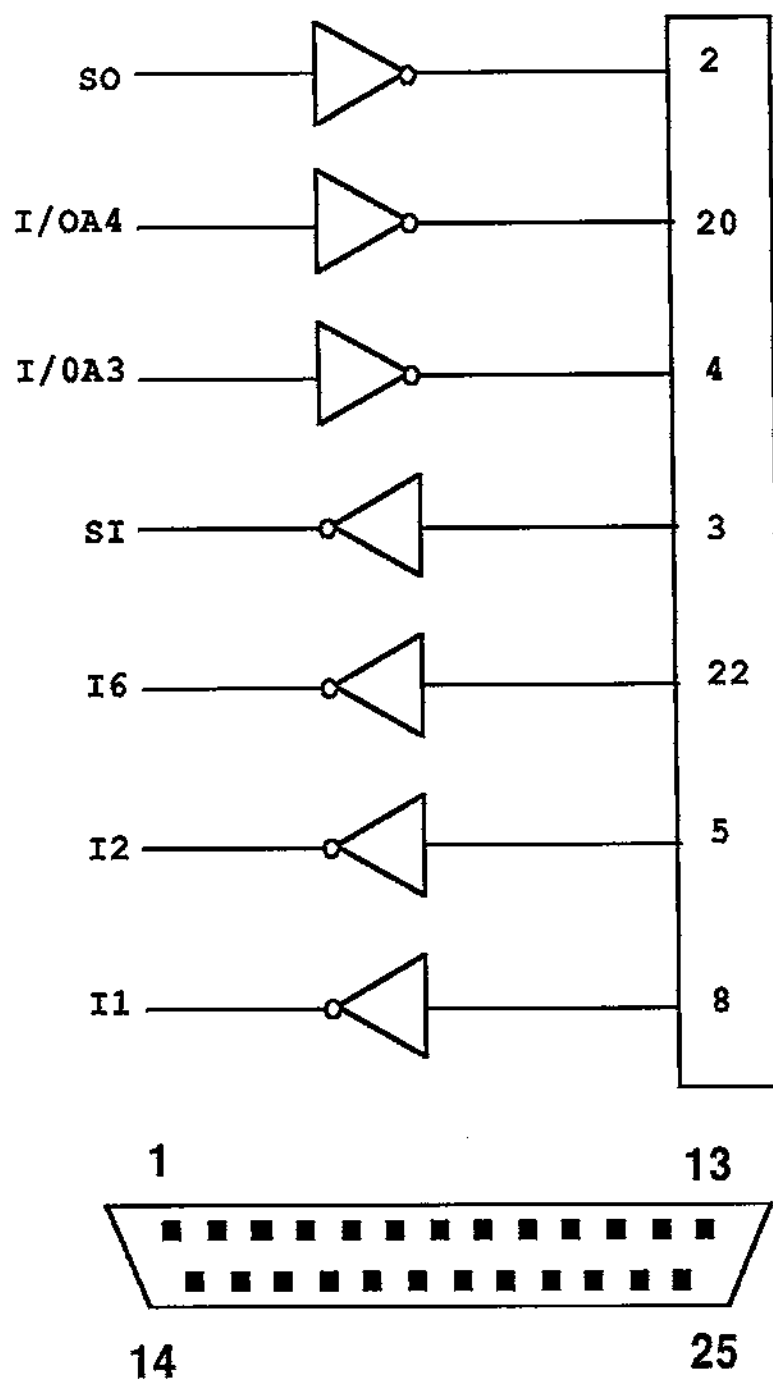
Figure 2.4-1 shows the connection to the system. Normally you don't have to do any special programming to use this interface. It is taken care of by the operating system. Here the control of the interface is not controlled by a special IC (UART) as is usually the case, but the lines are serviced more or less "by hand." The shift register in the MFP is used for this purpose. The handshake lines however come from a wide variety of sources. Note this in the following pin description:

- 1 CHASSIS GROUND (shield)  
This is seldom used.
- 2 TxD  
Send data
- 3 RxD  
Receive data
- 4 RTS  
Ready to send comes from I/O port A bit 3 of the sound chip and is always high when the computer is ready to receive a byte. On the Atari, this signal is first placed low after receiving a byte and is kept low until the byte has been processed.
- 5 CTS  
Clear to send of a connected device is read at interrupt input I2 of the MFP. At the present time this signal is handled improperly by the operating system. Therefore it is possible to connect only devices which "rattle" the line after every received byte (like the 520ST with RTS). The signal goes to input I2 of the MFP, but unfortunately is tested only for the signal edge. You will not have any luck connecting a printer because they usually hold the CTS signal high as long as the buffer is not full. There is no signal edge after each byte, which means that only the first byte of a text is transmitted, and then nothing.



- 7    **GND**  
Signal ground.
  
- 8    **DCD**  
Carrier signal detected. This line, which goes to interrupt input I1 of the MFP, is normally serviced by a modem, which tells the computer that connection has been made with the other party.
  
- 20   **DTR**  
Device ready. This line signals to a device that the computer is turned on and the interface will be serviced as required. It comes from I/O port A bit 4 of the sound chip.
  
- 22   **RI**  
Ring indicator is a rather important interrupt on I6 of the MFP and is used by a modem to tell the computer that another party wishes connection, that is, someone called.

Figure 2.4-1 RS-232 Connection



## 2.5 The MIDI Connections

The term MIDI is probably unknown to many of you. It is an abbreviation and stands for Musical Instrument Digital Interface, an interface for musical instruments.

It is certainly clear that we can't simply hook up a flute to this port. So first a little history. Music professionals (more precisely: keyboardists, musicians who play the synthesizer) demanded agreement between the various manufacturers to interface computers to musical instruments. They found it absurd to connect complicated set-ups with masses of wire. The idea was to service several synthesizers from one keyboard.

The tone created was basically analog (and still is, to a degree), so that the manufacturers agreed that a control voltage difference of 1V corresponded to a difference in tone of 1 octave. This way one could play several devices under "remote control," but not service them.

This changed substantially when the change was made to digital tone creation. Here one didn't have to turn a bunch of knobs, there were buttons to press, whereby the basis for digital control was created.

Some manufacturers got together and designed a digital interface, the basic commands of which would be the same throughout, but which would still support the additional features of a given device.

The device is based on the teletype, the current-loop principle, which is not very susceptible to noise, but significantly faster. The transfer rate is 31250 baud (bits per second). The data format is set at one start bit, eight data bits, and one stop bit.

An IC can therefore be used for control which would otherwise be used for RS-232 purposes. You see the connection to the system in figure 2.5-1.

Logically, MIDI is multi-channel system, meaning that 16 devices can be serviced by one master, or a device with 16 voices. These devices are all connected to the same line (bus principle). To identify which device or which voice is intended, each data packet is preceded by the channel number. The device which recognizes this number as its own then executes the desired action.

You may wonder what such an interface is doing in a computer. A computer can provide an entire arsenal of synthesizers with settings or complete melodies (sequencer) because of its high speed and memory capacity. It can also be used to record and store input from a synthesizer keyboard.

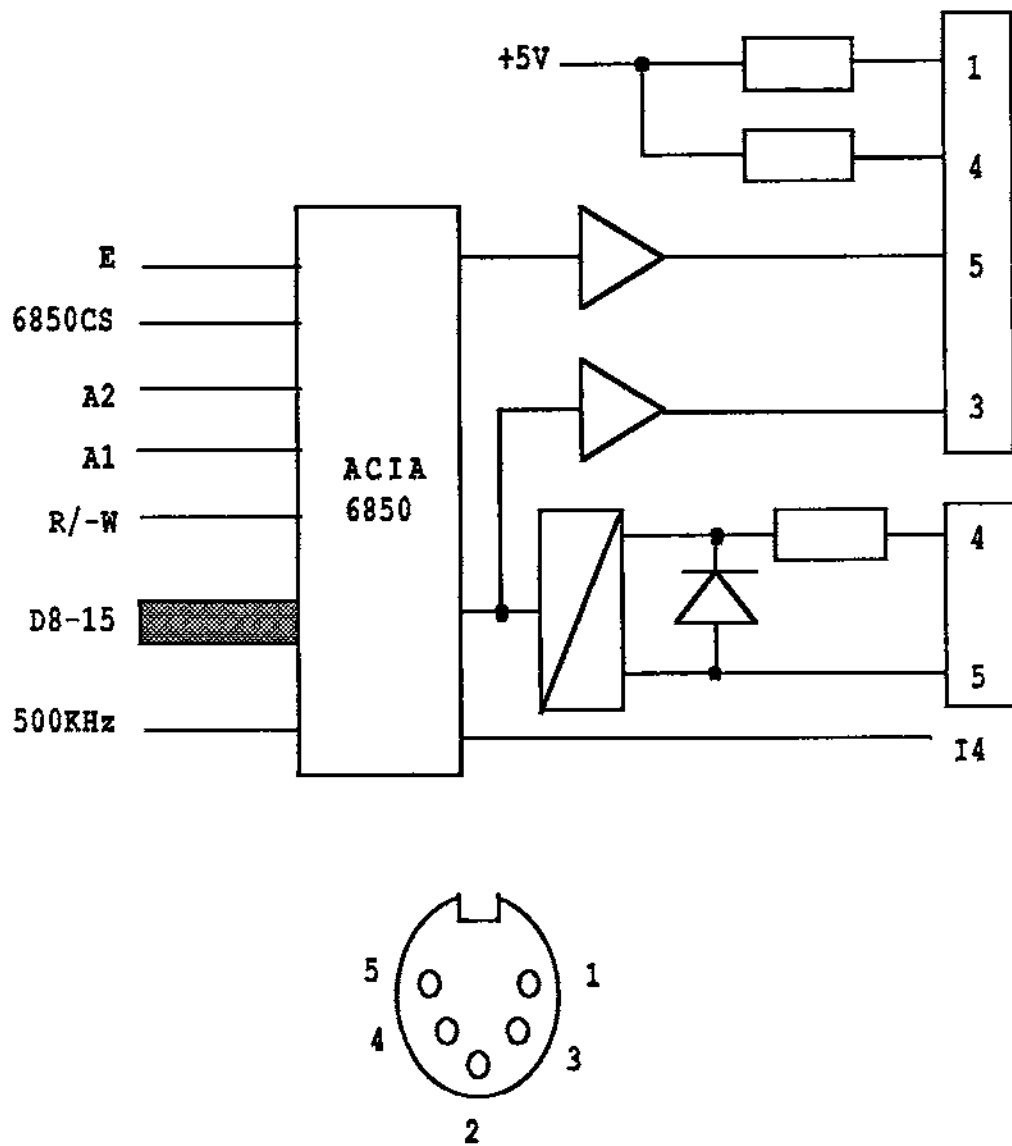
For this purpose the ST has the interfaces MIDI-IN and MIDI-OUT. The interfaces are even supported by the XBIOS so you don't have to worry about their actual operation.

The current loop travels on pins 4 and 5, out through pin 4 (+) of MIDI-OUT and in at 5, when a device is connected.

For MIDI-IN the situation is reversed because the current flows in through pin 4 and back out through pin 5. It goes through something called an optocoupler which electrically isolates the computer from the sender.

The receive data are looped back to MIDI-OUT (pins 1 and 3), which implements the MIDI-THRU function, although not entirely according to the standard.

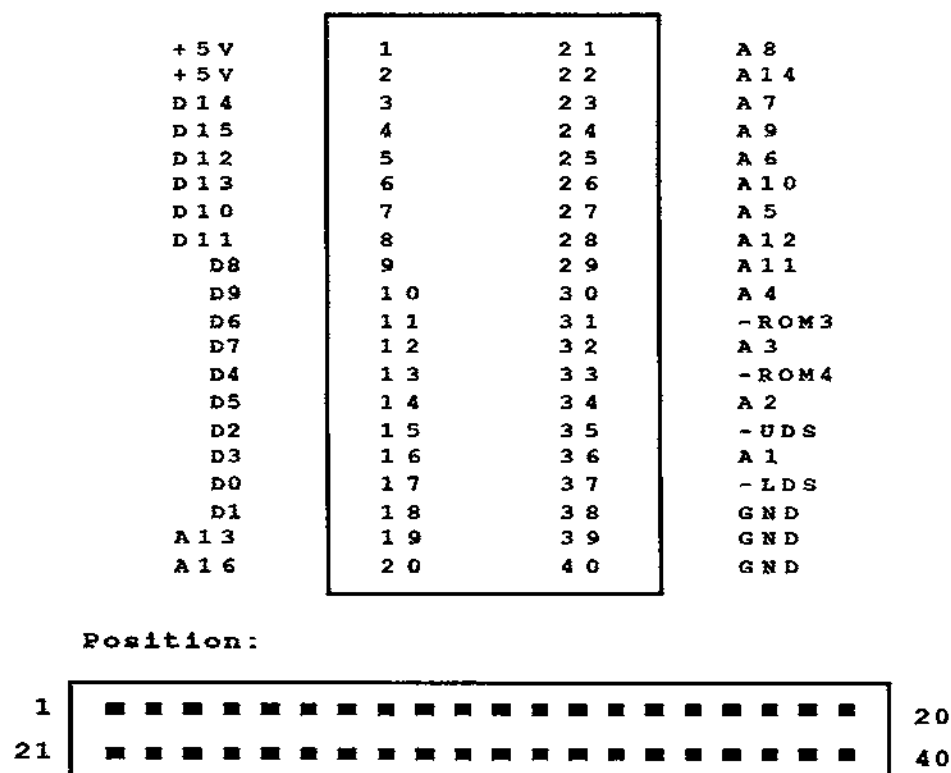
Figure 2.5-1 MIDI System Connection



## 2.6 The Cartridge Slot

The cartridge slot can be used *exclusively* for inserting ROM cartridges. Up to 128K in the address space \$FA0000 to \$FBFFFF can be addressed. The reason we stressed the exclusivity of the read access is the following. We thought it would be practical to outfit a cartridge with RAM and then load programs into it after the system start which would still remain after a reset. In order to try this we brought the R/-W signal to the outside. The experience taught us, however, that a write access to these addresses creates a bus error. The GLUE takes care of this. As you see, nothing is left to chance in the Atari.

### Figure 2.6-1 The Cartridge Slot



## 2.7 The Floppy Disk Interface

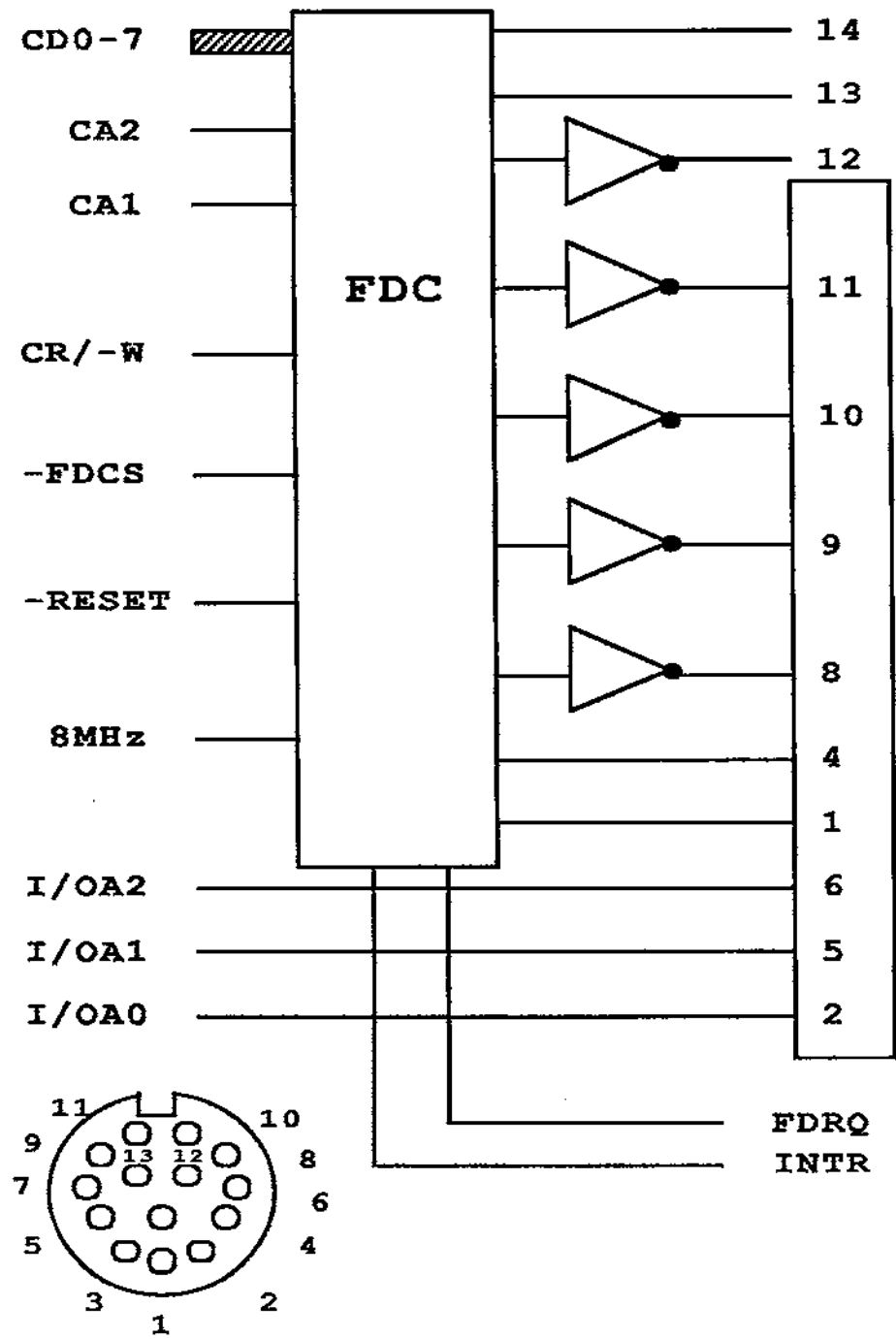
The interface for floppy disk drives is conspicuous because of the unusual connector, a 14-pin DIN connector. All of the signals required for the operation of two disk drives are available on it.

You know most of the signals from the description of the disk controller 1772, since nine of the available connections are directly or via a buffer connected to the controller. Only the drive select 1 and drive select 2 signals and the side 0 select are not derived from the disk controller. These signals come from port A of the sound chip.

Pinout of the disk connector:

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

Figure 2.7-1 Disk Connection





## 2.8 The DMA Interface

Up to 8 external devices can be connected to this 19-pin subminiature D connector. Such devices include hard disks, networks, and also coprocessors. The communication between the external devices and the ST runs at speeds up to 1 million bytes per second. Unfortunately, no experiments with DMA devices could be performed at the time this was printed. For this reason we cannot make the following statements with one hundred percent certainty.

The RESET line on pin 12 permits devices to be reset by the Atari. If this pin is low, as is the case when the Atari is turned on or when executing a RESET command, external devices are placed in a defined power-up state, without having to individually turn each device off and then on again.

Since most of the external devices will use a controller IC, the signal CS, Chip Select on pin 9, must also be available. The signal A1 is also to be seen in connection with this, because it is then important if the controller has more than just one register. This signal can distinguish between two registers.

The data transfer takes place over the bidirectional data bus on pins 1 to 8. The R/W line on the DMA bus determines the direction of the data transfer. The DMA chip can either write data to the bus (R/W is high), or read data from the bus (R/W is low). Data can be read or written only on the request of the external device. The release of a transfer is signaled by the signal DRQ (pin 19).

The ACK signal on pin 14 appears to be a purely hardware-dependent confirmation of the DRQ signal. The actual significance could not be checked however.

The last signal on the DMA port is the INT input. A low on this connection can generate an interrupt. The hard disk, for example, signals the end of the command through a low. The interrupt uses the same interrupt input on the MFP as the disk controller. This is input I/O 5. This means the at the floppy disk drive and the hard disk cannot transfer data together. The DMA is also not in such a position since it has only one DMA channel available.

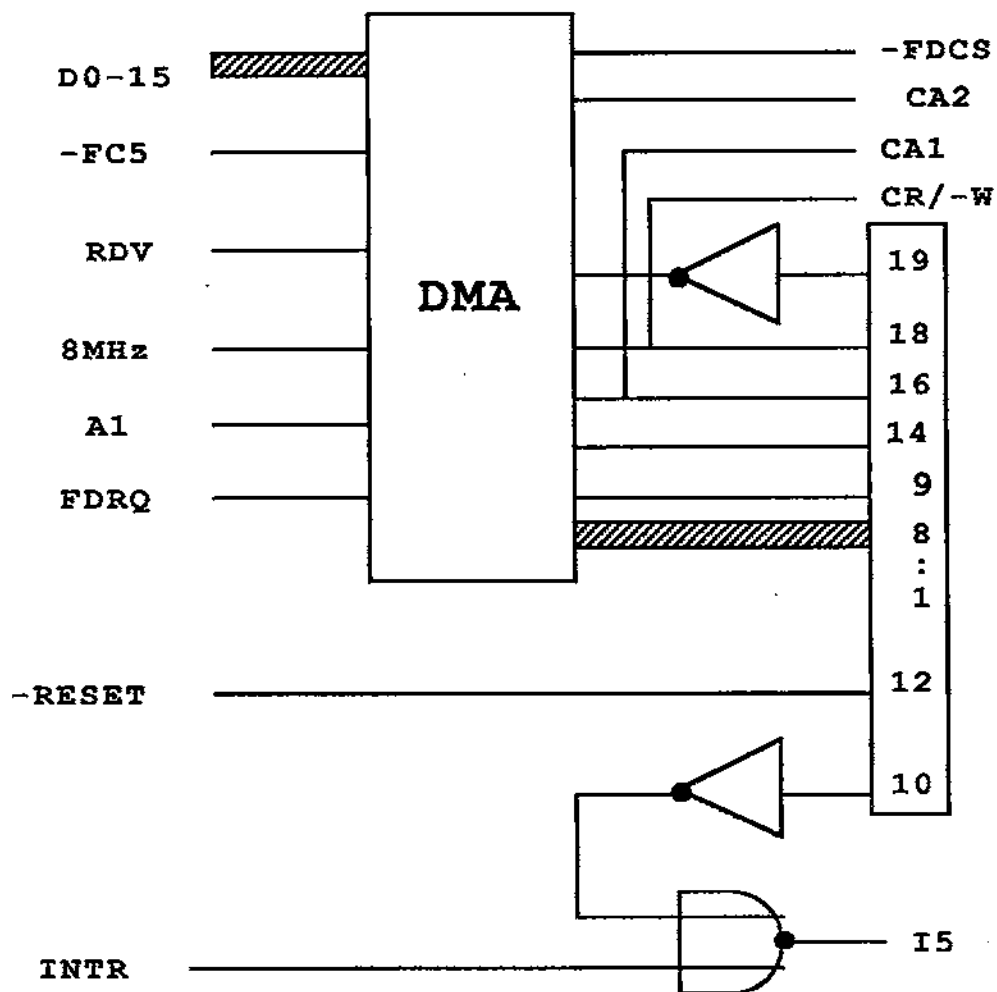
The interrupt of this input is disabled in the MFP internally because the floppy as well as the hard disk routines check the port bit in a loop in order

to determine the end of the command. This simplifies the implementation of the time out, which is always generated when the floppy or hard disk has not reacted to the command within a certain length of time.

Figure 2.8-1 DMA Port



Figure 2.8-2 DMA Connections



# Chapter 3

## The ST Operating System

- 3.1 The GEMDOS**
  - 3.1.1 GEMDOS error codes and their meaning**
- 3.2 The BIOS Functions**
- 3.3 The XBIOS**
- 3.4 The Graphics**
  - 3.4.1 An overview of the line-A variables**
  - 3.4.2 Examples for using the line-A opcodes**
- 3.5 The Exception Vectors**
  - 3.5.1 The interrupt structure of the ST**
- 3.6 The ST VT52 Emulator**
- 3.7 The ST System Variables**
- 3.8 The 68000 Instruction Set**
  - 3.8.1 Addressing modes**
  - 3.8.2 The instructions**
- 3.9 The BIOS listing**



## The ST Operating System

GEMDOS--what is it? Is it in the ST? The operating system is supposed to be TOS, though. Or CP/M 68K? Or what?

This question can be answered with few words. The operating system in the ST is named TOS--Tramiel Operating System--after the head of Atari. This TOS, in contrast to earlier information has nothing to do with CP/M 68K from Digital Research. At the start of development of the ST, CP/M 68K was implemented on it, but this was later changed because CP/M 68K is not exactly a model of speed and efficiency. A 68000 running at 8MHz and provided with DMA would be slowed considerably by the operating system.

At the beginning of 1985, Digital Research began developing a new operating system for 68000 computers, which would include a user-level interface. This operating system was named GEMDOS. It is exactly this GEMDOS which makes up the hardware-independent part of TOS. Like CP/M, TOS consists of a hardware-dependent and a hardware-independent part. The hardware-dependent part is the BIOS and the XBIOS, while the hardware-independent part is called GEMDOS. A large number of functions are built into GEMDOS, through which the programmer can control the actual input/output functions of the computer. Functions for keyboard input, text output on the screen or printer, and the operation of the various other interfaces are all present. Another quite important group contains the functions for file handling and for logical file and disk management.

### 3.1 The GEMDOS

When you look at the functions available under GEMDOS, you will eventually come to the conclusion that the whole thing is not really new. All the functions in GEMDOS are very similar to the functions of the MS-DOS operating system. Even the functions numbers used correspond to those of MS-DOS. But not all MS-DOS functions are implemented in GEMDOS. Especially in the area of file management, only the UNIX compatible functions are implemented in GEMDOS. The "old" block-oriented functions which are included in MS-DOS to maintain compatibility with CP/M are missing from GEMDOS. Also, special functions relating to the hardware of MS-DOS computers (8088 processor) are missing.

Another essential difference between MS-DOS and GEMDOS is that for GEMDOS calls as well as for the BIOS and XBIOS, the function number, the number of the desired GEMDOS routine, and the required parameters are placed on the stack and are not passed in the registers. The 68000 is particularly suited to this type of parameters passing. GEMDOS is called with TRAP #1 and the function is executed according to the contents of the parameter list. After the call, the programmer must put the stack back in order himself, by clearing the parameters from memory.

The basic call of GEMDOS functions differs from the BIOS and XBIOS calls only in the trap number.

In regard to all GEMDOS calls, it must be noted that registers D0 and A0 are changed in all cases. If a value is returned, it is returned in D0, or D0 may contain an error number, and after the call A0 (usually) points to the stack address of the function number. Any parameters required in D0 or A0 must be placed there before GEMDOS is called.

The remainder of this section describes the individual GEMDOS functions.

## \$00 TERM

Calling GEMDOS with function number 0 ends the running program and returns to the program from which it was started. For applications, programs started from the desktop, program control is returned to the desktop. If the program was called from a different program, execution is passed back to the calling program. This point is important for chaining program segments.

---

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

---

## \$01 CONIN

CONIN fetches a single character from the keyboard. The routine waits until a character is available. The result, the character read from the keyboard, is returned in the D0 register. The ASCII code of the pressed key is returned in the low byte of the low word, while the low byte of the high word of the register contains the scan code returned from the keyboard. This is important when reading keys which have no ASCII code. This applies to the 10 function keys or the keys of the cursor block, for example. These keys return the ASCII value zero when pressed.

If needed, the scan code can be used to determine if the digits on the keypad or the main keyboard were pressed, since these keys have identical ASCII codes, but return different scan codes.

---

```
.
MOVE.W #1,-(SP)    * Function number on the stack
TRAP    #1         * Call GEMDOS
ADDQ.L  #2,SP      * Correct stack
.
```

---

## \$02 CONOUT

CONOUT represents the simplest and most primitive character output of GEMDOS. With this function only one character is printed on the screen. The character to be displayed is placed on the stack as the first word. The ASCII value of the character to be printed must be in the low byte of the word and the high byte should be zero.

The character printed by CONOUT is outputted to device number 2, the normal console output. Control characters and escape sequences are interpreted normally.

---

```
MOVE.W #'A',-(SP)    * Output an A
MOVE.W #2,-(SP)      * CONOUT
TRAP    #1           * Call GEMDOS
ADDQ.L  #4,SP        * Correct stack
```

---

## \$03 AUXILLIARY INPUT

Under the designation "auxilliary port" is the RS-232 interface of the ST. A character can be read from the interface with the function CAUXIN. The function returns when a character has been completely received. The character is returned in the lower eight bits of register D0.

## \$04 AUXILLIARY OUTPUT

Similar to the input of characters via the serial interface, a character can be sent with this function. With this function the programmer should clear the upper eight bits of the word and pass the character to be sent in the lower eight bits.



---

## \$05 PRINTER OUTPUT

PRINTER OUTPUT is the simplest method of operating a printer connected to the Centronics interface. One character is printed with each call.

An important part of PRINTER OUTPUT is the return value in D0. If the character was sent to the printer, the value -1 (\$FFFFFFFF) is returned in D0. If, after 30 seconds, the printer was unable to accept the character (not turned on, OFF LINE, no paper, etc.), GEMDOS returns a time out to the program. D0 then contains a zero.

---

```
MOVE.W #'A',-(SP)    * Output an A
MOVE.W #5,-(SP)      * Function number
TRAP    #1            * Call GEMDOS, output character
ADDQ.L  #4,SP         * Correct stack
TST.W   D0            * Affect flags
BEQ     printererror
```

---

## \$06 RAWCONIO

RAWCONIO is a somewhat unusual mixture of keyboard input and screen output and receives a parameter on the stack.

With a function value of \$FF the keyboard is tested. If a character is present, the ASCII code and scan code are passed in D0 as described for CONIN. But if no key value is present, the value zero is passed as both the ASCII code and the scan code in D0. The call to RAWCONIO with parameter \$FF is comparable to the BASIC INKEY\$ function.

If a value other than \$FF is passed to the function, the value is interpreted as a character to be printed and it is output at the current cursor position. This output also interprets the control characters and escape sequences properly.

---

START:

```
MOVE.W #$FF,-(SP)  * Function value test keyboard
MOVE.W #6,-(SP)    * Function number
TRAP   #1          * Call GEMDOS, test keyboard
ADDQ.L #4,SP       * Correct stack
TST.W  D0          * Character arrived?
BEQ     START      * Not yet
CMP.B  #3,D0       * ^C selected as the end marker
BEQ     END
MOVE   D0,-(SP)    * Character for output on the stack
MOVE   #6,-(SP)    * Function number
TRAP   #1          * Call GEMDOS, test keyboard
ADDQ.L #4,SP       * Correct stack
BRA    START       * Get new character
```

---

## **\$07 DIRECT CONIN WITHOUT ECHO**

The function \$07 differs from \$01 only in that the character received from the keyboard is not displayed on the screen. It waits for a key just as does CONIN.

## **\$08 CONIN WITHOUT ECHO**

Function \$08 does not differ from function \$07. Both function calls have exactly the same effect. The reason for this seemingly nonsensical behavior lies in the mentioned compatibility to MS-DOS. Under MS-DOS the two functions are different in that with \$08, certain keys not present on the ATARI are evaluated correctly, while this evaluation does not take place with function \$07.

## \$09 PRINT LINE

You have already become familiar with functions to output individual characters on the screen with CONOUT and RAWCONIO. PRINT LINE offers you an easy way to output text. An entire string can be printed at the current cursor position with this function. To do this, the address of the string is placed on the stack as a parameter. The string itself is concluded with a zero byte. Escape sequences and control characters can also be evaluated with this function.

After the call, D0 contains the number of characters which were printed. The length of the string is not limited.

---

```

MOVE.L  #text, -(SP)    * Address of the string on the stack
MOVE     #$09, -(SP)    * Function number PRT LINE
TRAP     #1              * Call GEMDOS
ADDQ.L   #6, SP          * "Clean up" the stack
.
.
text     .DC.B 'This is the string to be printed', $0D, $0A, 0

```

---

## \$0A READLINE

READLINE is a very easy-to-use function for reading characters from the keyboard. In contrast to the "simpler" character-oriented input functions, an entire input line can be fetched from the keyboard with READLINE. The characters entered are displayed on the screen at the same time.

The address of an input buffer is passed to the function as the parameter. The value of the first byte of the input buffer determines the maximum length of the input line and must be initialized before the call. At the end of the routine, the second byte of the buffer contains the number of characters entered. The characters themselves start with the third byte.

The routine used by READLINE for keyboard input is quite different from the character-oriented console inputs. Escape sequences are not interpreted during the output. Only control characters like control-H (backspace) and control-I (TAB) are recognized and handled appropriately. The following control characters are possible:

^C Ends input AND program (!)  
^H Backspace one position  
^I TAB  
^J Linefeed, end input  
^M CR, end input  
^R Entered line is printed in new line  
^U Don't count line, start new line  
^X Clear line, cursor at start of line

A function like ^H, deleting a character entered, is useful, but for large programs you should write your own input routine because ^C is very "dangerous." Unlike CP/M, the program will be ended even if the cursor is not at the very start of the input line.

If more characters are entered than were indicated in the first byte of the buffer at the initialization, the input is automatically terminated. If the input is terminated by ENTER, ^J, or ^M, the terminating character will not be put in the buffer.

After the input, D0 contains the number of characters entered, excluding ENTER, which can be found at buffer+1.

## **\$0B CONSTAT**

All key presses are first stored in a buffer in the operating system. This buffer is 64 bytes in length. The key values stored there are taken from the buffer when a call to a GEMDOS output routine is made.

CONSTAT can be used to check if characters are stored in the keyboard buffer. After the call, D0 contains the value zero or \$FFFF. A zero in D0 indicates that no characters are available.

## **\$0E SETDRV**

The current drive can be determined with the function SETDRV. A 16-bit parameter containing the drive specification is passed to the routine. Drive A is addressed with the number 0 and drive B with the number 1.

After the call, D0 contains the number of the drive active before the call.

## **\$10 CONOUT STAT**

CONOUT STAT returns the status of the console in D0. If the value \$FFFF is returned, a character can be displayed on the screen. If the returned value is zero, however, no character output is possible on the screen at that time. Incidentally, all attempts to create a not-ready status at the console failed. The only imaginable possibility for the not-ready status would be if the output of the individual bit pattern of a character was interrupted and the interrupt routine itself tried to output a character. This case could not, however, be created.

## **\$11 PRTOUT STAT**

This function returns the status, the condition of the Centronics interface. If no printer is connected (or turned off, or off line), D0 contains the value zero after the call to indicate "printer not available." If, however, the printer is ready to receive, D0 contains the value \$FFFF.

## **\$12 AUXIN STAT**

By calling AUXIN STAT you can determine if a character is available from the receiver of the serial interface (\$FFFF) or not (\$0000). As with all other functions, the value is returned in D0.

## **\$13 AUXOUT STAT**

AUXOUT STAT gives information about the state of the serial bus. A value of \$FFFF indicates that the serial interface can send a character, while zero indicates that no characters can be sent at this time.

## **\$19 CURRENT DISK**

For many applications it is necessary to know which drive is currently active. The current drive can be determined by the function \$19. After the call, D0 contains the number of the drive. The significance of the drive numbers is the same as for \$0E, SET DRIVE (0=A, 1=B).

---

## \$1A SET DISK TRANSFER ADDRESS

The disk transfer address is the address of a 44-byte buffer required for various disk operations (especially directory operations). Along with the GEMDOS functions SEARCH FIRST and SEARCH NEXT are examples for using the DTA.

---

```
MOVE.L #DTADDRESS, -(SP) * Address of the 44-byte DTA buffer
MOVE.W #$1A, -(SP)      * Function number SET DTA
TRAP    #1               * Set DTA
ADDQ.L #6, SP            * Clean up the stack
```

---

## \$20 SUPER

This function is especially interesting for programmers who want to access the peripheral components or system variables available only in the supervisor mode while running a program in the user mode. After calling this function from the user mode, the 68000 is placed in the supervisor mode. In contrast to the XBIOS routine for enabling the supervisor mode, additional GEMDOS, BIOS, and XBIOS calls can be made after a successful SUPER call.

First we will look at the case in which the SUPER function is called from a program in the user mode with a value of zero on the stack. In this case the program finds itself in the supervisor mode after the call. The supervisor stack pointer is set to the value of the user stack pointer and the original value of the supervisor stack pointer is returned in D0. This value should be stored by the program in order to get back into the user mode later.

If a value other than zero is passed to the SUPER function the first time it is called, this value is interpreted as the desired value of the supervisor stack pointer. In this case as well, D0 contains the original value of the supervisor stack pointer, which the program should save.

Before a program ends, the user mode should be reenabled. This change of operating modes requires the address acquired the first time the routine was called in order to set the supervisor stack pointer back to its original value.

The SUPER function differs from all other GEMDOS functions in one very important respect. Under certain circumstances, this call can also change the contents of A1 and D1. If you store important values in these registers, you must save the values somewhere before calling the SUPER function.

---

```

CLR.L  -(SP)          * The 6800 is in the user mode
MOVE.W #$20,-(SP)     * User stack becomes supervisor stack
TRAP   #1              * Call SUPER
                        * The supervisor mode is active
                        * after the TRAP
ADD.L  $6,SP           * D0 = old supervisor stack
MOVE.L D0,_SAVE_SSP   * Save value
.
.
.
.   Here processing can be done in the supervisor mode
.
.
.
MOVE.L _SAVE_SSP,-(SP) * Old supervisor stack pointer
MOVE.W #$20,-(SP)     * Call SUPER
TRAP   #1              * Now we are back in the user mode
ADD.L  $6,SP
.
.
.

```

---

## \$2A GET DATE

You have no doubt experimented with the status field at one time or another. In addition to various other functions, the status field contains a clock with clock time and date. It can be useful for some applications to have the data available. The date can be easily determined by the GET DATE function. This function call requires no parameters and makes the date available in the low word of register D0. It is rather thoroughly encoded, though, so that the result in D0 must be prepared in order to get the correct date.

The day in the range 1 to 31 is coded in the lower five bits. Bits 5 to 8 contain the month in the value range 1 to 12, and the year is contained in

bits 9 to 15. The value range in these "year bits" goes from 0 to 119. The value of these bits must be added to the value 1980 in order to get the actual year. The date 12/12/1992, for example, would result in \$198C in D0. This can be represented in binary as %0001100.1100.01100. The lengths of the three fields are marked with periods.

## \$2B SET DATE

The clock time and date can also be set from application programs. This is particularly interesting for programs which use the date and/or clock time. An example of this would be invoice processing in which the current date is inserted in the invoice. Such programs can then ask the user to enter the date. This avoids the problems that occur if the user forgets to set the date and clock time on the status field beforehand.

The date must be passed to the function SET DATE in the same format as it is received from GET DATE, bits 0-4 = day, bits 5-8 = month, bits 9-15 = year-1980.

---

```
MOVE.W  #%101101011001, -(SP)  * Set date to 10/25/1985
MOVE.W  #$2B, -(SP)           * Function number of SET DATE
TRAP    #1                    * Set date
ADDQ.L  #6, SP                * Repair stack
```

---

## \$2C GET TIME

The function GET TIME returns the current (read: set) time from the GEMDOS clock. Similar to the date, the clock time is coded in a special pattern in individual bits of the register D0 after the call. The seconds are represented in bits 0-4. But since only values from 0 to 31 can be represented in 5 bits, the internal clock runs in two second increments. In order to get the correct seconds-result the contents of these five bits must be multiplied by two. The number of minutes is contained in bits 5 to 10, while the remaining bits 11-15 give information about the hour (in 24-hour format).



## \$2D SET TIME

It is also possible to set the clock time under GEMDOS. The function SET TIME expects a 16-bit value (word) on the stack, in which the time is coded in the same form as that in which GET TIME returns the clock time.

---

```
MOVE.W #%1000101010111101,-(SP) * Clock time 17:21:58
MOVE.W #$2D,-(SP)                * Function # of GET TIME
TRAP #1                          * Set date
ADDQ.L #6,SP                      * Repair stack
```

---

## \$2F GET DTA

The function \$2F is the counterpart of function \$1A, SET DTA. A call to this function returns the address of the current disk transfer buffer in D0. An exact explanation of this buffer is found together with the functions SEARCH FIRST and SEARCH NEXT.

## \$30 GET VERSION NUMBER

Calling this function returns in D0 the version number of GEMDOS. In the version of GEMDOS currently in release, this question is always answered with \$0D00, corresponding to version 13.00. Official Atari documentation claims that a value of \$0100 should be returned for this version, though perhaps the value should indicate that the present GEMDOS version is the \$D = diskette version.

## \$31 KEEP PROCESS

This function is comparable to the GEMDOS function TERM \$00. The program is also ended after a call to this function. \$31 does differ from \$00 in several important points.

After processing TRAP #1, like TERM, control is passed back to the program which started the program just ended. In contrast to TERM, a termination condition can be communicated to the caller. While TERM

returns the termination value zero (no error), zero or one may be selected as the termination value for \$31. A value other than zero means that an error occurred during program processing.

Another essential point lies in the memory management of GEMDOS. When a program is started, the entire available memory space is made available to it. If the program is ended with TERM, the memory space is released and made available to GEMDOS. The entire area of memory released is also cleared, filled with zeros. The program actually physically disappears from the memory. With function \$31, however, an area of memory can be protected at the start address of the program. This memory area is not released when the program is ended and it is also not cleared. The program could be restarted without having to load it in again.

KEPP PROCESS is called with two parameters. The example programs shows the parameter passing.

---

```
MOVE.W #0,-(SP)      * Error code no error, else 1
MOVE.L #$1000,-(SP)  * Protect $1000 bytes at program start
MOVE.W #$31,-(SP)    * Function number, end program
TRAP    #1           *                               now
```

---

## \$36 GET DISK FREE SPACE

It can be very important for disk-oriented programs to determine the amount of free space on the diskette. Then you have the ability to request that the user change disks at the appropriate time. "Disk full" messages or even data loss can then be avoided.

Function \$36 returns exactly this information. The number of the desired disk drive and the address of a 16-byte buffer must be passed to the function. If the value 0 is passed as the drive number, the information is fetched from the active drive, a 1 takes the information from drive A, and a 2 from drive B.

The information passed in the buffer is divided into four long words. The first long word contains the number of free allocation units. Each file, even if it is only eight bytes long, requires at least one such allocation unit.

The second long word gives information about the number of allocation units present on the disk, regardless of whether they are already used or are still free. For the "small," single-sided diskettes this value is \$15C or 351, while the double-sided disks have \$2C7 = 711 allocation units. The third long word contains the size of a disk sector in bytes. For the Atari this is always 512 bytes (\$200 bytes).

In the last word is the number physical sectors belonging to an allocation unit. This is normally 2. Two sectors form one allocation unit.

The number of available bytes of disk space can easily be calculated from this information.

---

```

MOVE.W #0,-(SP)      * Information from the active drive
MOVE.L #BUFFER,-(SP) * Address of the 16-byte buffer
MOVE     #$36,-(SP)  * Function number
TRAP     #1
ADDQ.L   #8,SP       * Clean up stack
.
.
.
.bss
BUFFER:
freal: .ds.1 1      * Free allocation units
total: .ds.1 1      * Total allocation units
bps:   .ds.1 1      * Bytes/physical sector
pspal: .ds.1 1      * Phys. sectors/alloc. unit

```

---

## \$39 MKDIR

A subdirectory can be created from the desktop with the menu option "NEW FOLDER". Such a subdirectory can also be created from an application program with a call to \$39.

In order to create a new folder, the function \$39 is given the address of the folder name, also called the pathname. This name may consist of 8 characters and a three-character extension. The same limitations apply to pathnames as do to filenames. The pathname must be terminated with a zero byte when calling MKDIR.

After the call, D0 indicates whether the operation was performed successfully. If D0 contains a zero, the call was successful. Errors are indicated through a negative number in D0. At the end of this chapter you will find an overview of all of the error messages occurring on connection with GEMDOS functions.

---

```

MOVE.L #pathname      * Address of the pathname
MOVE    #$39,-(SP)     * Function number
TRAP    #1
ADDQ.L  #6,SP          * Repair stack
TST.W   D0             * Error occurred?
BNE     error          * Apparently
.
.
.
pathname:
        .dc.b  'private.dat',0

```

---

### \$3A RMDIR

A subdirectory created with MKDIR can be removed again with \$3A. As before, the pathname, terminated with a zero, is passed to RMDIR. The error messages also correspond to those for MKDIR, with zero for success or a negative value for errors. An important error message should be mentioned at this point. It is the message -36 (\$FFFFFFCA). This is the error message you get when the subdirectory you are trying to remove contains files.

Only empty subdirectories can be removed with RMDIR. In the event of the described error message, one must first erase all of the files in the directory with UNLINK (\$41) and then call RMDIR again.

## \$3B CHDIR

The system of subdirectories available under GEMDOS is exactly the same form available under UNIX. This system is now running on systems with diskette drives, but its advantages become noticeable first when a large mass storage device such as a hard disk with several megabytes of storage capacity is connected to the system. After a while, most of the time would probably be spent looking for files in the directory.

To better organize the data, subdirectories can be placed within subdirectories. It can therefore become necessary to specify several subdirectories until one has the directory in which the desired file is stored. An example might be:

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

Translated this would mean: load the file `cqsort.s` from the subdirectory `csort.s`. This subdirectory `csort.s` is found in the subdirectory `cfiles.s`, which in turn is a subdirectory of `hugos.dat`. If the whole expression is given as a filename, the desired file will actually be loaded (assuming that the file and all of the subdirectories are present). If you want to access another file via the same path (do you understand the term *pathname*?), the entire path must be entered again. But you can also make the subdirectory specified in the path into the current directory, by calling CHDIR with the specification of the desired path. After this, all of the files in the selected subdirectory can be accessed just by the filenames. The path is set by the function.

---

```

MOVE.L #path,-(SP)      * Address of the path
MOVE.W #$3B,-(SP)      * Function number
TRAP    #1
ADDQ.L #6,SP           * Repair stack
TST.W  D0              * Error occurred?
BNE     error          * Apparently
.
.
.
path:
      .dc.b  '/hugos.dat/private.dat/',0

```

---

## \$3C CREATE

In all operating systems, the files are accessed through the sequence of opening the file, accessing the data, (reading or writing), and then closing the file. This "trinity" also exists under GEMDOS, although there is an exception. Under CP/M, for example, a non-existing file can also be opened. When a file which does not exist is opened, it is created. Under GEMDOS, the file must first be created. The call \$3C, CREATE, is used for this purpose. Two parameters are passed to this GEMDOS function: the address of the desired filename, and an attribute word.

If a zero is passed as the attribute word, a normal file is created, a file which can be written to as well as read from. If the value 1 is passed as the attribute, the file will only be able to be read after it is closed. This is a type of software write-protect (which naturally cannot prevent the file from disappearing if the disk is formatted).

Other possible attributes are \$02, \$04, and \$08. Attribute \$02 creates a "hidden" file and attribute \$02 a "hidden" system file. Attribute \$08 creates a file with a "volume label." The volume label is the (optional) name which a disk can be given when it is formatted. The disk name is then created from the maximum of 11 characters in the name and the extension. Files with one of the last three attributes are excluded from the normal directory search. On the ST, however, they do appear in the directory.

When the function CREATE is ended, a file descriptor, also called a file handle, is returned in D0. All additional accesses to the file take place over this file handle (a numerical value between 6 and 45). The handle must be given when reading, writing, or closing files. A total of  $2^8 = 256$  files can be opened at the same time.

If CREATE is called and a file with this name already exists, it is cut off at zero length. This is equivalent to the sequence delete the old file and create a new file with the same name, but it goes much faster.

If after calling CREATE you get a handle number back in D0, the file need not be opened again with \$3D OPEN.

---

```

MOVE.W #$0,-(SP)      * File should have R/W status
MOVE.L #filename,-(SP) * Address of the filename on stack
MOVE.W #$3C,-(SP)     * Function number
TRAP    #1            * Call GEMDOS
ADDQ.L  #8,SP         * Clean up stack
TST     D0            * Error occurred?
BMI     error         * It appears so
MOVE    D0,handle     * Save file handle
.
.
.
filename:              * Don't forget zero byte
        .dc.b  'myfile.dat',0
.
handle:
        .ds.w  1

```

---

### \$3D OPEN

You can create only new files with CREATE, or shorten existing files to length zero. But you must be able to process existing files further as well. To do this, such files must be opened with the OPEN function.

The first parameter of the OPEN function is the mode word. With a zero in the mode word, the opened file can only be read, with one it can only be written. With a value of 2, the file can be read as well as written. The filename, terminated with zero byte in the usual manner, is passed as the second parameter.

The OPEN function returns the handle number in D0 as the result if the file is present and the desired access mode is possible. Otherwise D0 contains an error number. See the end of the chapter for a list of the error numbers.

---

```

MOVE.W #$2,-(SP)      * File read and write
MOVE.L #filename      * Address of the filename on the stack
MOVE.W #$3D,-(SP)     * Function number
TRAP    #1            * Call GEMDOS
ADDQ.L  #8,SP         * Clean up the stack
TST.W   D0            * Error occurred?
BMI     error         * Apparently
MOVE    D0,handle     * Save file handle for later accesses
.
.
.
filename:              * Don't forget zero byte!
        .dc.b    'myfile.dat',0
.
handle:
        .ds.w    1

```

---

### \$3E CLOSE

Every opened file should be closed when it will no longer be accessed within a program, or when the program itself is ended. Especially when writing, files must absolutely be closed before the program ends or data may be lost.

Files are closed by a call to CLOSE, to which the handle number is passed as a parameter. The return value will be zero if the file was closed correctly.

---

```

MOVE.W handle,-(SP)   * Handle number
MOVE.W #$3E,-(SP)    * Function number
TRAP    #1            * Call GEMDOS
ADDQ.L  #4,SP         * Error occurred?
BMI     error         * Apparently
.
.
.
handle:
        .ds.w    1

```

---



## \$3F READ

Opening and closing files is naturally only half of the matter. Data must be stored and the retrieved later. Reading such files can be done in a very elegant manner with the function READ. READ expects three parameters: first the address of a buffer in which the data is to be read, then the number of bytes to be read from the file, and finally the handle number of the file. This number you have (hopefully) saved from the previous OPEN.

We mentioned the possible handle numbers in conjunction with CREATE. What we didn't mention, however, is why the first handle number is six. The cause of this is that things called devices, like the keyboard, the screen, the printer, and the serial interface, are also accessed via handle numbers for READ and WRITE operations. The device assignments are:

```
0 = Console input
1 = Console output
2 = RS-232
3 = Printer
```

Numbers 4 and 5 also function as console input and output. When using these handle numbers, the system sometimes returns "invalid handle number". The correct programming and the exact purpose of these two numbers is not known.

As return value, D0 contains either an error number (hopefully not) or the number of bytes read without error. No message regarding the end of the file is returned. This is not necessary, however, since the size of the file is contained in the directory entry (see SEARCH FIRST/SEARCH NEXT). If the file is read past the logical end, no message is given. The reading will be interrupted at the end of the last occupied allocation unit of the file. The number of bytes read in this case is always divisible by \$400.

---

```

MOVE.L #buffer, -(SP)    * Address of the data buffer
MOVE.L #$100, -(SP)      * Read 256 bytes
MOVE.W handle, -(SP)     * Space for the handle number
MOVE.W #$3F, -(SP)       * Function number
TRAP    #1
ADD.L   #12, SP
TST.L   D0                * Did an error occur
BMI     error             * Apparently
.
.
handle:
        .ds.w   1          * Space for the handle number
.
buffer:
        .ds.b   $100       * Suffices in our example

```

---

## \$40 WRITE

Writing to a file is just as simple as reading from it. The parameters required are also the same as those required for reading. The file descriptors from OPEN and CREATE calls can be used as the handle, but the device numbers listed for READ can also be used. The output of a program can be sent to the screen, the printer, or in a file just by changing the handle number.

## \$41 UNLINK

Files which are no longer needed can be deleted with UNLINK. To do this, the address of the filename or, if necessary, the complete pathname must be passed to the function. If the D0 register contains a zero after the call, the file has been deleted. Otherwise D0 will contain an error number.

```
MOVE.L pathname,-(SP)    * Address of the data buffer
MOVE.W #$41,-(SP)       * Function number
TRAP    #1
ADD.L   #6,SP
TST.W   D0               * Did an error occur?
BMI     error           * Apparently
.
.
.
pathname:
        .dc.b    '/rolli/private/pacman.prg',0
```

---

## \$42 LSEEK

Up to now we have become acquainted only with sequential data accesses. We can read through any file from the beginning until we come the desired information. An internal file pointer which points to the next byte to be read goes along with each read. We can only move this pointer continuously in the direction of the end of file by reading. A few bytes forward or backward, setting the pointer as desired, is not something we can do. This is required for many applications, however.

LSEEK offers an extraordinarily easy-to-use method of setting the file pointer to any desired byte within the file and to read or write at this point. This UNIX-compatible option of GEMDOS is much easier to use than the methods available under CP/M for relative file management, for instance.

A total of three parameters are passed to the LSEEK function. The first parameter specifies the number of bytes by which the pointer should be moved. An additional parameter is the handle number of the file. The last parameter is a mode word which describes how the file is to be moved. A zero as the mode moves the pointer to the start of the file and from there the given number of bytes toward the end of the file. Only positive values may be used as the number. With a mode value of 1, the pointer is moved the desired positive or negative amount from the current position, and a 2 as the mode value means the distance specified is from the end of the file. Only negative values are allowed in this mode.

---

After the call, D0 contains the absolute position of the pointer from the start of the file, or an error message.

---

```

MOVE.W #1,-(SP)      * Relative from the current file ptr
MOVE.W handle,-(SP)  * File handle
MOVE.L #$-20,-(SP)   * 32 bytes back
MOVE.W #$42,-(SP)    * Function number
TRAP    #1
ADD.L   #10,SP
TST.W   D0            * Did an error occur?
BMI     error         * Apparently
.
.
handle:
        .ds.w 1        * Space for the handle number

```

---

### \$43 CHANGE MODE (CHMOD)

With the CREATE function a file can be assigned a specific attribute. This attribute can be determined and subsequently changed only with the function CHANGE MODE. The name of the file must be known because the address of the name or the complete pathname must be passed to CHMOD. Another parameter word specifies whether the file attribute is to be read or set. Moreover, a word must be passed which contains the new attribute. When reading the attribute of a file this word is not necessary, but should be passed to the routine as a dummy value. We indicated the possible file attributes in our discussion of the function CREATE, but here they are again in a table:

```

$00 = normal file status, read/write possible
$01 = File is READ ONLY
$02 = "hidden" file
$04 = system file
$08 = file is a volume label, contains disk name
$10 = file is a subdirectory
$20 = file is written and closed correctly

```

Attributes \$10 and \$20 cannot be specified when the file is created. Attribute \$20 is granted by the operating system, while the GEMDOS function

---

MKDIR is used to create a subdirectory. The MKDIR function creates not only the directory entry with the appropriate attribute, it also arranges the subdirectory on the disk physically.

After the call, D0 will contain the current attribute value, which will be the new value after setting the attribute, or, as with all other function calls, a negative error number.

---

#### First example:

```

MOVE.W #1,-(SP)      * Give file READ ONLY attribute
MOVE.W #1,-(SP)      * Set attribute
MOVE.L #pathname,-(SP) * We also need the pathname
MOVE.W #$43,-(SP)    * Function number
TRAP    #1
ADD.L   #10,SP
TST.W   D0            * Did an error occur?
BMI     error         * Apparently
.
.
.
pathname:            * Don't forget zero byte at end!
                .dc.b 'killme.not',0

```

#### Second example:

```

MOVE.W #0,-(SP)      * Dummy value, not actually required
MOVE.W #0,-(SP)      * Read attribute
MOVE.L #pathname,-(SP) * and the pathname
MOVE.W #$43,-(SP)    * Function number
TRAP    #1
ADD.L   #10,SP
TST.W   D0            * Did an error occur?
BMI     error         * Apparently
.
.
.
pathname:            * Don't forget zero byte at the end!
                .dc.b 'what-am.i',0

```

---

## **\$45 DUP**

As mentioned in connection with the functions READ and WRITE, the devices console, line printer, and RS-232 are also available to the programmer. This permits input and output to be redirected to these devices. One of the devices can be assigned a file handle number with the DUP function. After the call the next free handle number is returned.

## **\$46 FORCE**

The FORCE function allows further manipulation of the handle numbers. If in a program the console input and output are used exclusively via the READ and WRITE functions with the handle numbers 0 and 1, the input or output can be redirected with a call to this function. Screen outputs are written to a file, inputs are not taken from the keyboard, but from a previously-opened file.

## **\$47 GETDIR**

A given subdirectory can be made into the current directory with the function \$37. All file accesses with a pathname then run only in the set subdirectory. Under certain presumptions it can be possible to determine the pathname to the current subdirectory. This is accomplished by the function call GETDIR, \$47. This call requires the designation of the desired disk drive (0=current drive, 1=drive A, 2=drive B, etc.) and a pointer to a 64-byte buffer. The complete pathname to the current directory will be placed in this buffer. The pathname will be terminated by a zero byte. If the function is called when the main directory is active, no pathname will be returned. In this case, the first byte in the buffer will contain zero. After the call, D0 must contain the value zero. If the value is negative, an error occurred, for example if an incorrect drive number was passed.

---

```
MOVE.W #0,-(SP)      * Get pathname of the current drive
MOVE.L #buffer,-(SP) * Address of the 64-byte buffer
MOVE.W #$47,-(SP)    * Function number
TRAP    #1
ADDQ.L #8,SP
TST.L  D0              * Error?
BNE    error          * D0<>0 if error
.
.
.
buffer:
        .ds.b    64      * Buffer for pathname
```

---

## \$48 MALLOC

The MALLOC function and the two that follow it, MFREE and SETBLOCK, are concerned with the memory organization of GEMDOS. As already mentioned in conjunction with function \$31, KEEP PROCESS, a program is assigned all of the entire memory space available after it is loaded. This is uncritical in many cases, because only a single program is running. But there are applications under GEMDOS in which such organization is not sensible. An accessory such as the VT-52 emulator may be called from within a program, for example. Such a program also requires memory space, but the memory might not be available. No further program modules can be loaded if the entire memory is occupied. For this reason, each program should reserve only the space which it actually needs for the program and data. The memory not required can be given back to GEMDOS.

If the program should need some of the memory it gave back, it can request memory from GEMDOS via the function MALLOC (memory allocate). The number of bytes required is passed to MALLOC. After the call, D0 contains the starting address of the memory area reserved by the call or an error message if an attempt is made to reserve more memory than is actually available.

If -1L is passed as the number of bytes to be allocated, the number of bytes available is returned in D0.

First example:

```
MOVE.L #-1,-(SP)      * Determine number of free bytes
MOVE.W #$48,-(SP)     * Function number
TRAP    #1
ADDQ.L  #6,SP         * Number of free bytes in D0
.
.
```

Second example:

```
MOVE.L #$1000,-(SP)   * Get hex 1000 bytes for the program
MOVE.W #$48,-(SP)     * Function number
TRAP    #1
ADDQ.L  #6,SP
TST.W   D0            * Error or address of memory?
BMI     error         * Negative long word = error!
MOVE.L  D0,mstart     * Else start addr of the reserved area
.
.
mstart:
        .ds.l    1
```

---

## \$49 MFREE

An area of memory reserved with MALLOC can be released at any time with MFREE. To do this, GEMDOS is passed the address of the memory to be released. The value will usually be the address returned by MALLOC.

If a value of zero is returned in D0, the memory was released by GEMDOS without error. A negative values indicates errors.



---

```
MOVE.L mstart,-(SP)    * Addr of a previously allocated area
MOVE.W #$49,-(SP)      * Function number
TRAP    #1
ADDQ.L  #6,SP          * Number of free bytes in D0
TST.L   D0             * Error?
BNE     error          * D0<>0 is error!
.
.
mstart:
        .ds.l 1
```

---

## \$4A SETBLOCK

In contrast to the MALLOC function, a specific area of memory can be reserved with the function SETBLOCK. The memory beginning at the specified address is returned to GEMDOS, even if it was reserved before. This function can be used to reserve the actual memory requirements of a program and release the remaining memory.

The parameters the function requires are the starting address and the length of the area to be reserved. The area specified with these parameters is then reserved by GEMDOS and is not released again until the end of the program or after calling the MFREE function.

Usually programs will begin with the following command sequence or something similar. After the call, D0 must contain zero, otherwise an error occurred.

---

MOVE.L A7,A5	* Save stack pointer in A5
MOVE.L #USTCK,A7	* Set up stack for the program
MOVE.L 4(A5),A5	* A5 now points to the base-page start
	* exactly \$100 bytes below the prg start
MOVE.L \$(A5),D0	* \$(A5) contains length of the prg area
ADD.L \$14(A5),D0	* \$14(A5) containing the length of the
	* initialized data area
ADD.L \$1C(A5),D0	* \$1C(A5) contains length of the
	* uninitialized data area
ADD.L #\$100,D0	* Reserve \$100 bytes base page
MOVE.L D0,-(SP)	* D0 contains the length of the area
	* to be reserved
MOVE.L A5,-(SP)	* A5 contains the start of the area
	* to be reserved
MOVE.W #0,-(SP)	* Meaningless word, but still necessary!
MOVE.W #\$4A,-(SP)	* Function number
TRAP #1	
ADD.L #12,SP	* Clean up the stack as usual
TST.L D0	* Did an error occur?
BNE error	* Stop
.	* Here the program continues...
.	

---

## \$4B EXEC

The EXEC function permits loading and chaining programs. If desired, the program loaded can be automatically started. In addition to the function number, the addresses of three strings and a mode word are expected on the stack.

The first address is a pointer to something called an "environment" string, a string which describes the "environment." If the environment is not set, the address of a null string, the address of a zero byte, will suffice.

The second pointer contains a command line for the program being called. A command line is comparable to the line which may be entered from the command mode when you have selected the point "TOS -takes parameters" from the option "Options".

The third pointer points to the filename or pathname of the file. All three strings must be terminated with a zero byte or consist of only a zero byte.

The mode word can be either zero or three. The standard value zero starts the loaded program automatically, while a three loads the program without automatically executing it. In this last case, either the address of the base page or an error message is returned in D0.

---

```

MOVE.L #env,-(SP)      * Environment
MOVE.L #com,-(SP)      * Command line
MOVE.L #fil,-(SP)      * Filename
MOVE.W #0,-(SP)        * Load and start, please
MOVE.W #$4B,-(SP)      * Function number
TRAP    #1
ADD.L   #14,SP          * Here we come to the end of the
.                               * chained program or postloaded module
.
fil:                               * Load sort routine
      .dc.b  'qsort.prg',0
com:                               * Sort the file in ascending order
      .dc.b  'up data.asc',0
env:                               * No environment
      .dc.b  0
.

```

---

## \$4C TERM

TERM \$4C represents the third method, after TERM \$00 and TERM \$31, of ending a program. TERM \$4C automatically makes the memory used by the program available to GEMDOS again. Different from TERM \$00, however, a programmer-defined return value other than zero can be returned to the caller. This allows a short message to be passed back to the calling program.

---

```
MOVE.W #37,-(SP)      * Any 2-byte value
MOVE.W #$4C,-(SP)     * End program
TRAP    #1             *                      now
.                     * We never get here
```

---

## \$ 4E SFIRST

The SFIRST function can be used to check to see if a file with the given name is present in the directory. If a file with the same name is found, the filename, the file attribute, data and time of creation, and the size of the file in bytes is returned. This information is placed in the DTA buffer, whose address is set with the SETDTA function, by GEMDOS.

One feature of this function is that the filename need not be specified in its entirety. Individual characters in the filename can be exchanged for a question mark "?", but entire groups of letters can also be replaced by a "\*". In the extreme form a filename would be reduced to the string "\*. \*". In this case the first file in the directory would satisfy the conditions and the filename would appear in the DTA buffer along with the other information.

In addition to the filename, the SFIRST function must also be given a search attribute. The possible parameters of the search attribute correspond to the attributes which can be specified in CHMOD function:

```
$00 = Normal access, read/write possible
$01 = Normal access, write protected
$02 = Hidden entry (ignored by the ST desktop)
$04 = Hidden system file (ignored like $02)
$08 = Volume label, diskette name
$10 = Subdirectory
$20 = File will be written and closed
```

The following rules apply when searching for files:

- If the attribute word is zero, only normal files are recognized. System files or subdirectories are not recognized.
- System files, hidden files, and subdirectories are found when the corresponding attribute bits are set. Volume labels are not recognized, however.

In order to get the volume label, this option must be expressly set in the attribute word. All other files are then ignored. After the call, D0 contains the value zero if a corresponding file has been found. In this case the 44-byte DTA buffer is constructed as follows:

Bytes	0-20	Reserved for GEMDOS
Byte	21	File attribute
Bytes	22-23	Clock time of file creation
Bytes	24-25	Date of file creation
Bytes	26-29	File size in bytes (long)
Bytes	30-43	Name and extension of the file

If, however, no file is found which corresponds to the specified search string, the error message -33, file not found, is returned.

---

```

MOVE.L #dta,-(SP)      * Set up DTA buffer
MOVE.W #1A,-(SP)      * Function number SETDTA
TRAP   #1
ADDQ.L #6,SP
MOVE.W #attrib,-(SP)   * Attribute value
MOVE.L #filnam,-(SP)   * Name of file to search for
MOVE.W #$4E,-(SP)      * Function number
TRAP   #1
ADDQ.L #8,SP
TST    D0              * File found?
BNE     notfound       * Apparently not
.
.
attrib:
        .dc.b  0        * Search for normal files only
filnam:
        .dc.b  '*.*',0  * Search for the 1st possible file
.
.
dta:
        .ds.b  44       * Space for the DTA buffer

```

---

## **\$4F SNEXT**

The SNEXT function (Search next) can be used to see if there are other files on the disk which match the filename given. To do this, only the function number need be passed; SNEXT does not require any parameters. All of the parameters are set from the SFIRST call.

If the search string is very global, as in the previous example, all of the files on a diskette can be determined and displayed one after the other with SFIRST and SNEXT. This makes it rather easy to display a directory within a program. The SNEXT function is called repeatedly and the contents of D0 are checked afterwards. If D0 contains a value other than zero, either an error occurred, or all of the directory entries have been searched.

## **\$56 RENAME**

A RENAME function is found in almost every disk-oriented operating system in one form or another, since renaming files is required fairly often. Under GEMDOS, files are renamed with the RENAME function, which requires two pointers to file or pathnames. The first pointer points to the new name, with the specification of the pathname of the file if necessary, and the second pointer points to the previous name. A 2-byte parameter is required in addition to the two pointers. We were not able to determine the significance of the additional word parameter. Different values had no (recognizable) effect.

As a return value, D0 contains either zero, meaning that the name was changed correctly, or an error code.

---

```

MOVE.L #newnam,-(SP)    * New filename
MOVE.L #oldname,-(SP)  * File to rename
MOVE.W #0,-(SP)        * Dummy?
MOVE.W #$56,-(SP)      * Function number
TRAP    #1
ADD.L   #12,SP
TST.L   D0              * Test for error
.
.
oldnam:                * Don't forget zero byte at end!
        .dc.b  'oldfile.dat',0
newnam:
        .dc.b  'newname.dat',0
.
.

```

---

## \$57 GSDTOF

If the directory is displayed as text rather than icons on the desktop, the date and time of file creation as well as the size of the file in bytes is shown. The time and date can either be set or read with function \$57. To do this it is necessary that the file be already opened with OPEN or CREATE. The handle number obtained at the opening must be passed to the function. Additional parameters are a word which acts as a flag as to whether the time and data are to be set (0) or read (1), and a pointer to a 4-byte buffer which either contains the result data or will be provided with the required data before the call.

This date buffer contains the time in the first two bytes and the date in the last two. The format of the data is identical to that of the functions for setting/reading the time and date.

**Example 1:**

```
MOVE.W #1,-(SP)      * Read time and date
MOVE.W #handle,-(SP) * File must first be opened
MOVE.L #buff,-(SP)   * 4 byte buffer
MOVE.W #$57,-(SP)    * Function number
TRAP    #1
ADD.L   #10,SP
.
.
handle:
        .ds.b 2
buff:
        .ds.b 4
.
.
```

**Example 2:**

```
MOVE.W #0,-(SP)      * Set time and date
MOVE.W #handle,-(SP) * File must first be opened
MOVE.L #buff,-(SP)   * 4 byte buffer
MOVE.W #$57,-(SP)    * Function number
TRAP    #1
ADD.L   #10,SP
.
.
handle:
        .ds.b 2
buff:
        .ds.b 4
.
.
```

---



### 3.1.1 GEMDOS error codes and their meaning

The GEMDOS functions return a value giving information about whether or not an error occurred during the execution of the function. A value of zero means no error; negative values have the following meanings:

- 32 Invalid function number
- 33 File not found
- 34 Pathname not found
- 35 Too many files open (no more handles left)
- 36 Access not possible
- 37 Invalid handle number
- 39 Not enough memory
- 40 Invalid memory block address
- 46 Invalid drive specification
- 49 No more files

In addition to these error messages, the BIOS error messages may occur. These error messages have numbers -1 to -31 and are described in section 3.3

## 3.2 The BIOS Functions

The software interface between the GEMDOS and the hardware of the computer is the BIOS (Basic Input Output System). The BIOS, as the name suggests, is concerned with the fundamental input/output functions. This includes screen output, keyboard input, printer output, as well as the RS-232 interface and, of course, input/output to the disk.

The BIOS functions are also available to user programs. The TRAP instruction of the 68000 processor is used to call them. Any data required is passed through the stack and the result of the function is returned in the D0 register. The machine language programmer should be aware that the contents of D0-D2 and A0-A2 are changed when calling BIOS functions; the remaining registers remain unchanged.

BIOS function calls are even simpler if you program in C. Here you can use simple function calls with the corresponding parameter lists. The function calls are stored as macros in an include file. In the examples, the definition of the function and its parameters in C will be shown. For assembly language programmers, the use is described in an example.

TRAP #13 is reserved for the BIOS functions.

**0 getmpb***get memory parameter block*

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

**Assembler:**

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

This function fills a 12-byte block whose address is contained in *pointer* with the memory parameter block. This block contains three pointers itself:

long	mfl	Memory free list
long	mal	Memory allocated list
long	rover	Roving pointer

The structures to which each pointer points are constructed as follows:

long	link	Pointer to next block
long	start	Start address of the block
long	length	Length of the block in bytes
long	own	Process descriptor

**Example:**

```
move.l  #buffer, -(sp)  Buffer for MPB
move.w  #0, -(sp)       getmpb
trap    #13             Call BIOS
addq.l  #6, sp          Stack correction
```

We get the values \$48E, 0, and \$48E. The following data are at address \$48E:

link	0	No additional block
start	\$3B900	Start address of the free memory
length	\$3C700	Length of the free memory
own	0	No process descriptor

## 1 bconstat *return input device status*

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

Assembler:

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

This function returns the status of an input device which is defined as follows:

```
Status 0      No characters ready
Status -1     (at least) one character ready
```

The parameter dev specifies the input device:

```
dev    Input device
0      PRT:, Centronics interface
1      AUX:, RS-232 interface
2      CON:, Keyboard and screen
3      MIDI, MIDI interface
4      IKBD, Keyboard port
```

The following table lists the allowed accesses to these devices:

Operation	PRT:	AUX:	CON:	MIDI	IKBD
Input status	no	yes	yes	yes	no
Input	yes	yes	yes	yes	yes
Output status	yes	yes	yes	yes	yes
Output	yes	yes	yes	yes	yes

This example waits until a character from the RS-232 interface is ready.

```
wait move.w #1,-(sp)      RS-232
    move.w #1,-(sp)      bconstat
    trap    #13
    addq.l #4,sp
    tst     d0            character available?
    beq     wait          no, wait
```

**2 conin***read character from device*

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

Assembler:

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

This function fetches a character from an input device. The parameter `dev` has the same meaning as in the previous function. The function does not return until a character is ready.

The character received is in the lowest byte of the result. If the input device was the keyboard (`con, 2`), the key scan code is also returned in the lower byte of the upper word (see description of the keyboard processor).

Example:

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

### 3 bconout

*write character to device*

```
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
```

This function serves to output a character "c" to the output device dev (meaning is the same as for the previous function). The function returns when the character has been outputted.

Example:

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

The example outputs the letter "A" to the printer.

## 4 rwabs

*read and write disk sector*

```
C: long rwabs(rwflag, buffer, number, recno, dev)
    long buffer;
    int rwflag, number, recno, dev;
```

Assembler:

```
move.w dev, -(sp)
move.w recno, -(sp)
move.w number, -(sp)
move.l buffer, -(sp)
move.w rwflag, -(sp)
move.w #4, -(sp)
trap    #13
add.l   #14, sp
```

This function serves to read and write sectors on the disk. The parameters have the following meaning:

rwflag	Meaning
0	Read sector
1	Write sector
2	Read sector, ignore disk change
3	Write sector, ignore disk change

The parameter `buffer` is the address of a buffer into which the data will be read from the disk or from which the data will be written to the disk. The buffer should begin at an even address, or the transfer will run very slowly.

The parameter `number` specifies how many sectors should be read or written during the call. The parameter `recno` specifies which logical sector the process will start with.

The parameter `dev` determines which disk drive will be used:

dev	Drive
0	Drive A
1	Drive B
2	Hard disk

The function returns an error code as the result. If this value is zero, the operation was performed without error. The returned value will be negative if an error occurred. The error code has the following meaning:

- 0 OK, no error
- 1 General error
- 2 Drive not ready
- 3 Unknown command
- 4 CRC error
- 5 Bad request, invalid command
- 6 Seek error, track not found
- 7 Unknown media (invalid boot sector)
- 8 Sector not found
- 9 (No paper)
- 10 Write error
- 11 Read error
- 12 General error
- 13 Diskette write protected
- 14 Diskette was changed
- 15 Unknown device
- 16 Bad sector (during verify)
- 17 Insert diskette (for connected drive)

**Example:**

```
move.w #0,-(sp)      Drive A
move.w #10,-(sp)     Start at logical sector 10
move.w #2,-(sp)      Read 2 sectors
move.l #buffer,-(sp) Buffer address
move.w #0,-(sp)      Read sectors
move.w #4,-(sp)      rwabs
trap    #13
add.l   #14,sp
...
buffer  ds.b  2*512
```



## 5 setexec

*set exception vectors*

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

Assembler:

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

The function `setexec` allows one of the exception vectors of the 68000 processor to be changed. The number of the vector must be passed in `number` and the address of the routine pertaining to it in `vector`. The function returns the old vector as the result. If you just want to read the vector, pass the value -1 as the new address. The 256 processor vectors as well as 8 vectors for GEM, which numbers \$100 to \$107 (address \$400 to \$41C) can be changed with this function.

Example:

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

**6 tickcal***return millisecond per tick*

C: long tickcal()

Assembler:

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

This function returns the number of milliseconds between two system timer calls.

Example:

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

Result: 20 ms

## 7 getbpb

*get BIOS parameter block*

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

Assembler:

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

This function returns a pointer to the BIOS Parameter Block of the drive dev (0=drive A, 1=drive B).

The BPB (BIOS Parameter Block ) is constructed as follows:

int	recsiz	Sector size in bytes
int	clsiz	Cluster size in sectors
int	clsizb	Cluster size in bytes
int	rdlen	Directory length in sectors
int	fsiz	FAT size in sectors
int	fatrec	Sector number of the second FAT
int	datrec	Sector number of the first data cluster
int	numcl	Number of data clusters on the disk
int	bflags	Misc. flags

The function returns the address \$3E3E for drive A and the address \$3E5E for drive B. An address of zero indicates an error.

Example:

move.w #0,-(sp)	Drive A
move.w #7,-(sp)	getbpb
trap   #13	
addq.l #4,sp	

Here are the BPB data for 80 track single and double-sided disk drives:

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***return output device status*

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

**Assembler:**

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

This function tests to see if the output device specified by `dev` is ready to output the next character. `dev` can accept the values which are described in function one. The result of this function is either -1 if the output device is ready, or zero if it must wait.

**Example:**

```
move.w #0,-(sp)      Printer ready?
move.w #8,-(sp)      bcostat
trap   #13
addq.l #4,sp
```

## 9 mediach

*inquire media change*

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

Assembler:

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

This function determined if the disk was changed in the meantime. The parameter `dev`, the drive number (0=drive A, 1=drive B), must be passed to the routine. One of three values can occur as the result:

```
0    Diskette was definitely not changed
1    Diskette may have been changed
2    Diskette was definitely changed
```

Example:

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

**10 drvmap***inquire drive status*

C: long drvmap()

Assembler:

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

This function returns a bit vector which contains the connected drives. The bit number *n* is set if drive *n* is available (0 means A, etc.). Even if only one drive is connected, %11 is still returned, since two logical drives are assumed.

Example:

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

## 11 kbshift

*inquire/change keyboard status*

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

Assembler:

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

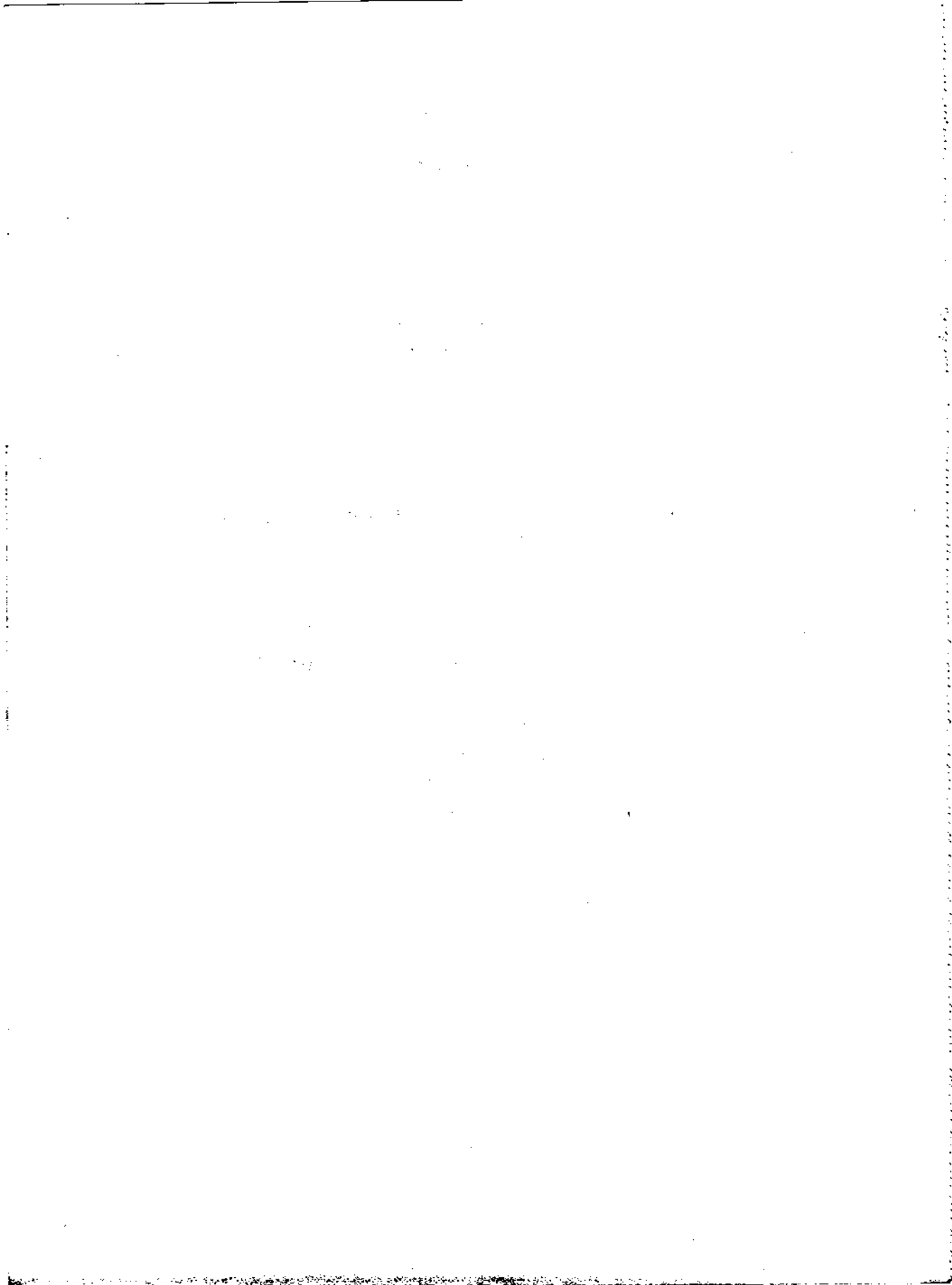
With this function you can change or determine the status of the special keys on the keyboard. If mode is -1, you get the status, a positive value is accepted as the status. The status is a bit vector which is constructed as follows:

Bit	Meaning
0	Right shift key
1	Left shift key
2	Control key
3	ALT key
4	Caps Lock on
5	Right mouse button (CLR/HOME)
6	Left mouse button (INSERT)
7	Unused

Example:

```
move.w #-1,-(sp)      Read shift status
move.w #11,-(sp)      kbshift
trap   #13
addq.l #4,sp
```





### 3.3 The XBIOS

To support the special hardware features of the Atari ST, there are extended BIOS functions, which are called via a TRAP #14 instruction. The functions, like the normal BIOS functions, can be called from assembly language as well as from C. When calling from C, a small TRAP handler in machine language is again necessary, which can look like this:

```
trap14:
    move.l    (sp)+,retsave    Save return address
    trap      #14              Call XBIOS
    move.l    retsave,-(sp)    Restore return address
    rts

    .bss
retsave    ds.l    1           Space for the return address
```

Macro functions can be used in C which allow the extended BIOS functions (eXtended BIOS, XBIOS) to be called by name. The appropriate function number and TRAP call will be created when the macro is expanded.

When working in assembly language, the function number of the XBIOS routine need simply be passed on the stack. The XBIOS has 40 different functions whose significance and use are described on the following pages.

**0 initmous***initialize mouse*

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

Assembler:

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

This XBIOS function initializes the routines for mouse processing. The parameter *vector* is the address of a routine which will be executed following a mouse-report from the keyboard processor. The parameter *type* selects from among the following alternatives:

type	
0	Disable mouse
1	Enable mouse, relative mode
2	Enable mouse, absolute mode
3	unused
4	Enable mouse, keyboard mode

This allows you to select if mouse movements are to be reported and in what manner this will occur.

The parameter *parameter* points to a parameter block, which is constructed as follows:

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

The parameter *topmode* determines the layout of the coordinate system. A 0 means that Y=0 lies in the lower corner, 1 means that Y=0 lies in the upper corner.

The parameter `buttons` is a parameter for the command "set mouse buttons" of the keyboard processor (see description of the IKBD, intelligent keyboard).

The parameters `xparam` and `yparam` are scaling factors for the mouse movement. If you have selected 2 as the `type`, the absolute mode, the parameter block determines four more parameters:

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

These are the X and Y-coordinates of the maximal value which the mouse position can assume, as well as the start value to which the mouse will be set.

Example:

```
move.l #vector,-(sp)      Address of the mouse position
move.l #parameter,-(sp)  Address of the parameter block
move.w #1,-(sp)          Enable relative mouse mode
move.w #0,-(sp)          Init mouse
trap   #14
add.l  #12,sp
...
parameter dc.b .....
...
vector    ...           Mouse interrupt routine
```

**1 ssbrk***save memory space*

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

Assembler:

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

This function reserves memory space. The number of bytes must be passed in *number*. The memory space is prepared at the upper end of memory. The function returns the address of the reserved memory area as the result. This function must be called before initializing the operating system, meaning that it must be called from the boot ROM, before the operating system is loaded.

Example:

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

**2 physbase***return screen RAM base address***C:** long physbase()**Assembler:**

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

This function returns the base of the physical screen RAM. The physical screen RAM is the area of memory which is displayed by the video shifter. The result is a long word.

**Example:**

```
$78000, base address of the screen for 512K RAM
```

### 3 logbase

*set logical screen base*

C: long logbase()

Assembler:

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

The logical screen base is the address which is used for all output functions as the screen base. If the physical and logical screen bases are different, one screen will be displayed while another picture is being constructed in a different area of RAM, which will be displayed later. The result of this function call is again a longword.

Example:

```
$78000, base address of the screen for 512K RAM
```

**4 getrez** *return screen resolution*

C: int getrez()

Assembler:

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

This function call returns the screen resolution:

```
0 := Low resolution, 320*200 pixels, 16 colors
1 := Medium resolution, 640*200 pixels, 4 colors
2 := High resolution, 640*400, pixels, monochrome
```

Example:

```
2, monochrome
```



## 5 setscreen

*set screen parameters*

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

### Assembler:

```
move.w res,-(sp)
move.l physadr,-(sp)
move.l logadr,-(sp)
move.w #5,-(sp)
trap   #14
add.l  #12,sp
```

This function changes the screen parameters which can be read with the previous three functions. If a parameter should not be set, a negative value must be passed. The parameters are set in the next VBL routine so that no disturbances appear on the screen.

### Example:

Set the physical and the logical screen address to \$70000, retain the resolution.

move.w #-1,-(sp)	Retain resolution
move.l #\$70000,-(sp)	Physical base
move.l #\$70000,-(sp)	Logical base
move.w #5,-(sp)	setscreen
trap   #14	
add.l  #12,sp	

**6 setpalette***set color palette*

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

Assembler:

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

A new color palette can be loaded with this function. The parameter `paletteptr` must be a pointer to a table with 16 colors (each a word). The address of the table must be even. The colors will be loaded at the start of the next VBL. Example:

```
move.l #palette, -(sp)    Address of the new color palette
move.w #6, -(sp)         set palette
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 *set color*

```
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
```

This function allows just one color to be changed. The color number (0-15) and the color belonging to it (0-\$777) must be specified. If -1 is given as the color, the color is not set but the previous color is returned.

Example:

```
move.w #$777,-(sp)    Color white
move.w #1,-(sp)       As color number 1
move.w #7,-(sp)
trap    #14
addq.l #6,sp
```

## 8 floprd

*read diskette sector*

```
C: int floprd(buffer, filler, dev, sector, track, side,  
count)
```

```
    long buffer, filler;
```

```
    int dev, sector, track, side, count;
```

### Assembler:

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

This function reads one or more sectors in from the diskette. The parameters have the following meaning:

- count: Specifies how many sectors are to be read. Values between one and nine (number of sectors per track) are possible.
- side: Selects the diskette side, zero for single-sided drives and zero or one for double-sided drives.
- track: Determines the track number (0-79 for 80-track drives or 0-39 for 40-track drives).
- sector: The sector number of the first sector to be read (0-9).
- dev: Determine drive number, 0 for drive A and 1 for drive B.
- filler: Unused long word.
- buffer: Buffer in which the diskette data should be written. The buffer must begin on a word boundary and be large enough for the data to be read (512 bytes times the number of sectors).

The function returns an error code which has the following meaning:

- 0 OK, no error
- 1 General error
- 2 Drive not ready
- 3 Unknown command
- 4 CRC error
- 5 Bad request, invalid command
- 6 Seek error, track not found
- 7 Unknown media (invalid boot sector)
- 8 Sector not found
- 9 (No paper)
- 10 Write error
- 11 Read error
- 12 General error
- 13 Diskette write protected
- 14 Diskette was changed
- 15 Unknown device
- 16 Bad sector (during verify)
- 17 Insert diskette (for connected drive)

Example:

move.w #1,-(sp)	Read a sector
move.w #0,-(sp)	Page zero
move.w #0,-(sp)	Track zero
move.w #1,-(sp)	Sector one
move.w #1,-(sp)	Drive B
clr.l -(sp)	
move.l #buffer,-(sp)	
move.w #8,-(sp)	flopdr
trap #14	
add.l #20,sp	
tst d0	Did error occur?
bmi error	yes
...	
buffer ds.b 512	Buffer for a sector

## 9 flopwr

*write diskette sector*

```
C: int floprd(buffer, filler, dev, sector, track, side,
             count)
    long buffer, filler;
    int  dev, sector, track, side, count;
```

Assembler:

```
move.w count, -(sp)
move.w side, -(sp)
move.w track, -(sp)
move.w sector, -(sp)
move.w dev, -(sp)
clr.l  -(sp)
move.l buffer, -(sp)
move.w #9, -(sp)
trap   #14
add.l  #20, sp
```

One or more sectors can be written to disk with this XBIOS function. The parameters have the same meaning as for the function 8 *floprd*. The function returns an error code which also has the same meaning as for reading sectors. Example:

move.w #3, -(sp)	Write three sectors
move.w #0, -(sp)	Side zero
move.w #7, -(sp)	Track seven
move.w #1, -(sp)	Sector one
move.w #0, -(sp)	Drive A
clr.l  -(sp)	
move.l #buffer, -(sp)	Address of the buffer
move.w #9, -(sp)	flopwr
trap   #14	
add.l  #20, sp	
tst    d0	Did an error occur?
bmi    error	yes
...	
buffer ds.b 3*512	Buffer for three sectors

## 10 flopfmt *format diskette*

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

### Assembler:

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

This routine serves to format a track on the diskette. The parameters have the following meanings:

virgin:	The sectors are formatted with this value. The standard value is \$E5E5. The high nibble of each byte may not contain the value \$F.
magic:	The constant \$87654321 must be used as magic or formatting will be stopped.
interleave:	Determines in which order the sectors on the disk will be written, usually one.
side:	Selects the disk side (0 or 1).
track:	The number of the track to be formatted (0-79).
spt:	Sectors per track, normally 9.
dev:	The drive, 0 for A and 1 for B.

---

filler: Unused long word.

buffer: Buffer for the track data; for 9 sectors per track the buffer must be at least 8K large.

The function returns an error code as its result. The value -16, bad sectors, means that data in some sectors could not be read back correctly. In this case the buffer contains a list of bad sectors (word data, terminated by zero). You can format these again or mark the sectors as bad.

Example:

```

move.w #$E5E5,-(sp)      Initial data
move.l #$87654321,-(sp)  magic
move.w #1,-(sp)          interleave
move.w #0,-(sp)          side 0
move.w #79,-(sp)         track 79
move.w #9,-(sp)          9 sector per track
move.w #0,-(sp)          drive A
clr.l -(sp)
move.w #buffer,-(sp)
move.w #10,-(sp)         flopfmt
trap    #14
add.l   #26,sp
tst     d0
bmi     error

buffer   ds.b   $2000      8K buffer

```

## 11 unused



## 12 midiws

*write string to MIDI interface*

```
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
```

With this function it is possible to output a string to the MIDI interface (MIDI OUT). The parameter `ptr` must point to a string, `count` must contain the number of characters to be sent minus 1.

Example:

```
move.l #string, -(sp)           Address of the string
move.w #stringend-string-1, -(sp) Length
move.w #12, -(sp)              midiws
trap #14
addq.l #8, sp
```

....

```
string    dc.b 'MIDI data'
stringend equ *
```

## 13 mfpint

*initialize MFP format*

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

Assembler:

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

This function initializes an interrupt routine in the MFP. The number of the MFP interrupt is in `number` while `vector` contains the address of the corresponding interrupt routine. The old interrupt vector is overwritten.

Example:

```
move.l #busy, -(sp)      Busy interrupt routine
move.w #0, -(sp)         Vector number 0
move.w #13, -(sp)       mfpint
trap   #14
addq.l #8, sp
....
busy:
```

## 14 iorec

*return record buffer*

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

Assembler:

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

This function fetches a pointer to a buffer data record for an input device. The following input devices can be specified:

dev	Input device
0	RS-232
1	Keyboard
2	MIDI

The buffer record for an input device has the following layout:

long	ibuf	Pointer to an input buffer
int	ibufsize	Size of the input buffer
int	ibufhd	Head index
int	ibuftl	Tail index
int	ibuflow	Low water mark
int	ibufhi	High water mark

The input buffer is a circular buffer; the `head index` specifies the next write position (the buffer is filled by an interrupt routine) and the `tail index` specifies from where the buffer can be read. If the head and tail indices are the same, the buffer is empty. The low and high marks are used in connection with the communications status for the RS-232 (XON/XOFF or RTS/CTS). If the input buffer is filled up to the high water mark, the sender is informed via XON or CTS that the computer cannot receive any more data. When data received by the computer can be processed again, so that the buffer contents sink below the low water mark, the transfer is resumed.

There is an identically-constructed buffer record for the RS-232 output which is located directly behind the input record.

**Example:**

```

move.w #1,-(sp)    Buffer record for keyboard
move.w #14,-(sp)   iorec
trap    #14
addq.l #4,sp

```

...

**Result: \$9F2****The following table contains the data for all devices:**

	RS-232 input	RS-232 output	Keyboard	MIDI
Address	\$9D0	(\$9DE)	\$942	\$A00
Buffer address	\$6D0	\$7D0	\$8D0	\$950
Buffer length	\$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 and tail indices are naturally dependent on the current operating mode. High and low water marks are set at 3/4 and 1/4 of the buffer size. They have significance only for XON/XOFF or RTS/CTS in connection with RS-232.

**15 rsconf***set RS-232 configuration*

```
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
```

This XBIOS function serves to configure the RS-232 interface. The parameters have the following significance:

```
scr:  Synchronous Character Register in the MFP
tsr:  Transmitter Status Register in the MFP
rsr:  Receiver Status Register in the MFP
ucr:  USART Control Register in the MFP
ctrl: Communications parameters
baud: Baud rate
```

See the section on the MFP 68901 for information on the MFP registers. If one of the parameters is -1, the previous value is retained. The handshake mode can be selected with the ctrl parameter:

ctrl	Meaning
0	No handshake, default after power-up
1	XON/XOFF
2	RTS/CTS
3	XON/XOFF and RTS/CTS (not useful)

The baud parameter contains an indicator for the baud rate:

baud	Baud rate
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

Example:

move.w #-1,-(sp)	
move.w #-1,-(sp)	Don't change MFP registers
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***set keyboard table*

```
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
addi.l #14,sp
```

With this function it is possible to create a new keyboard layout. To do this you must pass the address of the new tables which contain the key codes for normal keys (without shift), shifted keys, and keys with caps lock. The function returns the address of the vector table in which the three keyboard table pointers are located. If a table should remain unchanged, -1 must be passed as the address. A keyboard table must be 128 bytes long. It is addressed via the key scan code and returns the ASCII code of the given key.

Example:

```
move.l #-1,-(sp)      Don't change caps lock
move.l #shift,-(sp)   Shift table
move.l #unshift,-(sp) Table without shift
move.w #16,-(sp)
trap   #14
addi.l #14,sp
```

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

**17 random***return random number*

C: long random()

Assembler:

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

This function returns a 24-bit random number. Bits 24-31 are zero. With each call you receive a different result. After turning on the computer a different seed is created.

Example:

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



## 18 protobt *produce boot sector*

```
C: void protobt(buffer, serialno, disktype, execflag)
    long buffer, serialno;
    int  disktype, execflag;
```

Assembler:

```
move.w execflag, -(sp)
move.w disktype, -(sp)
move.l serialno, -(sp)
move.l buffer, -(sp)
move.w #18, -(sp)
trap   #14
add.l  #14, sp
```

This function serves to create a boot sector. A boot sector is located on track 0, sector 1 on side 0 of a diskette and gives the DOS information about the disk type. If the boot sector is executable, it can be used to load the operating system. With this function you can create a new boot sector, for a different disk format or to change an existing boot sector. The parameters:

**execflag:** determines if the boot sector is executable.

```
0 not executable
1 executable
-1 boot sector remains as it was
```

The disk type can assume the following values:

```
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 Disk type remains unchanged
```

The parameter **serialno** is a 24-bit serial number which is written in the boot sector. If the serial number is greater than 24 bits (\$01000000), a random serial number is created (with the above function). A value of -1 means that the serial number will not be changed.

The parameter **buffer** is the address of a 512-byte buffer which contains the boot sector or in which the boot sector will be created.

A boot sector has the following construction:

Address 40 track SS 40 track DS 80 track SS 80 track DS

0- 1	Branch instruction to boot program if executable			
2- 7	'Loader'			
8-10	24-bit serial number			
11-12	BPS	512	512	512
13	SPC	1	2	2
14-15	RES	1	1	1
16	FAT	2	2	2
17-18	DIR	64	112	112
19-20	SEC	360	720	1440
21	MEDIA	252	253	248
22-23	SPF	2	2	5
24-25	SPT	9	9	9
26-27	SIDE	1	2	2
28-29	HID	0	0	0
510-511	CHECKSUM			

The abbreviations have the following meanings:

- EPS: Bytes per sector. The sector size is 512 bytes for all formats
- SPC: Sectors per cluster. The number of sectors which are combined into one block by the DOS, 2 sectors equals 1K.
- RES: Number of reserved sectors at the start of the disk including the boot sector.
- FAT: The number of file allocation tables on the disk.
- DIR: The maximum number of directory entries.
- SEC: The total number of sectors on the disk.
- MEDIA: Media descriptor byte, not used by the ST-BIOS.
- SPF: Number of sectors in each FAT.
- SPT: Number of sectors per track.

SIDE: Number of sides of the diskette.

HID: Number of hidden sectors on the disk.

The boot sector is compatible with MS-DOS 2.x. This is why all 16-bit words are stored in 8086 format (first low byte, then high byte).

If the checksum of the whole boot sector is \$1234, the sector is executable. In this case the boot program is located at address 30. Example:

```
move.w #-1,-(sp)      Don't change executability
move.w #3,-(sp)        80 tracks DS
move.l #-1,-(sp)      Don't change serial number
move.l #buffer,-(sp)
move.w #18,-(sp)      protobt
trap    #14
add.l   #14,sp
```

```
buffer ds.b 512
```

This example program can be used to adapt an existing boot sector for 80 tracks, double sided.

## 19 flopver

*verify diskette sector*

```
C: int flopver(buffer, filler, dev, sector, track, side,
               count)
    long buffer, filler;
    int dev, sector, track, side, count;
```

Assembler:

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

This function serves to verify one or more sectors on the disk. The sectors are read from the disk and compared with the buffer contents in memory. The parameters have the same meaning as for reading and writing sectors. If the sector and buffer contents agree, the result of the function will be zero. If an error occurs, the error number will be returned in D0 that has the following meaning:

- 0 OK, no error
- 1 General error
- 2 Drive not ready
- 3 Unknown command
- 4 CRC error
- 5 Bad request, invalid command
- 6 Seek error, track not found
- 7 Unknown media (invalid boot sector)
- 8 Sector not found
- 9 (No paper)
- 10 Write error
- 11 Read error
- 12 General error
- 13 Diskette write protected
- 14 Diskette was changed
- 15 Unknown device

- 16 Bad sector (during verify)
- 17 Insert diskette (for connected drive)

In the case of an error, the buffer will contain a list of erroneous sectors (16-bit values), terminated by a zero word. If the BIOS function 4 *rwabs* was used to write the sectors and if the variable *fverify* (\$444) is set, the sectors will automatically be verified after they are written.

Example:

```
move.w #1,-(sp)      A sector
move.w #0,-(sp)      Side zero
move.w #39,-(sp)     Track 39
move.w #1,-(sp)      Sector 1
move.w #0,-(sp)      Drive A
clr.l -(sp)
move.l #buffer,-(sp) Buffer address
move.w #19,-(sp)     flopver
trap #14
add.l #16,sp
tst d0               Error?
bmi error
```

**20 scrdmp***output screen dump*

C: void scrdmp()

Assembler:

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

This function outputs a hardcopy of the screen to a connected printer. The previously-set printer parameters ("desktop Printer setup") are used. You can also perform this function by simultaneously pressing the ALT and HELP keys or from the desktop through "Print Screen" from the "Options" menu.

Example:

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

## 21 cursconf

*set cursor configuration*

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

Assembler:

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

This XBIOS function serves to set the cursor function. The parameter `function` can have a value from 0-5, which have the following meanings:

function	Meaning
0	Disable cursor
1	Enable cursor
2	Flash cursor
3	Steady cursor
4	Set cursor flash rate
5	Get cursor flash rate

You can use this function to set whether the cursor is visible, and whether it is flashing or steady. This XBIOS function returns a result only if you fetch the old baud rate. The unit of the flash frequency is dependent on the screen frequency: It is 70 Hz for a monochrome monitor or 50 Hz for a color monitor. You can set a new flash rate with function number 5. You need only use the parameter `rate` if you want to pass a new flash rate.

Example:

```
move.w #20,-(sp)      20/70 seconds
move.w #4,-(sp)       Set flash rate
move.w #21,-(sp)     cursconf
trap    #14
addq.l #6,sp
```

## 22 settime

*set clock time and date*

C: void settime(time)  
long time;

### Assembler:

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

This function is used to set the clock time and date. The time is passed in the lower word of `time` and the date in the upper word. The time and date are coded as follows:

```
bits 0- 4  Seconds in two-second increments
bits 5-10  Minutes
bits 11-15  Hours

bits 16-20  Day 1-31
bits 21-24  Month 1-12
bits 25-31  Year (minus offset 1980)
```

### Example:

```
move.l #%101100110000001000000000000000,-(sp)
move.w #22,-(sp)      settime
trap #14
addq.l #6,sp
```

This call sets the date to the 16th of September, 1985, and the clock time to 8 o'clock.



**23 gettimeofday***return clock time and date*

C: long gettimeofday()

Assembler:

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

This function returns the current date and the clock time in the following format:

```
bits 0- 4  Seconds in two-second increments
bits 5-10  Minutes
bits 11-15  Hours

bits 16-20  Day 1-31
bits 21-24  Month 1-12
bits 25-31  Year (minus offset 1980)
```

Example:

```
move.w #23,-(sp)    gettimeofday
trap   #14
addq.l #2,sp
move.l d0,time      Save time and date
```

**24 bioskeys***restore keyboard table***C:** void bioskeys()**Assembler:**

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

If you have selected a new keyboard layout with the XBIOS function 16, *keytbl*, this function will restore the standard BIOS keyboard layout. You can call this function, for example, before exiting a program of your own which changed the keyboard layout.

**Example:**

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

**25 ikbdws***intelligent keyboard send*

```
C: void ikbdws(number, pointer)
    int  number;
    long pointer;
```

Assembler:

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

This XBIOS function serves to transmit commands to the keyboard processor (intelligent keyboard). The parameter `pointer` is the address of a string to be sent, `number` is the length of a string minus 1.

Example:

move.l #string, -(sp)	Address of the string
move.w #strend-string-1, -(sp)	Length minus 1
move.w #25, -(sp)	ikbdws
trap   #14	
addq.l #8, sp	
...	
string    dc.b  \$80,1	
strend    equ   *	

**26 jdisint***disable interrupts on MFP*

```
C: void jdisint(number)
    int number;
```

Assembler:

```
    move.w number,-(sp)
    move.w #26,-(sp)
    trap   #14
    addq.l #4,sp
```

This function makes it possible to selectively disable interrupts on the MFP 68901. The parameter is the MFP interrupt number (0-15). The significance of the individual interrupts is described in the section on interrupts.

Example:

```
    move.w #10,-(sp)    Disable RS-232 transmitter interrupt
    move.w #26,-(sp)    Disable interrupt
    trap   #14
    addq.l #4,sp
```

**27 jenabint***enable interrupts on MFP*

```
C: void jenabint(number)
    int number;
```

Assembler:

```
    move.w number,-(sp)
    move.w #27,-(sp)
    trap   #14
    addq.l #4,sp
```

This function can be used to re-enable an interrupt on the MFP. The parameter is again the number of the interrupt, 0-15.

Example:

```
    move.w #12,-(sp)      Enable RS-232 receiver interrupt
    move.w #27,-(sp)      Enable interrupt
    trap   #14
    addq.l #4,sp
```

## 28 giaccess

*access GI sound chip*

```
C: char giaccess(data, register)
    char data;
    int  register;
```

### Assembler:

```
    move.w #register, -(sp)
    move.w #data, -(sp)
    move.w #28, -(sp)
    trap   #14
    addq.l #6, sp
```

This function allows access to the registers of the GI sound chip. `register` must contain the register number of the sound chip (0-15). The meaning of the individual registers is given in the hardware description of the sound chip. Bit 7 of the register number determines whether the specified register will be written or read:

```
Bit 7  0: Read
       1: Write
```

When writing, an 8-bit value is passed in `data`; when reading, the function returns the contents of the corresponding register.

### Example:

```
    move.w #$80+3, -(sp)    Write register 3
    move.w #$50, -(sp)      Value to write
    move.w #28, -(sp)
    trap   #14
    addq.l #6, sp
```

## 29 offgibit

*reset Port A GI sound chip*

```
C: void offgibit(bitnumber)
    int bitnumber;
```

Assembler:

```
    move.w #bitnumber, -(sp)
    move.w #29, -(sp)
    trap   #14
    addq.l #4, sp
```

A bit of port A of the sound chip can be selectively set with this function call. Port A is an 8-bit output port in which the individual bits have the following function:

```
Bit 0:  Select disk side 0/side 1
Bit 1:  Select drive A
Bit 2:  Select drive B
Bit 3:  RS-232 RTS (Request To Send)
Bit 4:  RS-232 DTR (Data Terminal Ready)
Bit 5:  Centronics strobe
Bit 6:  General Purpose Output
Bit 7:  unused
```

Example:

```
    move.w #4, -(sp)      DTR bit
    move.w #29, -(sp)     offgibit
    trap   #14
    addq.l #4, sp
```

**30 ongibit***clear Port A of GI sound chip*

```
C: void ongibit(bitnumber)
    int bitnumber;
```

**Assembler:**

```
    move.w #bitnumber, -(sp)
    move.w #30, -(sp)
    trap    #14
    addq.l  #4, sp
```

This function is the counterpart of the previous function. With this it is possible to clear a bit of port A in the sound chip.

**Example:**

```
    move.w #4, -(sp)      DTR bit
    move.w #30, -(sp)     ongibit
    trap    #14
    addq.l  #4, sp
```



## 31 xbtimer

*start MFP timer*

```
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
```

This function allows you to start a timer in the MFP 68901 and assign an interrupt routine to it. `timer` is the number of the timer in the MFP:

```
Timer A : 0
Timer B : 1
Timer C : 2
Timer D : 3
```

The parameters `data` and `control` are the values which are placed in the corresponding control and data registers of the timer. We refer you to the hardware description of the MFP 68901.

The parameter `vector` is the address of the interrupt routine which will be executed when the timer runs out. The four timers in the MFP are already partly used by the operating system:

```
Timer A: Reserved for the end user
Timer B: Horizontal blank counter
Timer C: 200 Hz system timer
Timer D: RS-232 baud rate generator
         (the interrupt vector is free)
```

## Example:

```
move.l #vector,-(sp)    Interrupt routine
move.w data,-(sp)       Data and
move.w control,-(sp)    Control registers
move.w #0,-(sp)         Timer A
move.w #31,-(sp)        xbtimer
trap    #14
add.l   #12,sp
```

**32 dosound***set sound parameters*

```
C: void dosound(pointer)
    long pointer;
```

Assembler:

```
move.l pointer, -(sp)
move.w #32, -(sp)
trap   #14
addq.l #6, sp
```

This function allows for easy sound processing. The parameter `pointer` must point to a string of sound commands. The following commands can be used:

**Commands: \$00-\$0F**

These commands are interpreted as register numbers of the sound chip. A byte following this is loaded into the corresponding register.

**Command \$80**

An argument follows this command which will be loaded into a temporary register.

**Command \$81**

Three arguments must follow this command. The first argument is the number of the register in the sound chip in which the contents of the temporary register will be loaded. The second argument is a two's-complement value which will be added to the temporary register. The third argument contains an end criterium. The end is reached when the content of the temporary register is equal to the end criterium.

**Commands \$82-\$FF**

One argument follows each of these commands. If this argument is zero, the sound processing is halted. Otherwise this argument specifies the number of timer ticks (20ms, 50Hz) until the next sound processing.

**Example:**

```
    move.l #pointer,-(sp)    Pointer to sound command
    move.w #32,-(sp)        dosound
    trap    #14
    addq.l #6,sp
    ....
pointer    dc.b 0,10,1,50,...
```

**33 setprt***set printer configuration*

```
C: void setptr(config)
    int config;
```

Assembler:

```
move.w config,-(sp)
move.w #33,-(sp)
trap    #14
addq.l  #4,sp
```

This function allows the printer configuration to be read or changed. If `config` contains the value -1, the current value is returned, otherwise the value is accepted as the new printer configuration. The printer configuration is a bit vector with the following meaning:

Bit number	0	1
0	matrix printer	daisy-wheel
1	monochrome printer	color printer
2	Atari printer	Epson printer
3	Test mode	Quality mode
4	Centronics port	RS-232 port
5	Continuous paper	Single-sheet
6-14	reserved	
15	always 0	

Example:

```
move.w #%000100,-(sp)    Epson printer
move.w #33,-(sp)         setprt
trap    #14
addq.l  #4,sp
```

### 34 kbdvbase

*return keyboard vector table*

C: long kbdvbase()

Assembler:

```
move.w #34, -(sp)
trap   #14
addq.l #2, sp
```

This XBIOS function returns a pointer to a vector table which contains the address of routines which process the data from the keyboard processor. The table is constructed as follows:

long	midivec	MIDI input
long	vkbderr	Keyboard error
long	vmiderr	MIDI error
long	statvec	IKBD status
long	mousevec	Mouse routines
long	clockvec	Clock time routine
long	joyvec	Joystick routines

The parameter `midivec` points to a routine which writes data received from the MIDI input (byte in D0) to the MIDI buffer.

The parameters `vkbderr` and `vmiderr` are called when an overflow is signaled by the keyboard or MIDI ACIA.

The remaining four routines `statvec`, `mousevec`, `clockvec`, and `joyvec` process the corresponding data packages which come from the keyboard ACIA. A pointer to the packaged received is passed to these routines in D0. The mouse vector is used by GEM. If you want to use your own routine, you must terminate it with RTS and it may not require more than one millisecond of processing time.

Example:

```
move.w #34, -(sp)    kbdvbase
trap   #14
addq.l #2, sp
```

We get \$A0E as the result. The vector field contains the following values:

A0E	midivec	\$79C6
A12	vkbderr	\$759C
A16	vmiderr	\$759C
A1A	statvec	\$7034
A1E	mousevec	\$15296
A22	clockvec	\$6A46
A26	joyvec	\$7034
A2A	MIDI	\$7556
A2E	keyboard	\$7568

**35 kbrate***set keyboard repeat rate*

```
C: int kbrate(delay, repeat)
    int delay, repeat;
```

**Assembler:**

```
move.w repeat,-(sp)
move.w delay,-(sp)
move.w #35,-(sp)
trap    #14
addq.l #6,sp
```

The keyboard repeat can be controlled with this function. The parameter *delay* specifies the delay time after a key is pressed before the key will automatically be repeated. The parameter *repeat* determines the time span after which the key will be repeated again. These values can be changed from the desktop by means of the two slide controllers on the control panel. The times are based on the 50 Hz system clock. If -1 is specified for one of the parameters, the corresponding value is not set. The function returns the previous values as the result; bits 0-7 contain the *repeat* value and bits 8-15 the value of *delay*.

**Example:**

```
move.w #-1,-(sp)      Read old values
move.w #-1,-(sp)
move.w #35,-(sp)      kbrate
trap    #14
addq.l #6,sp
```

**Result:** D0 = \$0B03



**36 prtblk***output block to printer*

```
C: void prtblk(parameter)
    long parameter;
```

Assembler:

```
move.l parameter, -(sp)
move.w #36, -(sp)
trap   #14
addq.l #6, sp
```

This function resembles the function *scrdmp*(20) and is used by it. The function expects a parameter list, however, whose address is passed to it. This list is constructed as follows:

long	blkprt	Address of the screen RAM
int	offset	
int	width	Screen width
int	height	Screen height
int	left	
int	right	
int	scrres	Screen resolution (0, 1, or 2)
int	dstres	Printer resolution (0 or 1)
long	colpal	Address of the color palette
int	type	Printer type (0-3)
int	port	Printer port (0=Centronics, 1=RS232)
long	masks	Pointer to half-tone mask

Assembler:

```
move.l #parameter, -(sp)    Address of the parameter block
move.w #36, -(sp)           prtblk
trap   #14
addq.l #6, sp
...
parameter dc.l ...
```

**37 wvbl***wait for video*

C: void wvbl()

Assembler:

```
move.w #36,-(sp)
trap   #14
addq.l #2,sp
```

This function waits for the next picture return. It can be used to synchronize graphic outputs with the beam return, for example.

Example:

```
move.w #36,-(sp)    wait for wvbl
trap   #14
addq.l #2,sp
```

### 38 supexec

*set supervisor execution*

```
C: void supexec(address)
    long address;
```

Assembler:

```
    move.l address, -(sp)
    move.w #38, -(sp)
    trap   #14
    addq.l #6, sp
```

If a routine is to be executed in the supervisor mode of the 68000 processor, you can accomplish this with this function. Simply pass the address of the routine to the function. Example:

```
    move.l #address, -(sp)
    move.w #38, -(sp)
    trap   #14
    addq.l #6, sp
    ...
address    move.l $400,00
    ...
```

**39 puntaes***disable AES*

C: void puntaes()

Assembler:

```
move.w #39,-(sp)
trap   #14
addq.l #2,sp
```

The AES can be disabled with this function, provided it is not in ROM.

Example:

```
move.w #39,-(sp)
trap   #14
addq.l #2,sp
```

### 3.4 The Graphics

Next to the high processing speed and the large memory available, the graphics capability is certainly the most fascinating aspect of the ST. With the standard monochrome monitor and the resolution of 640x400 points, it creates a whole new price/performance class for itself. But also in the color resolution the ST can display 16 colors with 320x200 screen points.

In this chapter we want to explain how the graphics are organized and how you can create fast and effective graphics without using the GEM graphics package, which is rather complicated for beginners. The ST offers the programmer (assembler and C) very useful routines, with whose help graphics programming isn't quite child's play, but they can take away a good deal of the programming work. Unfortunately, some of these functions are so comprehensive that a detailed description would exceed the scope of this book. We have therefore had to limit ourselves to the simpler, but no less interesting functions.

These graphics routines are called in a very elegant manner. The software developers have made use of the fact that there are two groups of opcodes in the 68000 which the 68000 does not "understand" and which generate a trap, or software interrupt, when they are encountered in a program. These are the two groups of opcodes which begin with \$Axxx and \$Fxxx. In the ST, the \$Axxx opcode trap is used in order to access the graphics routines. The trap handler, the program called by the trap, checks the lowest byte of the "command" to see what value it has. Values between zero and \$E are permissible here. This gives a total of 14 graphics routines, which should first be presented in an overview. Later we will talk about the actual commands in detail.

- \$A000 Determine address of required variable range
- \$A001 Set point on the screen
- \$A002 Determine color of a screen point
- \$A003 Draw a line on the screen
- \$A004 Draw a horizontal line (very fast!)
- \$A005 Fill rectangle with color
- \$A006 Fill polygon line by line
- \$A007 Bit block transfer
- \$A008 Text block transfer
- \$A009 Enable mouse cursor
- \$A00A Disable mouse cursor

\$A00B Change mouse cursor form  
\$A00C Clear sprite  
\$A00D Enable sprite  
\$A00E Copy raster form

These routines are the ground work for the hardware-dependent part of GEM. All GEM graphic and text output is performed by the routines of the \$Axxx opcodes. The set of A-opcodes are very useful in games. In games windows are needed only in the rarest cases. Another important point is the speed of the A-instructions. Using the graphic routines directly is clearly faster than if the output is handled by GEM. Before we describe the individual commands in detail, we will take a brief look at the construction of graphics in the various graphic modes of the ST.

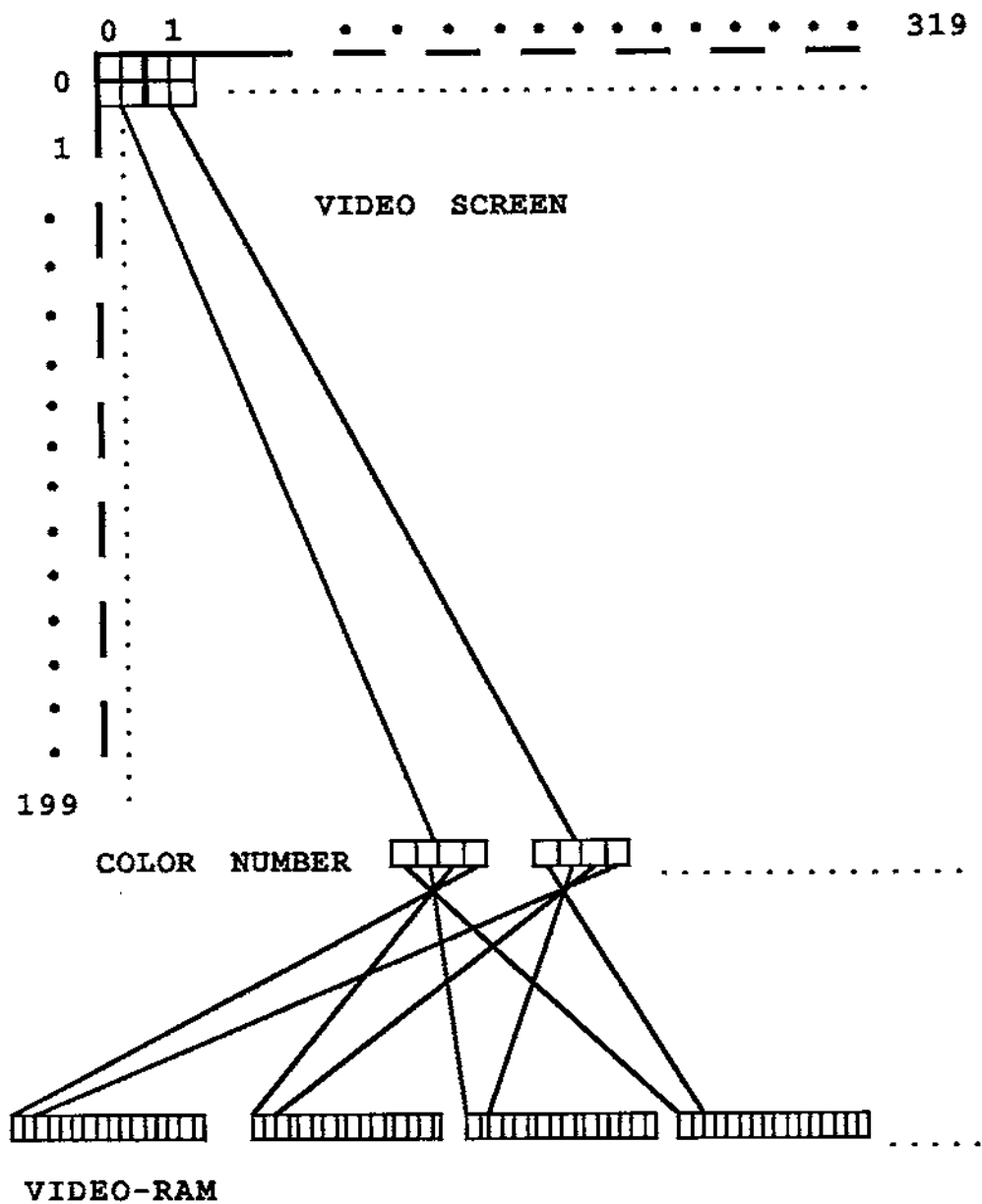
Immediately after turning the ST on, an area of 32K bytes is initialized at the upper memory border as the video RAM. In normal operation this results in addresses \$78000 to \$7FFFF acting as the screen RAM. This video RAM can be viewed as a window in the ST. We will start with the simplest mode, the 640x400 mode. In this case each 80 bytes, or better, each 40 words forms one screen line. The word with the lowest address is displayed on the left edge of the screen, the additional words are displayed in order from left to right. Within a word, the highest-order bit lies at the left and the lowest-order bit at the right.

With this data, any point on the screen can be easily controlled or read. For example, to set the first screen point, the value \$8000 must be written into memory location \$78000. Therefore you might store \$8000 into memory location \$78000. But this isn't recommended.

You might recall that the screen RAM in the ST can be moved quite easily. Then the absolute address of \$78000 is no longer correct, of course. For this reason, it is usually more advantageous to set the the point with the "A" function \$A001. Function \$A001 assumes an X-Y coordinate system with origin in the upper left-hand corner, and determines the position of the video RAM itself in order to set the point at the proper screen location.

In this resolution mode, each screen point is represented by a bit. If the bit is set, the point appears dark, or bright if the the inverse display mode is selected in color palette register 0. The screen consists of only one bit plane. Different colors cannot be represented with just one plane, however. This is why when the resolution increases in the color modes, the number of displayable colors decreases.

Figure 3.4-1 LO-RES-MODE (0)



Four colors possible in the 640x200 resolution mode. In this mode, two contiguous memory words form a single logical entity. The color of a point is determined by the value of the two corresponding bits in the two words. If both bits are zero, the background color results. Therefore two sequential words are used together for pixel representation. For the colors, however, all odd words belong to a plane. The second plane is made up of the even words. In this mode, there are two planes available.

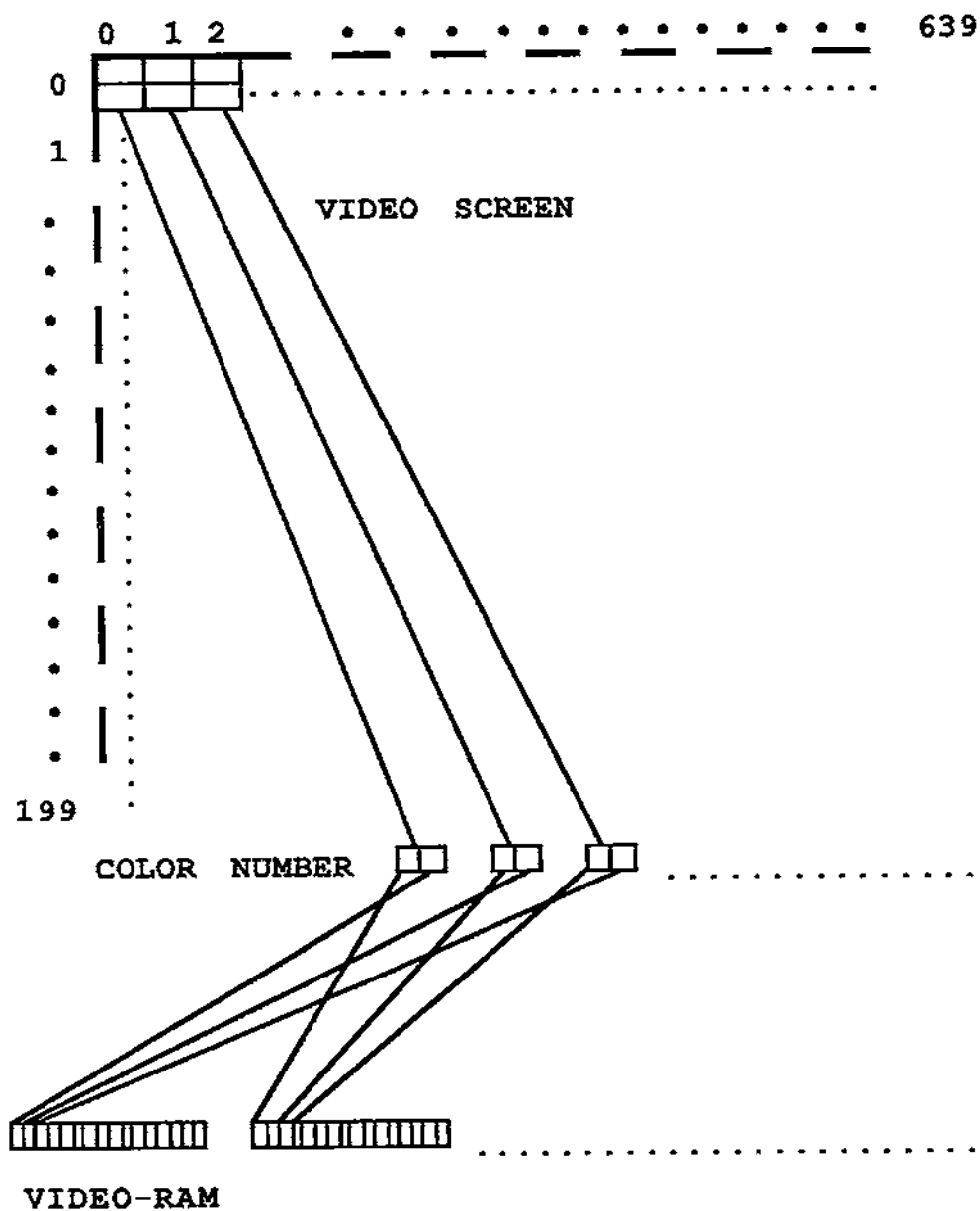
Things become quite colorful in the mode with "only" 320x200 points. In this operating mode, 4 contiguous memory words form one entity which determines the color of the 16 pixels. To stick to the example we used before: in order to set the point in the upper left-hand corner, the topmost bits of words \$78000, \$78002, \$78004, and \$78006 must be manipulated. The desired color results from the bit pattern in the words. It naturally requires some computer time to set a point in the desired color, independent of the mode. All of this work is handled by the \$A001 routine, however. This routine sets all of the pertaining bits for the desired color in the current resolution. Naturally, all four planes are present in this mode. The first plane, keeping to our example, made up of the words at address \$7F000, \$7F008, \$7F010, ..., and the other planes are composed of the other addresses correspondingly.

Another point to be clarified concerns the fonts or character sets. Since the ST does not have a text mode, only a graphics mode, the text output is created in high-resolution graphics. There are three different fonts built into the ST. You can load additional fonts from disk. Each font has a header which contains important information about the displayable characters. Since the important data are contained in the font header, there are unusually few limits for display. The characters can be arbitrarily high or wide. The age of the 8x8 matrix for character output is over. Genuine proportional type on the screen (!) is even possible.

The three built-in fonts use relatively few of the many possibilities which GEM allows for character generation. All three fonts are mono-spaced fonts, meaning they have a fixed defined size in pixels and a defined pitch. The smallest font has a matrix of 6x6. With a resolution of 640x400 points, 66 lines of 106 characters each can be displayed. This font is only accessible for output under GEM, not for output under TOS, and is used in the output of the directory in the icon form, for example. The next-largest type is composed of 8x8 points. This type is used when a color monitor is connected to the ST, while the third and largest font is used for the normal black-and-white mode. This font uses a matrix of 8x16 points.



Figure 3.4-2 MEDIUM-RES-MODE (1)



The exact layout of the font header is found under command \$A008, which represents a very versatile text output which goes far beyond what is possible with the routine of the BIOS and GEMDOS.

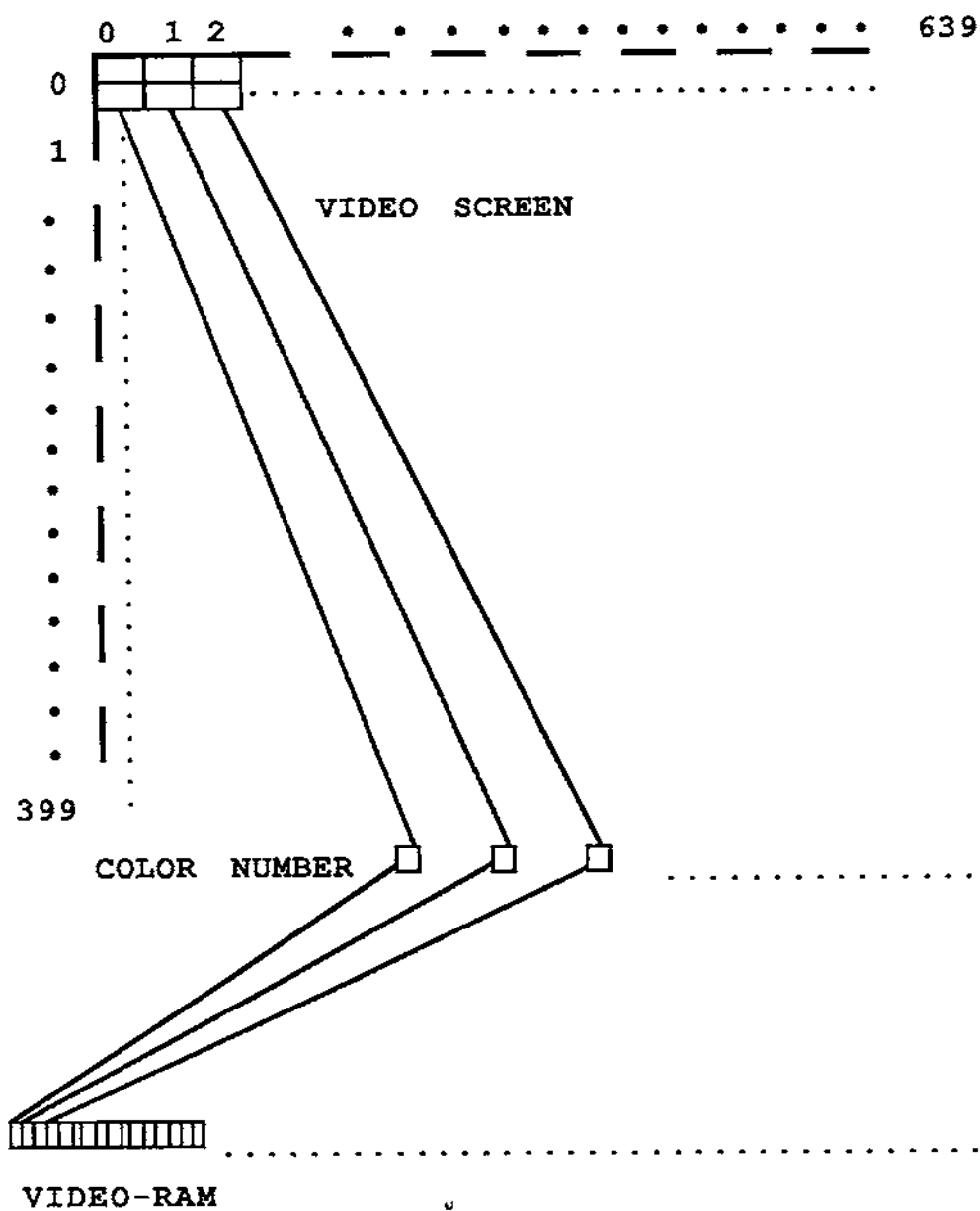
Finally, we must clarify some of the terms which will come up often in the following descriptions, whose meaning may not be so clear. These are the terms CONTRL array, INTIN array, INTOUT array, PTSIN array and PTSOUT array. These arrays are mainly used by GEM to pass parameters to individual GEM functions or to store results from these functions. But line-A functions use parts of these arrays to pass parameters also. The arrays are defined in memory as data areas, whereby each element in the array consists of 2 bytes.

For GEM functions, the CONTRL array always contains the number desired in the first element (CONTRL(0)). This parameter is not used by the line-A commands, however. CONTRL(1) contains the number of XY coordinates required for the function. These coordinates must be placed in the PTSIN array before the call. The element CONTRL(2) is not supplied before the call. After the call it contains the number of XY coordinates in the PTSOUT array. CONTRL(3) specifies how many parameters will be passed to the function in the INTIN array, while CONTRL(4) contains the number of parameters in the INTOUT array after the call. The additional parameters of the CONTRL array are not relevant for users of the line A.

Unfortunately, not all of the parameters for the A opcodes can be in these arrays. For this reason there is another memory area which used as a variable area for (almost) all graphic outputs. The function and use of these over 50 variables is found in a table at the end of this chapter. Important variables are also explained in conjunction with the functions which require them.

By the way, you should be aware that registers D0 to D2 and A0 to A2 are changed by calling the functions. Important values contained in these registers should be saved before a call.

Figure 3.4-3 HI-RES-MODE (2)



## **\$A000 Initialize**

Initialize is really the wrong expression for this function. After the call, the addresses of the more important data areas are returned in registers D0 and A0 to A2. This function does not require input parameters.

The program is informed of the starting address of the line-A variables in D0 and A0. After the call, A1 points to a table with three addresses. These three addresses are the starting address of the three system font headers. Register A2 points to a table with the starting addresses of the 15 line-A routines.

This opcode destroys (at least) the contents of registers D0 to D2 and A0 to A2. Important values should be saved before the call.

## **\$A001 PUT PIXEL**

This opcode sets a point at the coordinates specified by the coordinates in `PTSIN(0)` and `PTSIN(1)`. The color is passed in `INTIN(0)`. `PTSIN(0)` contains X-coordinate, `PTSIN(1)` the Y-coordinate.

The coordinate system used has its origin in the upper left corner. The possible range of the X and Y coordinates is naturally set according to the graphic mode enabled. Overflows in the X range are not handled as errors. Instead, the Y coordinate is simply incremented by the appropriate amount. No output is made if the Y range is exceeded.

The color in `INTIN(0)` is dependent on the mode used. When driving the monochrome monitor, only bit zero of the value of `INTIN(0)` is evaluated.

## **\$A002 GET PIXEL**

The color of a pixel can be determined with this opcode. As with \$A001, the XY coordinates are passed in `PTSIN(0)` and `PTSIN(1)`; the color value is returned in the D0 register.

## \$A003 LINE

With the LINE opcode a line can be drawn between the points with coordinates x1,y1 and x2,y2. The parameters for this function are not passed via the parameter arrays, but must be transferred to the line-A variables before the call. The variables used are:

```

_X1      = x1 coordinate
_Y1      = y1 coordinate
_X2      = x2 coordinate
_Y2      = y1 coordinate
_FG_BP_1 = Plane 1 (all three modes)
_FG_BP_2 = Plane 2 (640x200, 320x200)
_FG_BP_3 = Plane 3 (only 320x200)
_FG_BP_4 = Plane 4 (only 320x200)
_LN_MASK = Bit pattern of the line
           For example: $FFFF = filled
                       $CCCC = broken
_WRT_MOD = Determines the write mode
_LSTLIN  = This variable should be set to -1 ($FFFF)

```

One point to be noted for some applications is the fact that when drawing a line, the highest bit of the line bit pattern is always set on the left screen edge. The line is always drawn from left to right and from top to bottom, not from x1,y1 to x2,y2.

Range overflows are handled as for PUT PIXEL. If an attempt is made to draw a line from 0,0 to 650,50, a line is actually drawn from, 0,0 to 639,48. The "remainder" results in an additional line from 0,49 to 10,50.

A total of four different write modes, with values 0 to 3, are available for drawing lines. With write mode zero, the original bit pattern "under" the line is erased and the bit pattern determined by \_LN\_MASK is put in its place (replace mode). In the transparent mode (\_WRT\_MOD=1), the background, the old bit pattern, is ORed with the new line pattern so only additional points are set. In the XOR mode (\_WRT\_MOD=2), the background and the line pattern are exclusive-ored. The last mode (\_WRT\_MOD=3) is the so-called "inverse transparent mode." As in the transparent mode, it involves an OR combination of the foreground and background data, in which the foreground data, the bit pattern determined by \_LN\_MASK, are inverted before the OR operation.

## \$A004 HORIZONTAL LINE

This function draws a line from x1,y1 to x2,y1. Drawing a horizontal line is significantly faster than when a line must be drawn diagonally. Diagonal lines are also created with this function, in which the line is divided into multiple horizontal lines segments. The parameters are entered directly into the required variables.

```
_X1      = x1 coordinate
_Y1      = y1 coordinate
_X2      = x2 coordinate
_FG_BP_1 = Plane 1 (all three modes)
_FG_BP_2 = Plane 2 (640x200, 320x200)
_FG_BP_3 = Plane 3 (only 320x200)
_FG_BP_4 = Plane 4 (only 320x200)
_WRT_MOD = Determines the write mode
_patptr  = Pointer to the line pattern to use
_patmsk  = "Mask" for the line pattern
```

The valid values in `_WRT_MOD` also lie between 0 and 3 for this call. The contents of the variable `_patptr` is the address at which the desired line pattern or fill pattern is located. The H-line function is very well-suited to creating filled surfaces. The variable `_patmsk` plays an important role in this. The number of 16-bit values at the address in `_patptr` is dependent on the its value. If, for example, `_patmsk` contains the value 5, six 16-bit values should be located at the address in `_patptr` as the line pattern. If a horizontal line with the Y-coordinate value zero is to be drawn, the first bit pattern is taken as the line pattern. The second word is taken as the pattern for a line drawn at Y-coordinate 1, and so on. The pattern for a line with Y-coordinate 6 is again determined by the first value in the bit table. In this manner, very complex fill patterns can be created with relatively little effort.

## \$A005 FILLED RECTANGLE

The opcode \$A005 represents an extension, or more exactly a special use, of opcode \$A004. It is used to create filled rectangles. The essential parameters are the coordinates of the upper left and lower right corners of the rectangle.

```

_X1      = x1 coordinate, left upper
_Y1      = y1 coordinate
_X2      = x2 coordinate, right lower
_Y2      = y2 coordinate
_FG_BP_1 = Plane 1 (all three modes)
_FG_BP_2 = Plane 2 (640x200, 320x200)
_FG_BP_3 = Plane 3 (only 320x200)
_FG_BP_3 = Plane 4 (only 320x200)
_WRT_MOD = Determines the write mode
_patptr  = Pointer to the fill pattern used
_patmsk  = "Mask" for the fill pattern
_CLIP    = Clipping flag
_XMN_CLIP = X minimum for clipping
_XMX_CLIP = X maximum for clipping
_YMN_CLIP = Y minimum for clipping
_YMX_CLIP = Y maximum for clipping

```

We have already explained all of the variables except the "clipping" variables. What is clipping? Clipping creates extracts or clippings of the total picture. If the clipping flag is set to one (or any value not equal to zero), the rectangle, drawn by \$A005, is displayed only in the area defined by the clipping-area variables. An example may explain this behavior better: The values 100,100 and 200,200 are specified as the coordinates. The clip flag is 1 and the clip variables contain the values 150,150 for `XMN_CLIP` and `YMN_CLIP` as well as 300,300 for `XMX_CLIP` and `YMX_CLIP`. The value \$FFFF will be chosen as the fill value for all of the lines. With these values, the rectangle will have the coordinate 150,150 as the upper left corner and 200,200 as the lower right. The "missing" area is not drawn because of the clip specifications. Clearing the clip flag draws the rectangle in the originally desired size.

## \$A006 FILLED POLYGON

\$A006 is also an extension of \$A004. Arbitrary surfaces can be filled with a pattern with this function. The entire surface is not filled with the call: just one raster line is filled, a horizontal line with a width of one point. The result is that there are significantly more options for influencing the fill pattern.

The necessary variables are:

- PTSIN = Array with the XY coordinates
- CONTRL(1) = Number of coordinate pairs
- \_Y1 = y1 coordinate
- \_FG\_BP\_1 = Plane 1 (all three modes)
- \_FG\_BP\_2 = Plane 2 (640x200, 320x200)
- \_FG\_BP\_3 = Plane 3 (only 320x200)
- \_FG\_BP\_4 = Plane 4 (only 320x200)
- \_WRT\_MOD = Determines the write mode
- \_patptr = Pointer to the fill pattern used
- \_patmsk = "Mask" for the fill pattern
- \_CLIP = Clipping flag
- \_XMN\_CLIP = X minimum for clipping
- \_XMX\_CLIP = X maximum for clipping
- \_YMN\_CLIP = Y minimum for clipping
- \_YMX\_CLIP = Y maximum for clipping

Basically, all of the parameters here are to be set exactly as they might be for a call to \$A005. Only the first three coordinates are different. The XY coordinates are stored in the PTSIN array. It is important you specify the start coordinate again as the last coordinate as well. In order to fill a triangle, you must, for example, enter the coordinates (320,100), (120,300), (520,300), and (320,100). The number of effective coordinate pairs, three in our example, must be placed in CONTRL(1), the second element of the array. With a call to the \$A006 function you must also specify the Y-coordinate of the line to be drawn. Naturally you can fill all Y-coordinates from 0 to 399 (0 to 199 in the color modes) in order. But it is faster to find the largest and smallest of the XY values and call the function with only these as the range.



## \$A007 BITBLT

The bit block transfer is used by the text block transfer, \$A008, and copy raster form, \$A00E. Register A6 must contain a pointer to a parameter table. Unfortunately, the construction of this parameter table could not be determined definitively. Our attempts led to classic system crashes about 70% of the time. For this reason, we cannot say much about the function.

## \$A008 TEXTBLT

A character from any desired text font can be printed at any graphic position with the TEXT BLock Transfer function. In addition, the form of the character can be changed. The character can be displayed in italics, boldface, outlines, enlarged, or rotated. These things cannot be achieved with the "normal" character outputs via the BIOS or GEMDOS. But to do this, a large number of parameters must be set and controlled. A rather complicated program must be written in order to output text with this function. If the additional options are not absolutely necessary, it is advisable not to use this function. But please decide for yourself.

Before we produce a character on the screen, we must first concern ourselves with the organization of the fonts. We must take an especially close look at the font header because the font is describe in detail by the information contained in it.

Basically, a font consists of four sets of data: font header, font data, character offset table, and horizontal offset table. The font header contains general data about the font, such as its name and size, the number of characters it contains, and various other aspects. This information takes up a total of 88 bytes. The font data contains the bit pattern of the existing, displayable characters. These data are organized so as to save as much space as possible.

In order to be able to better describe the organization, we will imagine a font with only two characters, such as "A" and "B". These characters are to be displayed in a 9x9 matrix. The font data are now in memory so that the bit pattern of the top scan line of the "A" is stored starting at a word boundary.

Since our font is 9 pixels = 9 bits wide, one byte is completely used, but only the top bit of the following byte. 7 bits must be wasted if the top scan line of the "B" is also to begin on a word boundary. This is not so, however, and the first scan line of the "B" starts with bit 6 of the second

byte of the font data. Only the data of the second and further scan lines always start on a word boundary. In this manner, almost no bits are wasted in the font. Only the start of the scan lines of the first character actually begin on a word boundary; all other scan lines can begin at any bit position.

Because of this space-saving storage, the position of each character within the font must be calculated. The calculation of the scan-line positions is possible through the character offset table. This table contains one entry for each displayable character. For our example, such a table would contain the entries \$0000, \$0009, \$0012. Through the direction of this table, it is possible to create true proportional type on the screen since the width of each character can be calculated. One subtracts the entry of the character to be displayed from the entry of the next character. The last entry is present so that the width of the last character can also be determined, although it is not assigned to a character.

In addition to the character offset table there is the horizontal offset table. This table is not used by most of the fonts, however. The fonts present in the ST do not use all the possibilities of this table either. If this table were present, it would contain a positive or negative offset value for each character, in order to shift the character to the right or left during output.

At the end of the description of the font construction are the meanings of the variables in the font header.

Bytes 0-1 : Font identifier. A number which describes the font. 1=system font  
Bytes 2-3 : Font size in points (point is a measure used in type-setting).  
Bytes 4-35 : The name of the font as an ASCII string.  
Bytes 36-37 : The lowest ASCII value of the displayable characters.  
Bytes 38-39 : The highest ASCII value of the displayable characters.  
Bytes 40-49 : Relative distances of top, ascent, half, descent, and bottom line from the base line.  
Bytes 50-51 : Width of the broadest character in the font.  
Bytes 52-53 : Width of the broadest character cell. The cell is always at least one pixel wider than the actual character so that two characters next to each other are separated from each other.  
Bytes 54-55 : Linker offset.

Bytes 56-57 : Right offset. The two offset values are only used for displaying the font in italics (skewing).

Bytes 58-59 : Thickening. If a character is to be displayed in boldface, the value of this variable is used.

Bytes 60-61 : Underline. Contains the height of the underline in pixels.

Bytes 62-63 : Lightening mask. "Light" characters are found on the desktop when an option on a pull-down menu is not available. This light grey character consists of masking the bits with the lightening mask. Usually the value is \$5555.

Bytes 64-65 : Skewing mask. As before, only for displaying characters in italics.

Bytes 66-67 : Flag. Bit 0 is set if the font is a system font.  
Bit 1 must be set if the horizontal offset table is present.  
Bit 2 is the so-called byte-swap flag. If it is set, the bytes in memory are in 68000 format (low byte-high byte). A cleared swap flag signals that the data is in INTEL format, reversed in memory. With this bit the fonts from the IBM version of GEM can be used on the ST and vice versa.  
Bit 3 is set if the width of all characters in the font is equal.

Bytes 68-71 : Pointer to the horizontal offset table or zero.

Bytes 72-75 : Pointer to the character offset table.

Bytes 76-79 : Pointer to the font data.

Bytes 80-81 : Form width. This variable contains the sum of widths of all the characters. The value represents the length of the scan lines of all of the characters and thereby the start of the next line.

Bytes 82-83 : Form height. This variable contains the number of scan lines for this font.

Bytes 84-87 : Contain a pointer to the next font.

After so much talk, we should now list the parameters which must be noted or prepared for the \$A008 opcode.

<code>_WRT_MODE</code>	= Write mode
<code>_TEXT_FG</code>	= Text foreground color
<code>_TEXT_BG</code>	= Text background color
<code>_FBASE</code>	= Pointer to the start of the font data
<code>_FWIDTH</code>	= Width of the font
<code>_SOURCEX</code>	= X-coordinate of the char in the font
<code>_SOURCEY</code>	= Y-coordinate of the char in the font
<code>_DESTX</code>	= X-coordinate of the char on the screen
<code>_DESTY</code>	= Y-coordinate of the char on the screen
<code>_DELX</code>	= Width of the character in pixels
<code>_DELY</code>	= Height of the character in pixels
<code>_STYLE</code>	= Bit-wise coded flag for special effects
<code>_LITEMASK</code>	= Bit pattern used for "lightening"
<code>_SKEWMASK</code>	= Bit pattern used for skewing
<code>_WEIGHT</code>	= Factor for character enlargement
<code>_R_OFF</code>	= Right offset of the char for skewing
<code>_L_OFF</code>	= Left offset of the char for skewing
<code>_SCALE</code>	= Flag for scaling
<code>_XACC_DDA</code>	= Accumulator for scaling
<code>_DDA_INC</code>	= Scaling factor
<code>_T_SCLSTS</code>	= Scaling direction flag
<code>_CHUP</code>	= Character rotation vector
<code>_MONO_STATUS</code>	= Flag for monospaced type
<code>_scrtp</code>	= Pointer to buffer for effects
<code>_scrtp2</code>	= Offset scaling buffer in <code>_scrtp</code>

The five clip variables are also evaluated.

As you can see, an enormous number of variables are evaluated for the output of graphic text. Here we can go into only the essential (and those we explored) variables.

The write mode allows the output of characters in the four known modes, replace, OR, XOR, and inverse OR. There are 16 other modes available whose effects are not yet known. The variable `_TEXT_FG` is in connection with first four write modes. They form the foreground color used for display. The background color `_TEXT_BG` plays a role only with the 16 additional modes. It is clear that the additional modes are relevant only in connection with a color screen.

The variables `_FBASE` and `_FWIDTH` are set according to the desired font. You can find the start of the font data from the header of the desired font (bytes 76-79 in the header). `_FWIDTH` must be loaded with the contents of the bytes 80 and 81 of the header.

The parameter `_SOURCEX` determines which character you output. It should contain the ASCII value of the desired character.

The parameter `_SOURCEY` is usually zero because the character is to be generated from the top to the bottom scan line.

The parameter `_DELX` can be calculated as the width of the character in which the entry in the character offset table of the desired character is subtracted from the next entry. The result is the width of the character in pixels. `_DELY` must be loaded with the value of byte 82-83 of the header.

The `_STYLE` is something special. Here you can specify if characters should be displayed normally or changed. The possible changes are boldface (thicken, bit 0), shading (lighten, bit 1), italic (bit 2), and outline (bit 4). The given change is enabled by setting the corresponding bit. Another change is scaling. The size of a character can be changed through scaling. Unfortunately, characters can only be enlarged on the ST.

If the scaling flag is cleared (zero), the character is displayed in its original size. The `_T_SCLSTS` flag determines if the font is to be reduced or enlarged. A value other than zero must be placed here for enlarging. `_DDA_INC` should contain the value of the enlargement or reduction. An enlargement could be produced only with a value of \$FFFF.

Another interesting variable is `_CHUP`. With the help of this variable, characters can be rotated on the screen. The angle must be given in the range 0 to 360 degrees in tenths of a degree. A restriction must also be made for this function. Usuable results are obtainable only with rotations by 90 degrees. The values are \$0000 for normal, \$0384 for 90-degree rotation, \$0704 (upside-down type), and \$0A8C for 270 degrees.

To work with the effects, `_scrchp` must contain a pointer to a buffer in which `TEXTBLT` can store temporary values. The exact size of this buffer is not known, but we always found a buffer of 1K to be sufficient. Another buffer must be specified for enlargement (`_scrtpt2`). An offset is passed as a parameter which refers to the start of the `_scrchp` buffer. A value of \$40 proved to be sufficient here.

## **\$A009 SHOW MOUSE**

Calling this opcode enables the display of the mouse cursor. The cursor follows the mouse when it is moved. If the mouse cursor is disabled, the mouse can be used in programs which abandon the user interface GEM. This option is particularly useful for games.

The parameters required are passed in the `INTIN` and `CONTRL` arrays. `CONTRL(1)` should be cleared before the call and `CONTRL(3)` set to one. `INTIN(0)` has a special significance. The routine for managing the mouse cursor counts the number of calls to remove and enable the cursor. If the cursor is disabled twice, two calls must be made to re-enable it before it will actually appear on the screen. This behavior can be changed by clearing `INTIN(0)`. With this parameter the cursor is immediately set independent of the number of previous `HIDE CURSOR` calls. If the value in `INTIN(0)` is not equal to zero the actually required number of `$A009` calls must be made in order to make the cursor visible.

## **\$A00A HIDE CURSOR**

This functions hides the cursor. If this function is called repeatedly, the number is recorded by the operating system and determines the number of calls of `SHOW CURSOR` before the cursor actually appears.

## **\$A00B TRANSFORM MOUSE**

Is the arrow unsuited as a mouse cursor for games? Simply make your own cursor. How would it be if a little car moved across the screen instead of an arrow? The opcode `$A00B` gives your fantasy free reign, at least as far as it concerns the mouse cursor.

The parameters must be passed in the `INTIN` array. A total of 34 words are necessary. The following table gives information about the use and possible values:

`INTIN(3)` Mask color index, normally 0  
`INTIN(4)` Data color index, normally 1  
`INTIN(5)` to `INTIN(20)` contain 16 words of the cursor mask  
`INTIN(21)` to `INTIN (36)` contain 16 words of cursor data

The form of the cursor is determined by the cursor data. Each 1 in the data creates a point on the screen. If a cursor is placed over a letter or pattern on the screen, the border between the cursor and the background cannot be determined. The mask enters at this point. Each set bit in the mask clears the background at the given location. This permits a light border to be drawn around the cursor. Take a look at the normal arrow cursor in order to see the operation of the mask.

## **\$A00C UNDRAW SPRITE**

This opcode is related to \$A00D, DRAW SPRITE. The ST actually has no hardware sprites in sense in which sprite is used on something like the Commodore 64. The ST sprites are organized purely in software. Each sprite is 16x16 pixels large. One example of an ST sprite is the mouse cursor. It is created with this function.

In order to clear a previously-drawn sprite, the address of a buffer in which the background was saved when the sprite was drawn is passed in register A2. The opcode simply transfers the contents of the background buffer to the right spot on the screen. The buffer itself must be 64 bytes large for each plane. Another 10 bytes are used, independent of the number of planes. For monochrome display, the buffer is a total of 74 bytes long, while in the 320x200 pixel resolution (for planes), it is  $4 \times 64 + 10 = 266$  bytes large.

## **\$A00D DRAW SPRITE**

This function draws the desired sprite on the screen. Parameters must be passed in the D0, D1, A0, and A2 registers.

D0 and D1 contain the X and Y-coordinates of the position of the sprite on the screen, called the hot spot. A0 is a pointer to the so-called sprite definition block and A2 contains the address of the sprite buffer in which the background will be saved for erasing the sprite later.

The sprite definition block must have the following construction:

- Word 1 : X offset to hot spot
- Word 2 : Y offset to hot spot
- Word 3 : Format flag 0=VDI format, 1=XOR format
- Word 4 : Background color (bg)
- Word 5 : Foreground color (fg)

Following this are 32 words which contain the sprite pattern. The pattern must be in memory in the following order:

Word 6 : Background pattern of the top line  
 Word 7 : Foreground pattern of the top line  
 Word 8 : Background pattern of the second line  
 Word 9 : Foreground pattern of the second line  
 etc.

The information in the format flag has the following significance:

VDI Format		
fg	bg	Result
0	0	The background appears
0	1	The color in word 4 appears
1	0	The color in word 5 appears
1	1	The color in word 5 appears

XOR Format		
fg	bg	Result
0	0	The background appears
0	1	The color in word 4 appears
1	0	The pixel on the screen is XORed with the fb bit
1	1	The color in word 5 appears

## \$A00E COPY RASTER FORM

Arbitrary areas of the screen can be copied with the \$A00E opcode. Not only areas within the screen, but also from the screen into free RAM, and even more important, from the RAM to the screen. Even complete screen pages can be copied very quickly with the COPY RASTER opcode. The name RASTER FORM does express one limitation of the function, however. Each raster form to be copied must begin on a word boundary and must be a set of words.

The parameters are quite numerous and are passed in the CTRL, PTSIN, and INTIN arrays. In addition, two "memory form definition" blocks must be in memory for COPY RASTER. We will start with the MFD blocks. Since a copy operation must always have a source and a destination, one block describes the source memory range and the second describes the destination. Each block consists of 10 words. The address of the memory



described by the block is contained in the first two words. The third word specifies the height of the form in pixels. Word 4 determines the width of the form in words. Word 6 should be set to 1 and word 7 specifies the number of planes of which the form is composed. The remaining words should be set to zero because they are reserved for future extensions.

### 3.4.1 An overview of the "line-A" variables

After the initialization \$A000, D0 and A0 contain the address of a variable area which contains more than 50 line-A variables. The essential variables have been described along with the various calls, but not the location of the variables within the variable block. We will present this list shortly. When naming the variables we have remained with the names used in the official Atari documentation.

Offset is the value which must be given to access the value register relative. Variables supplied with a question mark could not be definitively explained.

Offset	Name	Size	Function
0	v_planes	word	Number of planes
2	v_lin_wr	word	Bytes per scan line
4	CONTRL	long	Pointer to the CONTRL array
8	INTIN	long	Pointer to the INTIN array
12	PTSIN	long	Pointer to the PTSIN array
16	INTOUT	long	Pointer to the INTOUT array
20	PTSOUT	long	Pointer to the PTSOUT array
24	_FG_BP_1	word	Plane 0 color value
26	_FG_BP_2	word	Plane 1 color value
28	_FG_BP_2	word	Plane 2 color value
28	_FG_BP_2	word	Plane 3 color value
32	_LSTLIN	word	Should be -1 (\$FFFF) (?)
34	_LN_MASK	word	Line pattern for \$A003
36	_WRT_MODE	word	Write mode (0=write mode 1=transparent 2=XOR mode 3=Inverse trans.)
38	_X1	word	X1-coordinate
40	_Y1	word	Y1-coordinate
42	_X2	word	X2-coordinate
44	_Y2	word	Y2-coordinate

---

46	<code>_patptr</code>	long	Pointer to the fill pattern (see \$A004)
50	<code>_patmsk</code>	word	Fill pattern "mask" (see \$A004)
52	<code>_multifill</code>	word	0=fill pattern is only for one plane 1=fill pattern is for multi- plane
54	<code>_CLIP</code>	word	0=no clipping (see \$A005) not 0=clipping
56	<code>_XMN_CLIP</code>	word	and
58	<code>_YMN_CLIP</code>	word	define upper left corner of the visible area for clipping
60	<code>_XMX_CLIP</code>	word	and
62	<code>_YMX_CLIP</code>	word	define lower right corner of the visible area for clipping
64	<code>_XACC_DDA</code>	word	Should be set to \$8000 before each call to TXTBLT (?)
66	<code>_DDA_INC</code>	word	Enlargement/reduction factor \$FFFF for enlargement, reduction doesn't work (?)
68	<code>_T_SCLSTS</code>	word	0=reduction (?) 1=enlargement
70	<code>_MONO_STATUS</code>	word	1=not proportional font 0=proportional type or width of character changed by bold or italics
72	<code>_SOURCEX</code>	word	X-coordinate of char in font
74	<code>_SOURCEY</code>	word	Y-coord of char in font (0)

**Note:**

SOURCEX is the value of the character from the horizontal offset table (HOT) and can be calculated with the following formula:

$SOURCEX = HOT\text{-}element\ (ASCII\ value\ minus\ FIRST\ ADE)$

The variable FIRST ADE is contained in bytes 36,37 of the font header (see example)

76	<code>_DESTX</code>	word	X-position of char on screen
78	<code>_DESTY</code>	word	Y-position of char on screen
80	<code>_DELX</code>	word	Width of the character
82	<code>_DELY</code>	word	Height of the character

Note:

DELX can be calculated with this formula:

DELX = SOURCEX+1 minus SOURCEX

(see \$A008)

DELY is the value FORM height from bytes 82,83 of the font header.

84	_FBASE	long	Pointer to start of font data
88	_FWIDTH	long	Width of font form
90	_STYLE	word	Flags for special effects (see \$A008)
92	_LITEMASK	word	Mask for shading
94	_SKEWMASK	word	Mask for italic type
96	_WEIGHT	word	Number of bits by which the character will be expanded
98	_R_OFF	word	Offset for italic type
100	_L_OFF	word	Offset for italic type

Note:

The above five variables should be loaded with the corresponding values from the font header.

102	_SCALE	word	0=no scaling 1=scaling (enlarge/reduce)
104	_CHUP	word	Angle for character rotation 0=normal char representation \$384=rotated 90 degrees \$708=rotated 180 degrees \$A8C=rotated 270 degrees
106	_TEXT_FG	word	Foreground color for text display
108	_scrtchp	long	Address of buffer required for creating special text effects
112	_script2	word	Offset of the enlargement buffer in the scrtchp buffer
114	_TEXT_BG	word	Background color for text rep
116	_COPYTRAN	word	(?)

### 3.4.2 Examples for using the line-A opcodes

In order to ease your first experiments with the line-A opcodes, we have given a few examples which can serve as a starting-point for you. In the first example, a point is set on the screen with \$A001, and then the color of the point is determined with \$A002.

```
*****
*          Demo of the $A000,$A001 and $A002 functions
*
*          rbr 09/28/85
*****

intin      equ      8
ptsin      equ      12

init       equ      $a000
setpix     equ      $a001
getpix     equ      $a002

start:
        .dc.w      init          * call $A000
        move.l     intin(a0),a3   * address of INTIN-arrays
        move.l     ptsin(a0),a4   * address of PTSIN-arrays

        move       #300,(a4)      * X coordinate
        move       #100,2(a4)     * Y coordinate

        move       #1,(a3)        * color set, pixel set
*                                0 erase pixel

        .dc.w      setpix        * pixel set

        move       #300,(a4)      * X coordinate
        move       #100,2(a4)     * y coordinate

        .dc.w      getpix        * get color value

*          d0 contains present color value
```

Only color values zero and one make sense for a monochrome monitor. Other values can be entered when working in one of the color modes, however.

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.

```
*****
*
*
*          a006 - filled pologyn
*
*
*****

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

yl          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          * address of variable block
*                                     from A0

*
*          move.w     #1,fg_bp1(a0)  *set colors for monochrome
*          clr.w      fg_bp2(a0)
*          clr.w      fg_bp3(a0)
*          clr.w      fg_bp4(a0)
*
*          move.w     #2,wrt_mod(a0) * replace mode
```

```

        move.l    #fill,patptr(a0)    * pointer of the fill pattern
        move.w    #4,patmsk(a0)      * four fill patterns
        clr.w     multifill(a0)      * for monochrome
        clr.w     clip(a0)           * no clipping

*
        move.l    contrl(a0),a6       * address of CONTRL array
                                         from A6
        addq.l    #2,a6               * A6 > CONTRL(1)
        move.w    #3,(a6)            * the XY pair in PTSIN

        move.l    ptsin(a0),a6       * address PTSIN array from A6
        move.l    #tab,a5            * table of coordinates
        move.w    #8,d3              * recieve 8 coordinates
loop     move.w    (a5)+,(a6)+
        dbra      d3,loop

loop1    move.w    #100,d3            * first scanline
        move.w    d3,y1(a0)          * from Y1
        move.l    a0,-(sp)           * restore address variable block

        dc.w      polygon            * fill scanline, A0 destroyed

        move.l    (sp)+,a0            * A0 restored
        addq.w    #1,d3              * calculate next scanline
        cmp.w     #301,d3            * last scan line?
        bne       loop1              * no, next scanline

        move.w    #1,-(sp)           * Code CONIN wait for keypress
        trap      #1                 * Call GEMDOS
        addq.l    #2,sp              * stack correction

*
        rts                          * subroutine all done
        move.w    #0,-(sp)           * terminate to desktop
        trap      #1                 * Call GEMDOS

fill:
        dc.w      %1100110011001100
        dc.w      %0110110110110110
        dc.w      %0011001100110011
        dc.w      %1001100110011001

tab:
        dc.w      320,100
        dc.w      120,320
        dc.w      520,300
        dc.w      320,100

```

The next example shows how the mouse form can be manipulated and how the mouse can be enabled. The example waits for a key press before returning.

```
*****
*
*
*      show mouse - transform mouse
*
*
*****
```

```
intin      equ      8
```

```
init_a     equ      $a000
```

```
show_mouse equ      $a009
```

```
transmouse equ      $a00b
```

```
start:
```

```
    .dc.w    init_a        * address INIT from A5
```

```
    move.l   intin(a0),a5
```

```
    move     #0,6(a5)      * INTIN (3) = mask color value
```

```
    move     #1,8(a5)      * INTIN (4) = data color value
```

```
    add.l    #10,a5        * a5 > INTIN (5)
```

```
    lea      maus,a4       * data for new cursor
```

```
    move     #16,d0        * 32 words = 16 longs
```

```
loop:
```

```
    move.l   (a4)+,(a5)+   * transfer INTIN array
```

```
    dbra     d0,loop
```

```
    .dc.w    transmouse    * and set form
```

```
    .dc.w    init_a
```

```
    move.l   intin(a0),a0
```

```
    clr.w    (a0)          * Number Hide Cursor -ignore call
```

```
    .dc.w    show_mouse    * now the new cursor
```

```

        move.w    #1,-(sp)      * Code CONIN wait for keypress
        trap      #1           * Call GEMDOS
        addq.l     #2,sp        * stack correction

*
        rts                * subroutine all done
        move.w    #0,-(sp)      * terminate to desktop
        trap      #1           * Call GEMDOS

```

maus:

maske:

```

        .dc.w      %000000001100000000
        .dc.w      %000000111111000000
        .dc.w      %0001111111111000
        .dc.w      %0111111111111110
        .dc.w      %1111111111111111
        .dc.w      %1111001111001111
        .dc.w      %1111001111001111
        .dc.w      %1111001111001111
        .dc.w      %000000011110000000
        .dc.w      %000000011110000000
        .dc.w      %000000011110000000
        .dc.w      %000000011110000000
        .dc.w      %000000011110000000
        .dc.w      %000000011110000000
        .dc.w      %000000000000000000

```

daten:

```

        .dc.w      %000000000000000000
        .dc.w      %000000000000000000
        .dc.w      %000000001100000000
        .dc.w      %000000110011000000
        .dc.w      %0110000110000110
        .dc.w      %0110000110000110
        .dc.w      %000000001100000000
        .dc.w      %000000001100000000
        .dc.w      %000000001100000000
        .dc.w      %000000001100000000
        .dc.w      %000000001100000000
        .dc.w      %000000001100000000
        .dc.w      %000000001100000000
        .dc.w      %000000000000000000
        .dc.w      %000000000000000000

```



### 3.5 The Exception Vectors

The first 1024 bytes of the 68000 processor are reserved for the exception vectors. Routines which use exception handling store the addresses they require in this range of memory.

A condition which leads to an exception can come either from the processor itself or from the peripheral components and controls units connected to it. The interrupts, described in the next section, belong to the class of external events. In addition, a so-called bus error can be created externally.

A bus error can be created by many circumstances. For one, certain memory areas can be protected from unauthorized access by it. As you may already know, the 68000 can run in one of two operating modes. The operating system is driven at the first level, the *supervisor mode*. The *user mode* is intended for user programs. In order that a user program not be able to access important system variables as well as the system components in an uncontrolled fashion, such an access in the user mode leads to a bus error. If such an error occurs, the processor stops execution of the instruction, saves the program counter and status register on the stack, and branches to a routine, the address of which it fetches from the lowest 1024 bytes of memory. In the case of the bus error, the address is at memory location 8 (one long word). What happens in this routine?

First the vector number of the interrupt is determined. In the case of a bus error, this is 2. Mushroom clouds are then displayed on the screen. The user can determine the vector number by counting the number of mushroom pictures. Execution then returns to the GEM desktop.

The following table contains all of the exception vectors.

---

Vector number	Address	Exception vector meaning
0	\$000	Stack pointer after reset
1	\$004	Program counter after reset
2	\$008	Bus error
3	\$00C	Address error
4	\$010	Illegal instruction
5	\$014	Division by zero
6	\$018	CHK instruction
7	\$01C	TRAPV instruction
8	\$020	Privilege violation
9	\$024	Trace
10	\$028	Line A emulator
11	\$02C	Line F emulator
12-14	\$030-\$038	reserved
15	\$03C	Uninitialized interrupt
16-23	\$040-\$05C	reserved
24	\$060	Spurious 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 instruction
33	\$084	TRAP #1 instruction
34	\$088	TRAP #2 instruction
35	\$08C	TRAP #3 instruction
36	\$090	TRAP #4 instruction
37	\$094	TRAP #5 instruction
38	\$098	TRAP #6 instruction
39	\$09C	TRAP #7 instruction
40	\$0A0	TRAP #8 instruction
41	\$0A4	TRAP #9 instruction
42	\$0A8	TRAP #10 instruction
43	\$0AC	TRAP #11 instruction
44	\$0B0	TRAP #12 instruction
45	\$0B4	TRAP #13 instruction
46	\$0B8	TRAP #14 instruction
47	\$0BC	TRAP #15 instruction
48-63	\$0C0-\$0FC	reserved
64-255	\$100-\$3FC	User interrupt vectors

The following vectors are used on the ST:

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

The vector for division by zero points to `rte` and returns directly to the interrupted program. Vectors 64-79 are reserved for the MFP 68901 interrupts. All other vectors point to \$5838 which outputs the vector number and ends the program as described for the bus error.

All of the unused vectors can be used for your own purposes, such as the line F emulator or the 12 unused traps.

### 3.5.1 The interrupt structure of the ST

The interrupt possibilities which the 6800 microprocessor offers are put to good use in the ST. As you may have already gathered from the hardware description of the processor, the processor has seven interrupt levels with different priorities. The interrupt mask in the system byte of the status register determines which levels can generate an interrupt. An interrupt can only be generated by a level higher than the current contents of the mask in the status register. A interrupt of a certain priority is communicated to the processor by the three interrupt priority level inputs. The following assignment results:

Level	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

If all three lines are 1 (interrupt level 0), no interrupt is present. Interrupt level 7 is the NMI (non-maskable interrupt), which is executed even if the interrupt mask in the status register contains seven. Which interrupt is assigned which vector (that is, the address of the routine which will process the interrupt) depends on the peripheral component which generates the interrupt. For auto-vectors, the processor itself derives the interrupt number from the interrupt level. The following table is used in this process:

Level	Vector number	Vector address
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

Only lines IPL 1 and IPL 2 are used on the Atari ST; Line IPL 1 is permanently set to a 1 level so that only levels 2, 4 and 6 are available. The results in the following assignment:

IPL 2	HBL, horizontal blank, line return
IPL 4	VBL, vertical blank, picture return
IPL 6	MFP 68901

The HPL interrupt is generated on each line return from the video section. It is generated every 50 to 64  $\mu$ s depending on the monitor connected (monochrome or color). It occurs very often and is normally not permitted by an interrupt mask of three. The standard HBL routine therefore only has the task of setting the interrupt mask to three if it is zero and allows the HBL interrupt so that no more HBL interrupts will occur. One use of the HBL interrupt could be for special screen effects. With the help of this routine, you know exactly which line of the screen has just been displayed. Of much greater importance, however, is the VBL interrupt, which is generated on each picture return. This occurs 50, 60, or 70 times per second depending on the monitor.

The vertical blank interrupt (VBL) routine accomplishes a whole set of a tasks which must be periodically executed or which concern the screen display. When entering the routine, the frame counter `frclock` (\$466) is first incremented. Next, a test is made to see if the VBL interrupt is software-disabled. This is the case if `vblsem` (\$452) (vertical blank semaphore) is zero or negative. In this case the routine is exited immediately

and execution returns to the interrupted program. Otherwise, all of the registers are saved on the stack and the counter `vbclock` (\$462), which counts the executed VBL routines, is incremented. Next, a check is made to see if a different monitor has been connected in the meantime. If a change was made from a monochrome to color monitor, the video shifter is reprogrammed accordingly. This is necessary because the high screen frequency of 70 Hz of the monochrome monitor could damage a color monitor. The routine to flash the cursor is called next. If you load a new color palette via the appropriate BIOS functions or want to change the screen address, this happens here in the VBL routine. Since nothing is displayed at this time, a change can be made here without disturbing anything else. If `colorptr` (\$45A) is not equal to zero, it is interpreted as a pointer to a new color palette, and this is loaded into the video shifter. The pointer is then cleared again. If `screenptr` is set, this value is used as the new base address of the screen. This takes care of the screen specific portions.

Now the floppy VBL routine is called, which with help the of the write protect status, determines if a diskette was changed. An additional task of this routine is to deselect the drives after the disk controller has turned the drive motor off.

Now comes the most interesting part for the programmer, the processing of the VBL queue. There is a way to tell the operating system to execute your own routines within the VBL interrupt. The maximum number of routines possible is in `nvbls` (\$454). This value is normally initialized to 8, but it can be increased if required. Address `vblqueue` (\$456) contains a pointer to a vector array which contains the (8) addresses of the VBL routines. Each address is tested within the VBL routine and the corresponding routine executed if the address is not zero.

If you want to install your own VBL routine, check the 8 entries until you find one which contains a zero. At this address you can write a pointer to your routine which from now on will be executed in every VBL interrupt. In all 8 entries are already occupied, you can copy the entries into a free area of memory, append the address of your routine, and redirect `vblqueue` to point to the new vector array. Naturally, you must not forget to increment `vbls`, the number of routines, correspondingly. Your routine may change all registers with the exception of the USP.

As soon as the VBL routine is done, the `dmpflg` (\$4EE) is checked. If this memory location is zero, a hardcopy of the screen is outputted. The flag is set in the keyboard interrupt routine if the keys ALT and HELP are pressed

at the same time. Finally, the register contents are restored, vblsem is released and execution returns to the interrupted routine.

The MFP 68901 occupies interrupt level six in our previous table. This component is in the position to create interrupt vectors on its own. These are referred to non-auto vectors in contrast to the auto vectors used above, because the processor does not generate the vector itself. In the Atari ST, the MFP 68901 works as the interrupt controller. It manages the interrupt requests of all peripheral components including its own.

The MFP can manage sixteen interrupts which are prioritized in reference to each other, similar to the seven levels of the processor. All MFP interrupts appear on level 6 to the 68000, therefore prioritized higher than HBL and VBL interrupts. The following table contains the assignments within the MFP.

Level	Assignment
15	Monochrome monitor detect
14	RS-232 ring indicator
13	System clock timer A
12	RS-232 receive buffer full
11	RS-232 receive error
10	RS-232 transmit buffer empty
9	RS-232 transmit error
8	Line return counter, timer B
7	Floppy controller and DMA
6	Keyboard and MIDI ACIAs
5	Timer C
4	RS-232 baud rate generator, timer D
3	unused
2	RS-232 CTS
1	RS-232 DCD
0	Centronics busy

Not all of these possible interrupt sources are enabled, however. Some signals are processed through polling. The following is a description of the interrupts which are used by the operating system.

**Level 2, RS-232 CTS, Address \$73C0**

This interrupt is generated every time the RS-232 interface is informed via the CTS line that a connected receiver is ready to receive additional data. The routine then sends the next character from the RS-232 transmit buffer.

**Level 5, Timer C, Address \$7C5C**

This timer runs at 200 Hz. The 200 Hz counter at \$4BA is first incremented in the interrupt routine. The next actions are performed only every fourth call to the interrupt routine, that is, only every 20ms (50 Hz). First a routine is called which handles the sound processing. Another task of this interrupt is the keyboard repeat when a key is pressed and initial repeat. Finally, the evt timer routine of GEM is called, which is accessed via vector \$400.

**Level 6, Keyboard and Midi, Address \$752A**

Two peripheral components are connected to this interrupt level of the MFP, the two ACIAs which receive data from the keyboard and the MIDI interface. In order to decide which of the two components has requested an interrupt, the interrupt request bits in the status registers of the ACIAs are tested and the received byte is fetched if required. If it comes from the keyboard, the scan code is converted to the ASCII code by means of the keyboard table and written into the receive buffer, which happens immediately for MIDI data. Mouse and joystick data also come from the keyboard ACIA and are also prepared accordingly.

**Level 9, RS-232 transmit error, Address \$7426**

If an error occurs while sending RS-232 data, this interrupt routine is activated. Here the transmitter status register is read and the status is saved in the RS-232 parameter block.

**Level 10, RS-232 transmit buffer empty, Address \$7374**

Each time the MFP has completely outputted a data byte via the RS-232 interface, it generates this interrupt. It is then ready to send the next byte. If data is still in the transmit buffer, the next byte is written into the transmit register, which can now be shifted out according to the selected baud rate.

**Level 11, RS-232 receive error, Address \$7408**

If an error occurs when receiving RS-232 data, this interrupt routine is activated. This may involve a parity error or an overflow. The routine only clears the receiver status register and then returns.

**Level 12, RS-232 receive buffer full, Address \$72C0**

If the MFP has received a complete byte, this interrupt occurs. Here the character can be fetched and written into the receive buffer (if there is still room). This routine takes into account the active handshake mode (sending XON/XOFF or RTS/CTS).

The other interrupt possibilities of the MFP are not used, but they can be used for your own routines. For example, interrupt level 0, Centronics strobe, can be used for buffered printer output.



### 3.6 The Atari ST VT52 Emulator

There are two options for text output on the ST. You can work with the GEMDOS functions by means of TRAP #1 or a direct BIOS call with TRAP #13. The other possibility consists of using the VDI functions.

You have special possibilities for screen control with both variants. We will first take a look at output using the normal DOS or BIOS calls. Here a terminal of type VT52, which offers a wide variety of control functions, is emulated for screen output. These control characters are prefixed with a special character, the escape code. Escape, also shortened to ESC, has ASCII code 27. Following the escape code is a letter which determines the function, as well as additional parameters if required. The following list contains all of the control codes and their significance.

**ESC A Cursor up**

This function moves the cursor up one line. If the cursor was already on the top line, nothing happens.

**ESC B Cursor down**

This ESC sequence positions the cursor one line down. If the cursor is already on the bottom line, nothing happens.

**ESC C Cursor right**

This sequence moves the cursor one column to the right.

**ESC D Cursor left**

Moves the cursor one position to the left. This function is identical to the control code backspace (BS, ASCII code 8). If the cursor is already in the first column, nothing happens.

**ESC E Clear Home**

This control sequence clears the entire screen and positions the cursor in the upper left corner of the screen (home position).

**ESC H Cursor home**

With this function you can place the cursor in the upper left corner of the screen without erasing the contents of the screen.

**ESC I Cursor up**

This sequence moves the cursor one line towards the top. In contrast to ESC A, however, if the cursor is already in the top line, a blank line is inserted and the remainder of the screen is scrolled down a line correspondingly. The column position of the cursor remains unchanged.

**ESC J Clear below cursor**

By means of this function, the rest of the screen below the current cursor position is cleared. The cursor position itself is not changed.

**ESC K Clear remainder of line**

This ESC sequence clears the rest of the line in which the cursor is found. The cursor position itself is also cleared, but the position is not changed.

**ESC L Insert line**

This makes it possible to insert a blank line at the current cursor position. The remainder of the screen is shifted down; the lowest line is then lost. The cursor is placed at the start of the new line after the insertion.

**ESC M Delete line**

This function clears the line in which the cursor is found and moves the rest of the screen up one line. The lowest screen line then becomes free. After the deletion, the cursor is located in the first column of the line moved up to take the place of the old one.

**ESC Y Position cursor**

This is the most important function. It allows the cursor to be positioned at any place on the screen. The function needs the cursor line and column as parameters, which are expected in this order with an offset of 32. If you want to set the cursor to line 7, column 40, you must output the sequence `ESC Y CHR$(32+7) CHR$(32+40)`. Lines and columns are counter starting at zero; for an 80x25 screen the lines are numbered from 0 to 24 and the columns from 0 to 79.

The additional ESC sequences of the VT52 terminal start with a lower case letter.

**ESC b Select character color**

With this function you can select the character color for further output. With a monochrome monitor you have choice between just 0=white and 1=black. For color display you can select from 4 or 16 colors depending on the mode. Only the lowest four bits of the parameters are evaluated (mod 16). You can use the digit "1" for the color 1 as well as the letters "A" or "a" in addition to binary one.

**ESC c Select background color**

This function serves to select the background color in a similar manner. If you choose the same color for character and background, you will, of course, not be able to see text output any more.

**ESC d Clear screen to cursor position**

This sequence causes the screen to be erased starting at the top and going to the current position of the cursor, inclusive. The position of the cursor is not changed.

**ESC e Enable cursor**

Through this escape sequence the cursor becomes visible. The cursor can, for example, be enabled when waiting for input from the user.

**ESC f Disable cursor**

Turns the cursor off again.

**ESC j Save cursor position**

If you want to save the current position of the cursor, you can use this sequence to do so.

**ESC k Set cursor to the saved position**

This is the counterpart of the above function. It sets the cursor to the position which was previously saved with ESC j.

**ESC l Clear line**

Clears the line in which the cursor is located. The remaining lines remain unaffected. After the line is cleared, the cursor is located in the first column of the line.

**ESC o Clear from start**

This clears the current cursor line from the start to the cursor position, inclusive. The position of the cursor remains unchanged.

**ESC p Reverse on**

The reverse (inverted) output is enabled with this sequence. For all further output, the character and background colors are exchanged. With a monochrome monitor you get white type on a black background.

**ESC q Reverse off**

This sequence serves to re-enable the normal character display mode.

**ESC v Automatic overflow on**

After executing this sequence, an attempted output beyond the end of line will automatically start a new line.

**ESC w Automatic overflow off**

This deactivates the above sequence. An attempt to write beyond the line will result in all following characters being written in the last column.

Similar functions are available to you under VDI. The VDI escape functions (opcode 5) serve this purpose. The appropriate screen function is selected by choosing the proper function number. Note, however, that under VDI the line and column numbering does not begin with zero but with one.

Under VDI there is also a function which outputs a string at specific screen coordinates. If necessary, you can use the ESC functions of the VT52 emulation in addition.

The output of "unprintable" control characters

The three system fonts of the ST have also been supplied with characters for the ASCII codes zero to 31, which are normally interpreted as control codes. On the ST, only codes 7 (BEL), 8 (BS backspace), 9 (TAB), as well as 10, 11, and 12 (LF linefeed, VT vertical tab, and FF form feed all generate a linefeed) plus 13 (CR carriage return) have effect, in addition to ESC. The remaining codes have no effect. How does one access the characters below 32?

To do this, an additional device number is provided in the BIOS function 3 "conout". Normally number 2 "con" serves for output to the screen. If one selects number 5, however, all the codes from, 0 to 255 are outputted as printable characters, control codes are no longer taken into account.

In the appendix you find the three ST system fonts pictured.

### 3.7 The ST System Variables

The ST uses a set of system variables whose significance and addresses will not change in future versions of the operating system. If you use other variables, such as those from the BIOS listing which are not listed here, you should always remember that these could have a different meaning in a new version of the operating system. The system variables are in the lower RAM area directly above the 68000 exception vectors, at address \$400 to 1024. The address range from 0 to \$800 (2048) can be accessed only in the supervisor mode. An access in the user mode of the 68000 leads to a bus error.

In the following listing we will use the original names from Atari. In addition to the address of the given variable, typical contents and the significance will be described.

Address	length	name	Sample contents
---------	--------	------	-----------------

\$400	L	etv_timer	\$F526
-------	---	-----------	--------

This is the event timer vector of the GEM. It takes care of the periodic tasks of GEM.

\$404	L	etv_critlc	\$5562
-------	---	------------	--------

Critical error handler. Under GEM this pointer points to \$2A156. There an attempt is made to correct disk errors, such as if a another disk is requested in a single-drive system.

\$408	L	etv_term	\$5328
-------	---	----------	--------

This is the GEM vector for ending a program.

\$40C	5L	etv_xtra	
-------	----	----------	--

Here is space for 5 additional GEM vectors, which at the time are not yet used.

\$420    L     memvalid     \$752019F3

If the memory location contains the given value, the configuration of the memory controller is valid.

\$424    W     memctrl     \$0400

This is a copy of the configuration value in the memory controller. The value given applies for a 512K machine.

\$426    L     resvalid     \$31415926

If the given value is located here, a jump is made at a reset via the reset vector in address \$42A.

\$42A    L     resvector    \$FC0008

See above.

\$42E    L     phystop     \$80000

This is the physical end of the RAM memory; \$80000 for a 512K machine.

\$432    L     \_\_membot     \$3B900

The user memory begins here (TPA, transient program area).

\$436    L     \_\_memtop     \$78000

This is the upper end of the user memory.

\$43A    L     memval2     \$237698AA

This "magic" value together with "memvalid" declares the memory configuration valid.

\$43E    W     flock        0

If this variable contains a value other than zero, a disk access is in progress and the VBL disk routine is disabled.

**\$440    W    seekrate    3**

The seek rate (the time it takes to move the read/write head to the next track) is determined according to the following table:

Seek rate	Time
0	6 ms
1	12 ms
2	2 ms
3	3 ms

**\$442    W    \_timer\_ms    \$14, 20 ms**

The time span between two timer calls, 20 ms corresponds to 50 Hz.

**\$444    W    \_fverify    \$FF**

If this memory location contains a value other than zero, a verify is performed after every disk write access.

**\$446    W    \_bootdev    0**

Contains the device number of the drive from the operating system was loaded.

**\$448    W    palmode    0**

If this variable contains a value other than zero, the system is in the PAL mode (50 Hz); if the value is zero, it means the NTSC mode.

**\$44A    W    defshiftmod   0**

If the Atari is switched from monochrome to color, it gets the new resolution from here (0=low, 1 medium resolution).

**\$44C    W    sshiftmod    \$200**

Here is a copy of the register contents for the screen resolution.

0	320x200, low resolution
1	640x200, medium resolution
2	640x400, high resolution



\$44E    L     `_v_bas_ad`     \$78000

This variable contains a pointer to the video RAM (logical screen base). The screen address must always begin on a 256 byte boundary.

\$452    W     `vblsem`        1

If this variable is zero, the vertical blank routine is not executed.

\$454    W     `nvbls`         8

Number of vertical blank routines.

\$456    L     `_vblqueue`     \$4CE

Pointer to a list of `nvbls` routines which will be executed during the VBL.

\$45A    L     `colorptr`      0

If this value is not zero, it is interpreted as a pointer to a color palette which will be loaded at the next VBL.

\$45E    L     `screenpt`      0

This is a pointer to the start of the video RAM, which will be set during the next VBL (zero if no new address is to be set).

\$462    L     `_vbclck`        \$2D26A

Counter for the number of VBL interrupts.

\$466    L     `_frclock`        \$2D267

Number of VBL routines executed (not disabled by `vblsem`).

\$46A    L     `hdv_init`        \$5AE8

Vector for hard disk initialization.

---

\$46E    L      **swv\_vec**      \$501E

Vector for changing the screen resolution. A branch is made via this vector with the screen resolution is changed (default is reset).

\$472    L      **hdv\_bpb**      \$5B6E

Vector to fetch the BIOS parameter block for a hard disk.

\$476    L      **hdv\_rw**      \$5D88

Read/write routine for a hard disk.

\$47A    L      **hdv\_boot**      \$60B2

Vector to a routine to reboot the hard disk.

\$47E    L      **hdv\_mediach**    \$5D1E

Media change routine for hard disk.

\$482    W      **\_comload**      0

If this variable is set to a value other than zero by the boot program, an attempt will be made to load a program called "COMMAND.PRG" after the operating system is loaded.

\$484    B      **conterm**      6

Attribute vector for console output

Bit	Meaning
0	Key click on/off
1	Key repeat on/off
2	Tone after CTRL G on/off
3	"kbshift" is returned in bits 24-31 for the BIOS function "conin"

\$485    B      **unused, reserved**

\$486    L      **trp14ret**      0

Return address for TRAP #14 call.

---

\$48A    L      **criticret**      0

Return address of the critical error handler

\$48E    4L    **themd**            0

Memory descriptor, filled out by the BIOS function getmpb.

\$49E    2W      \_\_\_\_\_md      0

Space for additional memory descriptors.

\$4A2    L      **savptr**            \$5CE

Pointer to a save area for the processor registers after a BIOS call.

\$4A6    W      **\_nflops**          2

Number of connected floppy disk drives.

\$4A8    L      **con\_state**      \$8AEE

Vector for screen output; set by ESC functions to the appropriate routine, for example.

\$4AC    W      **save\_row**          0

Temporary storage for cursor line when positioning the cursor with ESC Y.

\$4AE    L      **sav\_context**      0

Pointer to a temporary areas for exception handling.

\$4B2    2L    **\_buf1**            \$4F9E,    \$4FB2

Pointer to two buffer list headers of GEMDOS. The first header is responsible for data sectors, the second for the FAT (file allocation table) and the directory. Each buffer control block (BCB) is constructed as follows:

---

```

long   BCB      $4F8A, pointer to next BCB
int    drive    -1,   drive number or -1
int    type      2    buffer type
int    rec      $41C   record number in this buffer
int    dirty     0     dirty flag (buffer changed)
long   DMD      $2854  pointer to drive media descriptor
long   buffer    $4292  pointer to the buffer itself

```

```
$4BA   L      _hz_200      $71280
```

Counter for 200 Hz system clock

```
$4BC   4B      the_env      0
```

Default environment string, four zero bytes.

```
$4C2   L      _drvbits      3
```

32-bit vector for connected drives. Bit 0 stands for drive A, bit 1 for drive B, and so on.

```
$4C6   L      _diskbufp     $12BC
```

Pointer to a 1024-byte disk buffer. The buffer is used for GSX graphic operations and should not be used by interrupt routines.

```
$4CA   L      _autopath     0
```

Pointer to autoexecute path.

```
$4CE   8L      _vbl_list     $15398, 0, 0...
```

List of the standard VBL routines.

```
$4EE   W      _dumpflg      $FFFF
```

This flag is incremented by one when the ALT and HELP keys are pressed simultaneously. A value of one generates a hardcopy of the screen on the printer. A hardcopy can be interrupted by pressing ALT HELP again.

\$4F0    W    \_prtabt       0

Printer abort flag due to time-out.

\$4F2    L    \_sysbase     \$5000

Pointer to start of the operating system.

\$4F6    L    \_shell\_p     0

Global shell information.

\$4FA    L    end\_os       \$3B900

Pointer to the end of the operating system in RAM, start of the TPA.

\$4FE    L    exec\_os      \$1EB00

Pointer to the start of the AES. Normally branched to after the initialization of the BIOS.

### 3.8 The 68000 Instruction Set

If you are already familiar with the machine language of some 8-bit processor: Forget everything you know. If you do, it will make it easier to understand the following material!

The 68000 processor is fundamentally different in construction and architecture from previous processors (including the 8086!). The essential difference does not lie in the fact that the standard processing width is 16 and not 8 bits (which is sometimes a drawback and can lead to programming errors), but in the fact that, with certain exceptions, the internal registers are not assigned to a specific purpose, but can be viewed as general-purpose registers, with which almost anything is possible.

In earlier processors, the accumulator was always the destination for arithmetic operations, but it is completely absent in the 68000. There are eight data registers (D0-D7) with a width of 32 bits, and as a general rule, at least one of these is involved in an operation. There are also eight address registers (A0-A7), each with 32 bits, which are usually used for generating complex addresses. Register A7 has a set assignment—it serves as the stack pointer. It is also present twice, once as the user stack pointer (USP) and once as the supervisor stack pointer (SSP). The distinction is made because there are also two operating modes, namely the user mode and the supervisor mode.

These two are not only different in that they use different stack pointers, but in that certain instructions are not legal in the user mode. These are the so-called privileged instructions (see also instruction description), with whose help an unwary programmer can easily "crash" the system rather spectacularly. This is why these instructions create an exception in the user mode. An exception, by the way, is the only way to get from the user mode to the supervisor mode.

In addition there is the status register, the upper half of which is designated as the system byte because it contains such things as the interrupt mask, things which do not concern the "normal" user, making access to this byte also one of the privileged instructions. The lower byte, the user byte, contains the flags which are set or cleared based on the result of operations, such as the carry flag, zero flag, etc. As a general rule, the programmer works with these flags indirectly, such as when the execution of a branch is made conditional on the state of a flag.

Two things should be mentioned yet: Multi-byte values (addresses or operands) are not stored in memory as they are with 8-bit processors, in the order low byte/high byte, but the other way around. Four-byte expressions (long word) are stored in memory (and the registers of course) with the highest-order byte first.

The second is that unsupported opcodes do not lead to a crash, but cause a special exception, whose standard handling must naturally be performed by the operating system.

### 3.8.1 Addressing modes

This is probably the most interesting theme of the 68000 because the enormous capability first takes effect through the many various addressing modes.

The effective address (the address which, sometimes composed of several components, finally determines the operand) is fundamentally 32 bits wide, even if one or more the components specified in the instruction is shorter. These are always sign-extended to the full 32-bit width.

The charm of the addressing lies in the fact that almost all instructions (naturally with exceptions), both the source and destination operands, can be specified with one of the addressing modes. This means that even memory operations do not necessarily have to use one of the registers; memory-to-memory operations are possible.

In the assembler syntax, the source operand is given first, followed by the destination operand (behind the comma).

## Register Direct

The operand is located in a register. There are two kinds of register direct addressing: data register direct and address register direct.

In the first case, the operand may be bit, byte, word, or long word-oriented; in the second case a word or long word is required, in case the address register is the destination of the operation.

Example: `ADD.B D0,D1` or `ADDA.W D0,A2`

## Absolute Data Addressing

The operand is located in the address space of memory. This can also be a peripheral component, naturally (see `MOVEP`). The address is specified in absolute form.

This can have a width of a a long word, whereby the entire address space can be accessed, or it can be only one word wide. In this case is sign-extended (the sign being the highest-order bit) to 32 bits. For example, the word `$7FFF` becomes the long word `$00007FFF`, while `$FFFF` becomes `$FFFFFFFF`. Only the lower 32K and the upper 32K of the address space can be accessed with the short form. This addressing mode is often used in the operating system of the ST because important system variables are stored low in memory and all peripheral components are decoded at the top.

Example: `MOVE.L $7FFF,$01234567`

Instructions in which both operands are addressed with a long word are the longest instructions in the set, consisting of 10 bytes.



## Program Counter Relative Addressing

This addressing mode allows even constants to be addressed in a completely relocatable program, since the base of the address calculation is the current state of the program counter.

There are two variations. In the first, a 16-bit signed offset is added to the program counter, and in the second, the contents of a register (sign-extended if only one word is specified) are also added in, though here the offset may be only 8 bits long.

Example: `MOVE.B $1234(PC), $12(PC,D0.W)`

## Register Indirect Addressing

There are several variations of this, and they will be discussed individually.

### *Register Indirect*

Here the operand address is located in an address register.

Example: `CLR.L (A0)`

### *Postincrement Register Indirect*

The operand is addressed as above, but the contents of the address register are then incremented by the length of the operand, by 1 for xxx.B or 4 for xxx.L.

Example: `BSET.B #0, (A0)+` or `BCLR.L #23, (A1)+`

### *Predecrement Register Indirect*

Here the address register is decremented by the length of the operand before the addressing.

Example: `EOR.L D0, $1234(A4)`

### *Indexed Register Indirect with Offset*

As above, but the contents of another register (address or data) are also added in, taking the sign into account. The offset may have a width of 8 bits here, however.

Example: `MOVE.W $12(A5,A6.L),D1`

### **Immediate Addressing**

Here the operand is contained as such in the instruction itself. Naturally, an operand specified in this manner can serve only as a source. The immediate operands can, as a general rule, be any of the allowed widths.

Example: `ADDI.W #$1234,D5`

In the variant **QUICK**, the constant may be only 3 bits long, therefore having a value from 0-7. An exception is the **MOVE** command, where the constant may have 8 bits, but in which only a data register is allowed as the destination.

Example: `ADDQ.L #1,A0` or `MOVEQ #123,D1`

### **Implied Register**

This addressing mode is mentioned only for the sake of completeness and in it, an operand address is already determined by the instruction itself. The operands are either in the program counter, in the status register, or the system stack pointer.

Example: `MOVE SR,D6`

Regarding the offsets, it should be noted that they are signed numbers in two's complement. Their highest-order bit forms the sign. With an 8-bit value, an offset of +127/-128 is possible, and about  $\pm 32K$  with 16 bits.

### 3.8.2 The instructions

In the following instruction description, the individual bit patterns are not listed since this would lead to far in this connection. Additional information can be gathered from books like the *M68000 16/32-Bit Microprocessor Programmer's Reference Manual* (Motorola).

The instructions are also explained only in their base form and variations are mentioned only in name. We will briefly explain what the individual variations can look like here.

The variations are indicated by letter after the operand. This can be one of the following:

- A** indicates that the destination of the operation is an address register. Word operations are sign-extended to 32 bits.
- I** indicates an immediate operand as the source of the operation. I operands may assume all widths as a general width.
- Q** means quick and represents a special form of immediate addressing. Such an operand is usually three bits wide, corresponding to a value range of 0 to 7. This limited range has the advantage that the operand will fit into the opcode. Since there is no special command for incrementing a register, something like `ADDQ.L #1,A0` works well in its place. An exception is `MOVEQ`. Here the operand may have a value of 0-255.
- X** indicates arithmetic operations which use the X flag. This flag has a special significance. It is set equal to the carry flag for all arithmetic operations. The carry flag, however, is also affected by transfer operations while the X flag is not so that it remains available for further calculations. This is especially useful for computations with higher precision than the standard 32 bits, where temporary results must first be saved, and where the carry flag can be changed as a result.

All instructions have a suffix after the opcode of the form `.B`, `.W`, or `.L`. This suffix indicates the processing width of the operation. Although a data register, for example, has a width of 32 bits = 4 bytes = 1 long word, the instruction `CLR.B D0` clears only the lowest-order byte of the register. For registers, `.W` specifies the lower word. The higher-order word is not

explicitly addressable. If the operand is in memory, it is important to know that .W and .L operands must begin on an even address. The same applies for the opcode as such, which also always comprises one word.

If the destination of an operation is an address register, only operands of type .W and .L are allowed, whereby the first is sign-extended to a long word.

Some listings contain instructions of the form `MOVE.L #27,D0`. The programmer then assumes that the assembler will produce `#$0000001B` from `#27`.

Now to the individual instructions:

#### **ABCD** Add Decimal with Extend

There is one data format which we have not yet discussed: the BCD format. This means nothing more than "Binary-Coded Decimal" and it uses digits in the range 0-9. Since this information requires only 4 bits, a byte can store a two-digit decimal number. The instruction `ABCD` can then add two such numbers. The processing width is always 8 bits.

#### **ADD** Add Binary

This instruction simply adds two operands.  
Variations are `ADDA`, `ADDQ`, `ADDI`, and `ADDX`.

#### **AND** Logical AND

Two operands are logically combined with each other according to the AND function.  
Variation: `ANDI`

#### **ASL** Arithmetic Shift Left

The operand is shifted to the left byte by the number of positions given, whereby the highest-order bit is copied into the C and X flags. A 0 is shifted in at the right. If a data register is shifted, the processing width can be any. The number of places to be shifted is either specified as an I operand (3 bits) or is placed in an additional register. If a memory location is shifted, the processing width is always one word. A counter is then not given; it is always =1.

#### **ASR** Arithmetic Shift Right

The operand is shifted to the right, whereby the lowest bit is copied to C and X. The sign bit is shifted over from the left. See `ASL` for information about processing width and counter.

**Bcc Branch Conditionally**

The branch destination is always a relative address which is either one byte or one word long (signed!). Correspondingly, the branch can jump over a range of +127/-128 bytes or +32K-1/-32K. The point of reference is the address of the following instruction.

Whether or not this instruction is actually executed depends on the required condition, which is verified by means of the flags. Here are the variations and their conditions. A minus sign before a flag indicates that it must be cleared to satisfy the condition. Logical operations are indicated with "\*" for AND and "/" for OR.

BRA	Branch Always	no condition
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**

The specified bit of the operand will be inverted. The original state can be determined from the Z flag. The operand is located either in memory (width=.B) or in a data register (width=.L). The bit number is given either as an I operand or is located in a data register.

**BCLR Bit Test and Clear**

The specified bit is cleared. Everything else is handled as per BCHG.

**BSET Bit Test and Set**

The specified bit is set. Boundary conditions are per BCHG.

**BSR Branch to Subroutine**

This is an unconditional branch to a subroutine. Branch distances as for Bcc.

**BTST Bit Test**

The bit is only checked as to its condition. Everything else as per BCHG.

**CHK Check Register Against Boundaries**

A data register is checked to see if its contents are less than zero or greater than the operand. Should this be the case, the processor executes an exception. The program is continued at the address in memory location \$18 (vector 6). Otherwise no action is taken. The processing width is only word.

**CLR Clear Operand**

The specified operand is cleared (set to zero).

**CMP Compare**

The first operand is subtracted from the second without changing either of the two operands. Only the flags are set, according to the result.

Variations: *CMPA* and *CMPI*

Both operands are addresses with the addressing mode (Ax)+ with the variant CMPM.

**DBcc Test Condition, Decrement and Branch**

A data register is decremented and the flags are checked for the specified condition. A branch is performed if either the condition is fulfilled or the register is -1. Branch conditions and ranges as per Bcc.

**DIVS Divide Signed**

The second operand is divided by the first operand, taking the sign into account. Afterwards the second operand contains the integer quotient in the lower word and the remainder in the upper word, which has the same sign as the quotient. The data width of the first operand is set at .W and at .L for the second.

**DIVU Divide Unsigned**

Operation as above, but the sign is ignored.

**EOR Exclusive OR**

The two operands are logically combined according to the rules of EXOR.

Variations: *EORI*

**EXG Exchange Registers**

The two registers specified are exchanged with each other.

**EXT Sign Extend**

The operand is filled to the given processing width with its bit 7 (in the case of .B) or bit 15 (.W).

**JMP Jump**

Unconditional jump to the specified address. The difference between this and BRA is that here the address is not relative but absolute, that is, the actual jump destination.

**JSR Jump to Subroutine**

Jump to a subroutine. The difference from BSR is as above.

**LEA Load Effective Address**

This often-misunderstood instruction loads an address register not with the contents of the specified operand address as is normal for the other instructions, but *with the address as such!*

**LINK Link Stack**

This instruction first places the given address register on the stack. The contents of the stack pointer (A7) are then placed in this register and the offset specified is added to the stack pointer.

With this practical instruction, data areas can be reserved for a subroutine, without having to make room in the program itself, which would also be impossible in programs which run in ROM. The C-compiler makes extensive use of this capability for local variables.

**LSL Logical Shift Left**

Function and limitations as per ASL.

**LSR Logical Shift Right**

Function and limitations as per ASR, except here the sign is not shifted in on the left, but a 0.

**MOVE**

The first operand is transferred to the second.

Variations: *MOVEA*, *MOVEQ*

**MOVEM Move Multiple Registers**

Here an operand can consist of a list of registers. This can be used to place all of the registers on the stack, for instance.

Example: *MOVEM.L A0-A6/D0-D7, -(A7)*

**MOVEP** Move Peripheral Data

This speciality is made expressly for the operation of peripheral components. As a general rule, these work only with an 8-bit data bus, and are then connected only to the upper or lower 8 bits of the 68000's data bus. If a word or long word is to be transferred, the bytes must be passed over either the upper or lower byte of the data bus, depending on whether the address is even or odd. The address is then always incremented by two so that the transfer always continues on the same half of the data bus on which it was begun. Corresponding to the purpose of this instruction, one operand is always a data register, and the other is always of type register indirect with offset.

**MULS** Multiply Signed

Signed multiplication of two operands.

**MULU** Multiply Unsigned

Multiplication of two operands, ignoring the sign.

**NBCD** Negate Decimal with Extend

A BCD operand is subjected to the operation 0-operand X.

**NEG** Negate Binary

The operand is subjected to the treatment 0-operand.

Variations: *NEGX*

**NOP** No Operation

As the name says, this instruction doesn't do anything.

**NOT** One's Complement

The operand is inverted.

**OR** Logical OR

The two operands are combined according to the rule for logical OR.

**PEA** Push Effective Address

The address itself, not its contents, is placed on the stack.

**RESET** Reset External Devices

The reset line on the 68000 is bidirectional. Not only can the processor be externally reset, but it can also use this instruction to reset all of the peripheral devices connected to the reset line.

*This is a privileged instruction!*



**ROL Rotate Left**

The operand is shifted to the left, whereby the bit shifted out on the left will be shifted back in on the right and the carry flag is affected. Processing widths and shift counter as per ASL.

**ROR Rotate Right**

As above, but shift from left to right.

**ROXL Rotate Left with Extend**

As ROL, but the shifted bit is first placed in the X flag, the previous value of which is shifted in on the right.

**ROXR Rotate Right with Extend**

As above, but reversed shift direction.

**RTE Return from Exception**

Return from an exception routine to the location at which the exception occurred.

**RTS Return from Subroutine**

Return from a subroutine to the location at which it was called.

**RTR Return and Restore**

As above, but the CC register (the one with the flags) is first fetched from the stack (on which it *must* have first been placed, because otherwise execution will not return to the proper address.

**SBCD Subtract Decimal with Extend**

The first operand is subtracted from the second. Refer to ABCD for information on the data format.

**Scc Set Conditionally**

The operand (only .B) is set to \$FF if the condition is fulfilled. Otherwise it is cleared. Refer to Bcc for the possible condition codes.

**STOP**

The processor is stopped and can only be called back to life through an external interrupt.

*This is a privileged instruction!*

**SUB Subtract Binary**

The first operand is subtracted from the second.

**SWAP** Swap Register Halves

The two halves of a data register are exchanged with each other.

**TAS** Test and Set Operand

The operand (only .B) is checked for sign and 0 (affecting the C and N flags). Bit 7 is then set to 1.

**TRAP**

The applications programmer uses this instruction when he wants to call functions of the operating systems. This instruction generates an exception, which consists of continuing the program at the address determined by the given vector number. See the chapter on the BIOS and XBIOS for the use of this instruction.

**TRAPV** Trap on Overflow

If the V flag is set, an exception is generated by this instruction, resulting in program execution continuing at the address in vector 7 (\$1C).

**TST** Test

Action like TAS, but the operand is not changed.

**UNLK** Unlink

This instruction is the counterpart of LINK. The stack pointer (A7) is loaded with the given address register and this is supplied with the last stack entry. In this manner the area reserved with LINK is released.

Addendum to the condition codes: The conditions listed under Bcc are not complete, because the additional conditions do not make sense at that point. But the instructions DBcc and Scc have the additional variations T (DBT, ST) and F (DBF, SF). T stands for true and means that the condition is always fulfilled. F stands for false and is the opposite: the condition is never fulfilled.

### 3.9 The BIOS Listings - Version 1

```

*****
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 23F9000501E0000042A move.l  $501E,$42A
00502C 23FC31415926000000426 move.l  #$31415926,$426
005036 41F9FFF8800   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     $5060
005058 13FC0002FFFFF820A move.b  #2,$FFFFF820A
005060 43F9FFF8240   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
*****
ATARI ST BIOS
To program start
Version 1
Reset address
Start of the operating system
End of the operating system
Reset address

Creation date 6/20/1985

Supervisor mode, no interrupts
Load resvector
Resmagic to resvalid
Address of the sound chip
Port A and B
To output
Select port A
Deselect floppies

Syncmode
Address color0
16 colors
Address of the color table
Load color palette
Next color
Clear A5
End of the operating system variables
Start of the operating system
Clear D0
Clear memory range
End reached ?

```

005084 68FA	bne	\$5080	No
005086 208D042E	move.l	\$42E(A5), A0	Phystop, end of RAM
00508A 91FC00008000	sub.l	#8000, A0	Minus 32 K
005090 2B48044E	move.l	A0, \$49E(A5)	As video address v bs ad
005094 13ED044FFFFF8201	move.b	\$44F(A5), \$FFFF8201	Dbaseh, video address for hardware
00509C 13ED0450FFFF8203	move.b	\$450(A5), \$FFFF8203	Dbase1
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)+	
0050AE 20C0	move.l	D0, (A0)+	
0050B0 51C9FFF6	dbra	D1, \$50A8	
0050B4 207AFF5E	move.l	\$5014(PC), A0	Clear screen
0050B8 0C9087654321	cmp.l	#\$87654321, (A0)	Next 16 K bytes
0050BE 6704	beq	\$50C4	
0050C0 41FAFF46	lea	\$5008(PC), A0	
0050C4 23E8004000004FA	move.l	4(A0), \$4FA	End os
0050CC 23E8008000004FE	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 2B7C000060E2047A	move.l	#\$60E2, \$47A(A5)	Hdv boot
0050FC 2B6D044E0436	move.l	\$44E(A5), \$436(A5)	V bs ad as memory top
005102 2B6D04FA0432	move.l	\$4FA(A5), \$432(A5)	End os as memory bottom
005108 4FF900003E2A	lea	\$3E2A, A7	Initialize stack pointer
00510E 3B7C000080454	move.w	#8, \$454(A5)	Nvbls, number of VBL routines
005114 50ED0444	st	\$444(A5)	Fverify, Floppy Verify after write
005118 3B7C000030440	move.w	#3, \$440(A5)	Seekrate for floppy to 3 ms
00511E 2B7C000012BC04C6	move.l	#\$12BC, \$4C6(A5)	Dskbufp to \$12BC, disk buffer
005126 3B7CFFFF04EE	move.w	#\$FFFF, \$4EE(A5)	Clear dumpflg for hardcopy
00512C 2B7C0000500004F2	move.l	#\$5000, \$4F2(A5)	Sysbase, start of the operating system

005134	2B7C000005FC04A2	move.l	#55FC,\$4A2(A5)	Savptr for TRAPs to \$5FC
00513C	2B7C00005328046E	move.l	#55328,\$46E(A5)	Hdv dsb auf rts
005144	47FA03FC	lea	\$5542(PC),A3	Pointer to rte
005148	49FA01DE	lea	\$5328(PC),A4	Pointer to rts
00514C	0CB9FA52235F00FA0000	cmp.l	#FA52235F,\$FA0000	Cartridge, Diagnostic cartridge inserted?
005156	6726	beq	\$517E	Yes, don't initialize vectors
005158	43FA06DE	lea	\$5838(PC),A1	Terminate process
00515C	D3FC02000000	add.l	#2000000,A1	Vector number is in bits 24-31
005162	41F900000008	lea	\$8,A0	Start with bus error
005168	303C003D	move.w	#3D,D0	Number of vectors
00516C	20C9	move.l	A1,(A0)+	Set vector
00516E	D3FC01000000	add.l	#1000000,A1	Increment number in bits 24-31
005174	51C8FFF6	dbra	D0,\$516C	Next vector
005178	23CB00000014	move.l	A3,\$14	'Division by zero' to 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 to rte
005192	2B7C0000556C00B4	move.l	#556C,\$B4(A5)	TRAP #13 vector
00519A	2B7C0000556600B8	move.l	#5566,\$B8(A5)	TRAP #14 vector
0051A2	2B7C0000EB9A0028	move.l	#EB9A,\$28(A5)	LINE A vector
0051AA	2B4C0400	move.l	A4,\$400(A5)	Etv timer to rts
0051AE	2B7C000055620404	move.l	#5562,\$404(A5)	Etv critic vector
0051B6	2B4C0408	move.l	A4,\$408(A5)	Etv term to rts
0051BA	41ED04CE	lea	\$4CE(A5),A0	Vbl list
0051BE	2B480456	move.l	A0,\$456(A5)	As pointer to vblqueue
0051C2	303C0007	move.w	#7,D0	8 vectors
0051C6	4298	clr.l	(A0)+	Clear list
0051C8	51C8FFF6	dbra	D0,\$51C6	Next vector
0051CC	61001D14	bsr	\$6EE2	Initialize MFP
0051D0	7002	moveq.l	#2,D0	Bit number
0051D2	6100012A	bsr	\$52FE	Cart scan, test cartridge
0051D6	9BCD	sub.l	A5,A5	Clear A5

0051D8 6100036A	bsr	\$5544	Wvbl, wait for next picture return
0051DC 61000366	bsr	\$5544	Wvbl, wait for next picture return
0051E0 103C0002	move.b	#2,D0	Default resolution to monochrome
0051E4 08390007FFFFFA01	btst	#7,\$FFFFFA01	Test MFP gpip monochrome detect
0051EC 670C	beq	\$51FA	No monochrome monitor ?
0051EE 102D044A	move.b	\$44A(A5),D0	Get defshiftmd
0051F2 B03C0002	cmp.b	#2,D0	Color mode ?
0051F6 6D02	blt	\$51FA	Yes
0051F8 4200	clr.b	D0	Otherwise select low resolution
0051FA 1B40044C	move.b	D0,\$44C(A5)	Save sshiftmd
0051FE 13C0FFFF8260	move.b	D0,\$FFFF8260	Shiftmd, program shifter
005204 B03C0001	cmp.b	#1,D0	Medium resolution ?
005208 660A	bne	\$5214	No
00520A 33F9FFFF825EFFFF8246	move.w	\$FFFF825E,\$FFFF8246	Copy color 15 (black) to color 3
005214 4EB90000F6C4	jsr	\$F6C4	Initialize screen output
00521A 2B7C0000501E046E	move.l	#501E,\$46E(A5)	Hdv dsb
005222 33FC000100000452	move.w	#1,\$452	Vblsem, free VBL
00522A 4240	clr.w	D0	Bit number 0
00522C 610000D0	bsr	\$52FE	Cartscan, test cartridge
005230 46FC2300	move.w	#2300,SR	IPL to 3
005234 7001	moveq.l	#1,D0	Bit number 1
005236 610000C6	bsr	\$52FE	Cartscan, test cartridge
00523A 61004234	bsr	\$9470	Initialize DOS
00523E 610000B0	bsr	\$52F0	Dskboot
005242 4A7900000482	tst.w	\$482	Cmdload, load shell from disk ?
005248 6718	beq	\$5262	No
00524A 61003C64	bsr	\$8EB0	Enable cursor
00524E 610006FE	bsr	\$594E	Auto exec
005252 487A0099	pea	\$52ED(PC)	Null name
005256 487A0095	pea	\$52ED(PC)	Null name
00525A 487A007E	pea	\$52DA(PC)	'COMMAND.PRG'
00525E 4267	clr.w	-(A7)	

005260 605C	bra	\$52BE	
005262 61006EA	bar	\$594E	
005266 41FA0066	lea	\$52CE(PC),A0	
00526A 327C0502	move.w	#\$502,A1	
00526E 0C100023	cmp.b	#35,(A0)	'#' drive indicator ?
005272 6602	bne	\$5276	No
005274 2449	move.l	A1,A2	Address of the drive indicator
005276 12D8	move.b	(A0)+,(A1)+	Copy drive number
005278 6AF4	bpl	\$526E	Copy entire string
00527A 10390000446	move.b	\$446,D0	Bootdev
005280 D03C0041	add.b	#\$41,D0	'A', calculate drive number
005284 1480	move.b	D0,(A2)	Insert in filename
005286 48790000502	pea	\$502	Environment string
00528C 4879000052ED	pea	\$52ED	Null name
005292 487A0059	pea	\$52ED(PC)	Null name
005296 3F3C0005	move.w	#5,-(A7)	Exec, load program
00529A 3F3C004B	move.w	#\$4B,-(A7)	GEMDOS call
00529E 4E41	trap	#1	Correct stack pointer
0052A0 DEFC000E	add.w	#\$E,A7	Save base page address
0052A4 2040	move.l	D0,A0	Exec os
0052A6 217900004FE0008	move.l	\$4FE,8(A0)	
0052AE 48790000502	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)	
0052C2 4E41	trap	#1	
0052C4 DEFC000E	add.w	#\$E,A7	Exec, load program
0052C8 4EF90000501E	jmp	\$501E	GEMDOS call
0052CE 504154483D00	dc.b	'PATH=',0:1=92	Correct stack
0052D4 233A5C00FF	dc.b	'#:1',0,\$FF	If return, then Reset
0052DA 434F4D414E442E5052	dc.b	'COMMAND.PRG',0	

```

0052E4 4700
0052E6 47454D2E505247      dc.b      'GEN.PRG'

0052ED 000000      dc.b      0,0,0

*****
0052F0 7003      moveq.l #3,D0
0052F2 610A      bsr      $52FE
0052F4 2079000047A      move.l $47A,A0
0052FA 4E90      jsr      (A0)
0052FC 4E75      rts

*****
0052FE 41F900FA0000      lea      $FA0000,A0
005304 0C98ABCDEF42      cmp.l   #ABCDEF42,(A0)+
00530A 661A      bne     $5326
00530C 01280004      btst    D0,4(A0)
005310 670E      beq     $5320
005312 48E7FFFE      movem.l D0-D7/A0-A6,-(A7)
005316 20680004      move.l  4(A0),A0
00531A 4E90      jsr     (A0)
00531C 4CDF7FFF      movem.l (A7)+,D0-D7/A0-A6
005320 4A90      tst.l   (A0)
005322 2050      move.l  (A0),A0
005324 66E6      bne     $530C
005326 4E75      rts

*****
005328 4E75      rts

*****
00532A D1C1      add.l   D1,A0

*****

```

Dskboot, boot from disk  
 Bit number 3  
 Test cartridge  
 Hdv boot, load boot vector  
 And start attempt

Test cartridge  
 Carbase  
 User cartridge inserted ?  
 No  
 Initialization ?  
 No  
 Save registers  
 Address of the init routine  
 Perform initialization  
 Restore registers  
 Additional application in cartridge ?  
 Get link address  
 Initialization next application

Default vector

Memory test  
 Add offset to base address



```

00532C 4240      clr.w    D0          Clear test pattern
00532E 43E801F8  lea      $1F8(A0),A1      End address of the test
005332 B058      cmp.w    (A0)+,D0    Test memory contents
005334 6608      bne      $533E      Unequal, terminate
005336 D07CFA54  add.w    #$FA54,D0        Create next test pattern
00533A B3C8      cmp.l    A0,A1      End address reached?
00533C 66F4      bne      $5332      No, continue test
00533E 4ED5      jmp      (A5)      Back to the calling address

*****
005340 9BCD      sub.l    A5,A5          Memvalid
005342 0CAD752019F30420  cmp.l    #1965038067,$420(A5)
00534A 6608      bne      $5354          Memval2
00534C 0CAD237698AA043A  cmp.l    #594974890,$43A(A5)
005354 4ED6      jmp      (A6)

*****
005356 00000000
00535A 00000000
00535E 00000000
005362 00000000

*****
005366 1839FFFFF8260  move.b    $FFFFF8260,D4      Shiftmd, Get video resolution
00536C C87C0003  and.w    #3,D4             Isolate bits 0 and 1
005370 D844      add.w    D4,D4             Double value for word table
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	Dbase1
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

*****
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)
0053FC E352      roxl.w #1, D2
0053FE 46DF      move.w (A7)+, SR
005400 E352      roxl.w #1, D2
005402 51CBFFF4  dbra   D3, $53F8
005406 260B      move.l A3, D3
005408 4ED5      jmp     (A5)

*****
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

```

High resolution

Save carry flag

Restore carry flag

Back to call

```

005414 0050          dc.w      80
005416 0000          dc.w      0
005418 0000          dc.w      0
00541A 0001          dc.w      0

*****
00541C 0777          dc.w      $777
00541E 0700          dc.w      $700
005420 0070          dc.w      $070
005422 0770          dc.w      $770
005424 0007          dc.w      $007
005426 0707          dc.w      $707
005428 0077          dc.w      $077
00542A 0555          dc.w      $555
00542C 0333          dc.w      $333
00542E 0733          dc.w      $733
005430 0373          dc.w      $373
005432 0773          dc.w      $773
005434 0337          dc.w      $337
005436 0737          dc.w      $737
005438 0377          dc.w      $377
00543A 0000          dc.w      $000

*****
00543C 3F00          move.w    D0,-(A7)
00543E 302F0002      move.w    2(A7),D0
005442 C07C0700      and.w      #$700,D0
005446 6606          bne        $544E
005448 006F03000002  or.w      #$300,2(A7)
00544E 301F          move.w    (A7)+,D0
005450 4E73          rte

*****
                                HBL interrupt
                                Save D0
                                Get status from stack
                                Isolate IPL mask
                                Not equal to zero ?
                                Otherwise IPL tp 3
                                Restore D0

```

```

*****
005452 52B900000466      addq.l #1,$466
005458 537900000452      subq.w #1,$452
00545E 6B0000DC          bml      $553C
005462 48E7FFFE          movem.l D0-D7/A0-A6,-(A7)
005466 52B900000462      addq.l #1,$462
00546C 9BCD             sub.l  A5,A5
00546E 1039FFFF8260      move.b $FFFF8260,D0
005474 C03C0003          and.b  #3,D0
005478 B03C0002          cmp.b  #2,D0
00547C 6C10             bge     $548E
00547E 08390007FFFFFFA01 btst   #7,$FFFFFFA01
005486 662C             bne     $54B4
005488 103C0002          move.b #2,D0
00548C 6016             bra     $54A4
00548E 08390007FFFFFFA01 btst   #7,$FFFFFFA01
005496 671C             beq     $54B4
005498 102D044A          move.b $44A(A5),D0
00549C B03C0002          cmp.b  #2,D0
0054A0 6D02             blt     $54A4
0054A2 4200             clr.b  D0
0054A4 1B40044C          move.b D0,$44C(A5)
0054A8 13C0FFFF8260      move.b D0,$FFFF8260
0054AE 206D046E          move.l $46E(A5),A0
0054B2 4E90             jsr     (A0)
0054B4 61003B42          bsr     $8FF8
0054B8 9BCD             sub.l  A5,A5
0054BA 4AAD045A          tst.l  $45A(A5)
0054BE 6718             beq     $54D8
0054C0 206D045A          move.l $45A(A5),A0
0054C4 43F9FFFF8240      lea     $FFFF8240,A1
0054CA 303C000F          move.w #$F,D0
*****
VBL interrupt
Increment frclock
Vblsem
VBL interrupt disabled ?
Save registers
Increment vbclock exh\hen
Clear A5
Load shiftmd
High resolution ?
Yes
Mfp gpip, monochrome detect
No color monitor
High resolution
Mfp gpip, monochrome detect
Monochrome monitor ?
Defshiftmd, get color resolution
Monochrome ?
No
Else low resolution
Save sshiftmd
Shiftmd, program shifter
Vector for resolution change
Execute routine
Flash cursor
Clear A5
Colorptr, reload color palette ?
No
Colorptr, address of the new palette
Color0, address in the video shifter
16 colors

```

0054CE 32D8	move.w (A0)+, (A1)+	Copy
0054D0 51C8FFFC	dbra D0, \$54CE	Next color
0054D4 42AD045A	clr.l \$A5(A5)	Clear colorptr again
0054D8 4AAD045E	tst.l \$A5(A5)	Screenpt, new video address ?
0054DC 671A	beq \$54F8	No
0054DE 2B6D045E044E	move.l \$A5(A5), \$A4E(A5)	Copy screenpt tp v bs ad
0054E4 202D044E	move.l \$A4E(A5), D0	V bs ad
0054E8 E048	lsl.w #8, D0	Bits 8-15
0054EA 13C0FFFFF8203	move.b D0, \$FFFFF8203	To dbasel
0054F0 E048	lsl.w #8, D0	Bits 16-24
0054F2 13C0FFFFF8201	move.b D0, \$FFFFF8201	To dbaseh
0054F8 610011EA	bsr \$66E4	VBL routine for floppies
0054FC 3E39000000454	move.w \$A54, D7	Nvbis, number of VBL routines
005502 6720	beq \$5524	No vectors ?
005504 5387	subq.l #1, D7	Convert to dbra counter
005506 2079000000456	move.l \$A56, A0	Address of the vblqueu
00550C 2258	move.l (A0)+, A1	Get vector
00550E B3FC000000000	cmp.l #0, A1	Zero ?
005514 670A	beq \$5520	Don't execute routine
005516 48E70180	movem.l D7/A0, -(A7)	Save registers
00551A 4E91	jsr (A1)	Execute routine
00551C 4CDF0180	movem.l (A7)+, D7/A0	Restore registers
005520 51CFFFEA	dbra D7, \$550C	Test next vector
005524 9BCD	sub.l A5, A5	Clear A5
005526 4A6D04EE	tst.w \$AEE(A5)	Dumpflg, hardcopy desired ?
00552A 660C	bne \$5538	No
00552C 61000532	bsr \$5A60	Hardcopy
005530 33FCFFFF000004EE	move.w \$FFFF, \$AEE	Clear dumpflg again
005538 4CDF7FFF	movem.l (A7)+, D0-D7/A0-A6	Restore registers
00553C 5279000000452	addq.w #1, \$A52	Vblsem, restore VBL again
005542 4E73	rte	

```

*****
005544 40E7      move.w  SR, -(A7)
005546 027CF8FF   and.w   #0F8FF,SR
00554A 20390000466  move.l  $466,D0
005550 B0B90000466  cmp.l   $466,D0
005556 67F8      beq     $5550
005558 46DF      move.w  (A7)+,SR
00555A 4E75      rts

*****
00555C 2F390000404  move.l  $404,-(A7)
005562 70FF      moveq.l  #-1,D0
005564 4E75      rts

*****
005566 41FA0084     lea     $55EC(PC),A0
00556A 6004      bra     $5570

*****
00556C 41FA004C     lea     $55BA(PC),A0
005570 227900004A2  move.l  $4A2,A1
005576 301F      move.w  (A7)+,D0
005578 3300      move.w  D0,-(A1)
00557A 231F      move.l  (A7)+,-(A1)
00557C 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
005580 23C900004A2  move.l  A1,$4A2
005586 0800000D    btst    #13,D0
00558A 6602      bne     $558E
00558C 4E6F      move.l  USP,A7
00558E 301F      move.w  (A7)+,D0
005590 B058      cmp.w  (A0)+,D0
005592 6C10      bge     $55A4

*****
005594 6004      bra     $5570

*****
005596 41FA0084     lea     $55EC(PC),A0
00559A 6004      bra     $5570

*****
00559C 41FA004C     lea     $55BA(PC),A0
0055A0 227900004A2  move.l  $4A2,A1
0055A6 301F      move.w  (A7)+,D0
0055AC 3300      move.w  D0,-(A1)
0055AE 231F      move.l  (A7)+,-(A1)
0055B0 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
0055B6 23C900004A2  move.l  A1,$4A2
0055BC 0800000D    btst    #13,D0
0055C0 6602      bne     $558E
0055C2 4E6F      move.l  USP,A7
0055C4 301F      move.w  (A7)+,D0
0055CA B058      cmp.w  (A0)+,D0
0055CE 6C10      bge     $55A4

*****
0055D0 6004      bra     $5570

*****
0055D2 41FA0084     lea     $55EC(PC),A0
0055D6 6004      bra     $5570

*****
0055D8 41FA004C     lea     $55BA(PC),A0
0055DC 227900004A2  move.l  $4A2,A1
0055E2 301F      move.w  (A7)+,D0
0055E8 3300      move.w  D0,-(A1)
0055EA 231F      move.l  (A7)+,-(A1)
0055EC 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
0055F0 23C900004A2  move.l  A1,$4A2
0055F6 0800000D    btst    #13,D0
0055FA 6602      bne     $558E
0055FC 4E6F      move.l  USP,A7
0055FE 301F      move.w  (A7)+,D0
005600 B058      cmp.w  (A0)+,D0
005602 6C10      bge     $55A4

*****
005604 6004      bra     $5570

*****
005606 41FA0084     lea     $55EC(PC),A0
00560A 6004      bra     $5570

*****
00560C 41FA004C     lea     $55BA(PC),A0
005610 227900004A2  move.l  $4A2,A1
005616 301F      move.w  (A7)+,D0
00561C 3300      move.w  D0,-(A1)
00561E 231F      move.l  (A7)+,-(A1)
005620 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
005626 23C900004A2  move.l  A1,$4A2
00562C 0800000D    btst    #13,D0
005630 6602      bne     $558E
005632 4E6F      move.l  USP,A7
005634 301F      move.w  (A7)+,D0
00563A B058      cmp.w  (A0)+,D0
00563C 6C10      bge     $55A4

*****
00563E 6004      bra     $5570

*****
005640 41FA0084     lea     $55EC(PC),A0
005644 6004      bra     $5570

*****
005646 41FA004C     lea     $55BA(PC),A0
00564A 227900004A2  move.l  $4A2,A1
00564E 301F      move.w  (A7)+,D0
005654 3300      move.w  D0,-(A1)
005656 231F      move.l  (A7)+,-(A1)
005658 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
00565E 23C900004A2  move.l  A1,$4A2
005664 0800000D    btst    #13,D0
00566A 6602      bne     $558E
00566C 4E6F      move.l  USP,A7
00566E 301F      move.w  (A7)+,D0
005674 B058      cmp.w  (A0)+,D0
005676 6C10      bge     $55A4

*****
005678 6004      bra     $5570

*****
00567A 41FA0084     lea     $55EC(PC),A0
00567E 6004      bra     $5570

*****
005680 41FA004C     lea     $55BA(PC),A0
005684 227900004A2  move.l  $4A2,A1
00568A 301F      move.w  (A7)+,D0
005690 3300      move.w  D0,-(A1)
005692 231F      move.l  (A7)+,-(A1)
005694 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
00569A 23C900004A2  move.l  A1,$4A2
0056A0 0800000D    btst    #13,D0
0056A6 6602      bne     $558E
0056AC 4E6F      move.l  USP,A7
0056AE 301F      move.w  (A7)+,D0
0056B4 B058      cmp.w  (A0)+,D0
0056B6 6C10      bge     $55A4

*****
0056B8 6004      bra     $5570

*****
0056BA 41FA0084     lea     $55EC(PC),A0
0056BE 6004      bra     $5570

*****
0056C0 41FA004C     lea     $55BA(PC),A0
0056C4 227900004A2  move.l  $4A2,A1
0056CA 301F      move.w  (A7)+,D0
0056D0 3300      move.w  D0,-(A1)
0056D2 231F      move.l  (A7)+,-(A1)
0056D4 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
0056DA 23C900004A2  move.l  A1,$4A2
0056E0 0800000D    btst    #13,D0
0056E6 6602      bne     $558E
0056EC 4E6F      move.l  USP,A7
0056EE 301F      move.w  (A7)+,D0
0056F4 B058      cmp.w  (A0)+,D0
0056F6 6C10      bge     $55A4

*****
0056F8 6004      bra     $5570

*****
0056FA 41FA0084     lea     $55EC(PC),A0
0056FE 6004      bra     $5570

*****
005600 41FA004C     lea     $55BA(PC),A0
005604 227900004A2  move.l  $4A2,A1
00560A 301F      move.w  (A7)+,D0
00560E 3300      move.w  D0,-(A1)
005612 231F      move.l  (A7)+,-(A1)
005614 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
00561A 23C900004A2  move.l  A1,$4A2
00561E 0800000D    btst    #13,D0
005624 6602      bne     $558E
00562A 4E6F      move.l  USP,A7
00562E 301F      move.w  (A7)+,D0
005634 B058      cmp.w  (A0)+,D0
005636 6C10      bge     $55A4

*****
005638 6004      bra     $5570

*****
00563A 41FA0084     lea     $55EC(PC),A0
00563E 6004      bra     $5570

*****
005640 41FA004C     lea     $55BA(PC),A0
005644 227900004A2  move.l  $4A2,A1
00564A 301F      move.w  (A7)+,D0
00564E 3300      move.w  D0,-(A1)
005652 231F      move.l  (A7)+,-(A1)
005654 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
00565A 23C900004A2  move.l  A1,$4A2
00565E 0800000D    btst    #13,D0
005664 6602      bne     $558E
00566A 4E6F      move.l  USP,A7
00566E 301F      move.w  (A7)+,D0
005674 B058      cmp.w  (A0)+,D0
005676 6C10      bge     $55A4

*****
005678 6004      bra     $5570

*****
00567A 41FA0084     lea     $55EC(PC),A0
00567E 6004      bra     $5570

*****
005680 41FA004C     lea     $55BA(PC),A0
005684 227900004A2  move.l  $4A2,A1
00568A 301F      move.w  (A7)+,D0
00568E 3300      move.w  D0,-(A1)
005692 231F      move.l  (A7)+,-(A1)
005694 48E11F1F    movem.l  D3-D7/A3-A7,-(A1)
00569A 23C900004A2  move.l  A1,$4A2
00569E 0800000D    btst    #13,D0
0056A4 6602      bne     $558E
0056AA 4E6F      move.l  USP,A7
0056AE 301F      move.w  (A7)+,D0
0056B4 B05
```

```

005594 E548      lsl.w      #2,D0
005596 20300000  move.l      0(A0,D0.w),D0
00559A 2040      move.l      D0,A0
00559C 6A02      bpl        $55A0
00559E 2050      move.l      (A0),A0
0055A0 9BCD      sub.l      A5,A5
0055A2 4E90      jsr        (A0)
0055A4 227900004A2  move.l      $4A2,A1
0055AA 4CD9F8F8  movem.l      (A1)+,D3-D7/A3-A7
0055AE 2F19      move.l      (A1)+,-(A7)
0055B0 3F19      move.w      (A1)+,-(A7)
0055B2 23C900004A2  move.l      A1,$4A2
0055B8 4E73      rte

*****
0055BA 000C      dc.w      12
0055BC 0000572E  dc.l      $572E
0055C0 00005694  dc.l      $5694
0055C4 0000569A  dc.l      $569A
0055C8 000056A6  dc.l      $56A6
0055CC 80000476  dc.l      $476+$80000000
0055D0 0000575A  dc.l      $575A
0055D4 00005772  dc.l      $5772
0055D8 80000472  dc.l      $472+$80000000
0055DC 000056A0  dc.l      $56A0
0055E0 8000047E  dc.l      $47E+$80000000
0055E4 00005716  dc.l      $5716
0055E8 0000571C  dc.l      $571C

```

Convert number of long counter  
Get address of the routine  
To A0  
Bit 31 cleared ?  
Else use address indirectly  
Clear A5  
Execute routine  
Get savptr  
Restore registers  
PC on stack  
Status on stack  
Update savptr  
Back to call

Addresses of the TRAP #13 calls  
Number of routines  
0, getmpb  
1, bconstat  
2, bconin  
3, bconout  
4, (indirect) rwabs  
5, setexec  
6, tickcal  
7, (indirect) getbpb  
8, bcostat  
9, (indirect) mediach  
10, drvmap  
11, shift



```

*****
0055EC 0028
0055EE 00007AC0
0055F2 00005328
0055F6 0000577A
0055FA 0000578E
0055FE 00005794
005602 000057A0
005606 000057EE
00560A 000057F6
00560E 000062D2
005612 000063B0
005616 00006468
00561A 00005B64
00561E 00006B74
005622 00007138
005626 00007440
00562A 00007468
00562E 00007BC6
005632 00006062
005636 0000614A
00563A 00006602
00563E 00005A4E
005642 00009024
005646 00006AAA
00564A 00006A90
00564E 00007BF2
005652 00006CE6
005656 00007162
00565A 0000719C
00565E 00007A30
005662 00007A9A

*****
dc.w 40
dc.l $7AC0
dc.l $5328
dc.l $577A
dc.l $578E
dc.l $5794
dc.l $57A0
dc.l $57EE
dc.l $57F6
dc.l $62D2
dc.l $63B0
dc.l $6468
dc.l $5B64
dc.l $6B74
dc.l $7138
dc.l $7440
dc.l $7468
dc.l $7BC6
dc.l $6062
dc.l $614A
dc.l $6602
dc.l $5A4E
dc.l $9024
dc.l $6AAA
dc.l $6A90
dc.l $7BF2
dc.l $6CE6
dc.l $7162
dc.l $719C
dc.l $7A30
dc.l $7A9A

*****
Addresses of the TRAP #14 calls
Number of routines
0, initmouse
1, rts
2, physbase
3, logbase
4, getrez
5, setscreen
6, setpalette
7, setcolor
8, floprd
9, floprw
10, flopfmt
11, getdsb
12, midlws
13, mfprint
14, lorec
15, rsconf
16, keytrans
17, rand
18, protobt
19, flopver
20, dumpit
21, cursconf
22, setttime
23, gettime
24, bioskeys
25, ikbdws
26, jdisint
27, jenabint
28, giaccess
29, offgibit

```

```

005666 00007A74      dc.l  $7A74      30, onglbit
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

*****
00568E 206F0004      move.l 4(A7),A0      supexec, routine in supervisor mode
005692 4ED0           jmp      (A0)          Get address from stack
                               Execute routine in supervisor mode

*****
005694 41FA0020      lea    $56B6(PC),A0      beaconstat, get input statis
005698 6010           bra    $56AA      Status table

*****
00569A 41FA0032      lea    $56CE(PC),A0      beaconin, input
00569E 600A           bra    $56AA      Input table

*****
0056A0 41FA0044      lea    $56E6(PC),A0      beaconstat, get output status
0056A4 6004           bra    $56AA      Status table

*****
0056A6 41FA0056      lea    $56FE(PC),A0      beaconout, output

```

```

*****
0056AA 302F0004          move.w 4(A7),D0
0056AE E548             lsl.w #2,D0
0056B0 20700000          move.l 0(A0,D0.w),A0
0056B4 4ED0             jmp      (A0)
*****
0056B6 00005328          dc.l  $5328
0056BA 00006C70          dc.l  $6C70
0056BE 00006CFA          dc.l  $6CFA
0056C2 00006B88          dc.l  $6B88
0056C6 00005328          dc.l  $5328
0056CA 00005328          dc.l  $5328
*****
0056CE 00006C3C          dc.l  $6C3C
0056D2 00006C86          dc.l  $6C86
0056D6 00006D10          dc.l  $6D10
0056DA 00006BA4          dc.l  $6BA4
0056DE 00005328          dc.l  $5328
0056E2 00005328          dc.l  $5328
*****
0056E6 00006C5C          dc.l  $6C5C
0056EA 00006C96          dc.l  $6C96
0056EE 00006D46          dc.l  $6D46
0056F2 00006CBA          dc.l  $6CBA
0056F6 00006B48          dc.l  $6B48
0056FA 00005328          dc.l  $5328
*****
Conout
Get device number
Times 4
Get address of the routine
Execute routine

Input status
Rts
RS 232 status
Console status
MIDI status
RTS
RTS

Input
Parallel port
RS 232 input
Console input
MIDI input
RTS
RTS

Output status
Centronics status
RS 232 status
Console status
Keyboard status
MIDI status
rts

```

```

*****
0056FE 00006BD4      dc.l  $6BD4
005702 00006CAE      dc.l  $6CAE
005706 00008ADE      dc.l  $8ADE
00570A 00006B5A      dc.l  $6B5A
00570E 00006CCC      dc.l  $6CCC
005712 00008AD2      dc.l  $8AD2

*****
005716 202D04C2      move.l  $4C2(A5),D0
00571A 4E75          rts

*****
00571C 7000          moveq.l  #0,D0
00571E 102D0A5D      move.b  $A5D(A5),D0
005722 322F0004      move.w  4(A7),D1
005726 6B04          bmi  $572C
005728 1B410A5D      move.b  D1,$A5D(A5)
00572C 4E75          rts

*****
00572E 206F0004      move.l  4(A7),A0
005732 43ED048E      lea  $48E(A5),A1
005736 2089          move.l  A1,(A0)
005738 42A80004      clr.l  4(A0)
00573C 21490008      move.l  A1,8(A0)
005740 4291          clr.l  (A1)
005742 236D04320004  move.l  $432(A5),4(A1)
005748 202D0436      move.l  $436(A5),D0
00574C 90AD0432      sub.l  $432(A5),D0
005750 23400008      move.l  D0,8(A1)
005754 42A9000C      clr.l  12(A1)

*****
Getmpb, Memory Parameter Block
New MPB
Themd, Memory descriptor
Mp mfl = address of the MD
Mp mal = zero
Mp rovr = address of the MD
Clear m link
Membot as mstart
Memtop
Minus membot
Length as m length
M own = zero

```

```

005758 4E75      rts

*****
00575A 302F0004      move.w 4(A7),D0
00575E E548      lsl.w #2,D0
005760 91C8      sub.l A0,A0
005762 41F00000      lea 0(A0,D0.w),A0
005766 2010      move.l (A0),D0
005768 222F0006      move.l 6(A7),D1
00576C 6B02      bml $5770
00576E 2081      move.l D1,(A0)
005770 4E75      rts

*****
005772 4280      clr.l D0
005774 302D0442      move.w $442(A5),D0
005778 4E75      rts

*****
00577A 7000      moveq.l #0,D0
00577C 1039FFFF8201      move.b $FFFF8201,D0
005782 E148      lsl.w #8,D0
005784 1039FFFF8203      move.b $FFFF8203,D0
00578A E188      lsl.l #8,D0
00578C 4E75      rts

*****
00578E 202D044E      move.l $44E(A5),D0
005792 4E75      rts

*****
005794 7000      moveq.l #0,D0

```

Setexc, set exception vector  
Get vector number  
Times 4 equals address  
Clear A0  
Get address of the old vector  
Old vector to D0  
New vector  
Negative, then don't set  
Set vector

Tickcal, timer value in milliseconds

Get timr ms

Physbase, physical video address

Dbaseh  
Dbase1  
Video address in D0

Logbase, logical video address  
Get v bs ad

Getrez, get video resolution

```

005796 102D8260      move.b  $FFFFF8260(A5),D0
00579A C03C0003      and.b   #3,D0
00579E 4E75          rts

*****
0057A0 4AAF0004      tst.l   4(A7)
0057A4 6B06          bmi    $57AC
0057A6 2B6F004044E  move.l  4(A7),$44E(A5)
0057AC 4AAF0008      tst.l   8(A7)
0057B0 6B10          bmi    $57C2
0057B2 13EF009FFFF8201 move.b  9(A7),$FFFF8201
0057BA 13EF000AFFFF8203 move.b  10(A7),$FFFF8203
0057C2 4A6F000C      tst.w   12(A7)
0057C6 6B24          bmi    $57EC
0057C8 1B6F00D044C  move.b  13(A7),$44C(A5)
0057CE 6100FD74      bsr    $5544
0057D2 13ED044CFFFF8260 move.b  $44C(A5),$FFFF8260
0057DA 426D0452      clr.w   $452(A5)
0057DE 4EB90000F6C4  jsr    $F6C4
0057E4 33FC00010000452 move.w  #1,$452
0057EC 4E75          rts

*****
0057EE 2B6F004045A  move.l  4(A7),$45A(A5)
0057F4 4E75          rts

*****
0057F6 322F0004      move.w  4(A7),D1
0057FA D241          add.w   D1,D1
0057FC C27C001F      and.w   #1F,D1
005800 41F9FFFF8240  lea    $FFFF8240,A0
005806 30301000      move.w  0(A0,D1.w),D0

Load shiftmd
Isolate bits 0 and 1

Setscreen, set screen address
Logical address
Negative, don't set
Set v bs ad
Physical address
Negative, don't set
Dbaseh
Dbasel
Video resolution
Negative, don't set
Shiftmd
Wvbl, wait for VBL
Shiftmd to shiftmd
Vblsem, VBL disabled
Initialize screen output
Vblsem, permit VBL again

Setpalette, load new color palette
Set colorpr (execution in VBL)

Setcolor, set single color
Color number
Times 2
Limit to valid number
Address color0
Get old color

```

```

00580A C07C0777      and.w    #$777,D0
00580E 4A6F0006      tst.w    6(A7)
005812 6B06          bmi     $581A
005814 31AF0061000   move.w    6(A7),0(A0,D1.w)
00581A 4E75          rts

*****
00581C 207AF7F6      move.l    $5014(PC),A0
005820 0C9087654321  cmp.l    #$87654321,(A0)
005826 660E          bne     $5836
005828 B1F90000042E  cmp.l    $42E,A0
00582E 6C06          bge     $5836
005830 4290          clr.l    (A0)
005832 6000F7EA      bra     $501E
005836 4E75          rts

*****
005838 6102          bsr     $583C
00583A 4E71          nop
00583C 23DF000003C4   move.l    (A7)+,$3C4
005842 48F9FFFF0000384  movem.l  D0-D7/A0-A7,$384
00584A 4E68          move.l    USP,A0
00584C 23C8000003C8   move.l    A0,$3C8
005852 303C000F      move.w    #SP,D0
005856 41F9000003CC   lea     $3CC,A0
00585C 224F          move.l    A7,A1
00585E 30D9          move.w    (A1)+,(A0)+
005860 51C8FFFC      dbra     D0,$585E
005864 23FC123456780000380  move.l    #$12345678,$380
00586E 7200          moveq.l  #0,D1
005870 1239000003C4   move.b    $3C4,D1
005876 5341          subq.w    #1,D1

*****
Term, interrupt running program
PC on stack

Save PC including vector number
Save registers
USP
Save
16 words
Save area
Get SP to A1
Save word from stack
Next word
Magic for saved registers

Get vector number to D1
Convert in dbra counter

```

```

005878 6116      bsr      $5890
00587A 23FC00005FC000004A2 move.l #55FC,$A2
005884 3F3C0001      move.w #1,-(A7)
005888 42A7      clr.l  -(A7)
00588A 4E41      trap   #1
00588C 6000F790      bra   $501E

*****
005890 1E39FFFF8260      move.b $FFFF8260,D7
005896 CE7C0003      and.w  #3,D7
00589A DE47      add.w  D7,D7
00589C 4280      clr.l  D0
00589E 1039FFFF8201      move.b $FFFF8201,D0
0058A4 E148      lsl.w  #8,D0
0058A6 1039FFFF8203      move.b $FFFF8203,D0
0058AC E188      lsl.l  #8,D0
0058AE 2040      move.l  D0,A0
0058B0 D0FB702A      add.w  $58DC(PC,D7.w),A0
0058B4 43FA0038      lea    $58EE(PC),A1
0058B8 3C3C000F      move.w #5F,D6
0058BC 3401      move.w  D1,D2
0058BE 2448      move.l  A0,A2
0058C0 3A3B7020      move.w $58E2(PC,D7.w),D5
0058C4      move.w  (A1),(A0)+
0058C6 51CDFFFFC      dbra   D5,$58C4
0058CA 51CAFFFF4      dbra   D2,$58C0
0058CE 5449      addq.w #2,A1
0058D0 D4FB7016      add.w  $58E8(PC,D7.w),A2
0058D4 204A      move.l  A2,A0
0058D6 51CEFFE4      dbra   D6,$58BC
0058DA 4E75      rts

Output appropriate # of 'mushrooms'
Reset savptr for BIOS
Return code for error
Terminate process
GEMDOS call
If return, then reset

Write 'mushrooms' on screen
Shiftmd, get resolution
Isolate significant bits
Times 2 for word access

Dbassh
Dbasel
Screen address
To A0
Plus offset for middle of screen
Address of bit pattern for "mushroom"
16 raster lines

Save pointer to start of line
Number of words (screen planes)
Write a raster line
A complete mushroom
The next on the same line
Next word of the bit pattern
Next destination address
Restore start of the line
Next raster line

```



```

*****
0058DC 3E80          dc.w    100*160
0058DE 3E80          dc.w    100*160
0058E0 3E80          dc.w    200*80
*****

*****
0058E2 0003          dc.w     3
0058E4 0001          dc.w     1
0058E6 0000          dc.w     0
*****

*****
0058E8 00A0          dc.w    160
0058EA 00A0          dc.w    160
0058EC 0050          dc.w     80
*****

*****
0058EE 07C0          dc.w    $000001111000000
0058F0 1FF0          dc.w    $000111111110000
0058F2 3BF8          dc.w    $001101111111000
0058F4 77F4          dc.w    $0110111111110100
0058F6 B7FA          dc.w    $1011011111111010
0058F8 BBFA          dc.w    $10111011111111010
0058FA DFF6          dc.w    $11011111111110110
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
*****

Screen center
Low resolution
Mid resolution
High resolution

Number of screen planes -1
Low resolution
Mid resolution
High resolution

Line length
Low resolution
Mid resolution
High resolution

Bit pattern 'mushrooms'

```

```

00590E 1290      dc.w      %0001001010010000

*****
005910 206F0004      move.l  4(A7),A0
005914 226F0008      move.l  8(A7),A1
005918 303C003F      move.w   #3F,D0
00591C 12D8      move.b   (A0)+,(A1)+
00591E 12D8      move.b   (A0)+,(A1)+
005920 12D8      move.b   (A0)+,(A1)+
005922 12D8      move.b   (A0)+,(A1)+
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
005930 4E75      rts

*****
005932 2F390000046A      move.l  $46A,-(A7)
005938 4E75      rts

*****
00593A 5C4155544F5C      dc.b    '1AUT01'
005940 2A2E50524700      dc.b    '*.PRG',0
005946 123456789ABCDEFO      dc.l    $12345678, $9ABCDEFO

*****
00594E 41FAFFEA      lea     $543(PC),A0
005952 43FAFFEC      lea     $5940(PC),A1
005956 23DF000005FC      move.l  (A7)+,$5FC
00595C 9BCD      sub.l    A5,A5
00595E 2B480600      move.l  A0,$600(A5)
005962 2B490604      move.l  A1,$604(A5)

```

Fastcopy, copy disk sector  
Source address  
Destination address  
(63+1)\*8 = 512 bytes

Copy 8 bytes

Next 8 bytes

Hard disk initialization  
Hdv init on stack  
Jump to routine

Auto, start and execute a program  
Address of pathname  
Address of filename '\*.PRG'  
Return address  
Clear A5  
Pathname  
Filename

```

005966 202D04C2          move.l  $4C2(A5),D0
00596A 323900000446      move.w  $446,D1
005970 0300              btst     D1,D0
005972 6736              beq      $59AA
005974 41FAF977          lea      $52ED(PC),A0
005978 2F08              move.l  A0,-(A7)
00597A 2F08              move.l  A0,-(A7)
00597C 2F08              move.l  A0,-(A7)
00597E 3F3C0005          move.w  #5,-(A7)
005982 3F3C004B          move.w  #$4B,-(A7)
005986 4E41              trap     #1
005988 DEFC0010          add.w   #$10,A7
00598C 2040              move.l  D0,A0
00598E 217C000059AC0008 move.l  #$59AC,8(A0)
005996 2F0B              move.l  A3,-(A7)
005998 2F00              move.l  D0,-(A7)
00599A 2F0B              move.l  A3,-(A7)
00599C 3F3C0004          move.w  #$4,-(A7)
0059A0 3F3C004B          move.w  #$4B,-(A7)
0059A4 4E41              trap     #1
0059A6 DEFC0010          add.w   #$10,A7
0059AA 4E75              rts

*****
0059AC 42A7              clr.l   -(A7)
0059AE 3F3C0020          move.w  #$20,-(A7)
0059B2 4E41              trap     #1
0059B4 5C4F              addq.w  #6,A7
0059B6 2840              move.l  D0,A4
0059B8 2A6F0004          move.l  4(A7),A5
0059BC 4FED0100          lea     $100(A5),A7
0059C0 2F3C00000100      move.l  #$100,-(A7)

Drvbits, vector with active drives
Bootdev
Drive active ?
No, done
Pointer to null name
Environment
Command tail
Shell name
Create PSP
Exec, load program
GEMDOS call
Correct stack
PSP
PC for auto exec program
Null string
PSP
Shell name
Exec, load program
GEMDOS call
Correct stack pointer

Start auto exec program
super, enter supervisor mode
GEMDOS call
Correct stack pointer
Saved stack pointer
Base page address
Space for base page
$100 bytes

```

0059C6 2F0D	move.l	A5,-(A7)	Start of the memory area
0059C8 4267	clr.w	-(A7)	
0059CA 3F3C004A	move.w	#\$4A,-(A7)	Setblock
0059CE 4E41	trap	#1	GEMDOS call
0059D0 5C4F	addq.w	#6,A7	Correct stack pointer
0059D2 4A40	tst.w	D0	Error ?
0059D4 666A	bne	\$5A40	Yes
0059D6 3F3C0007	move.w	#7,-(A7)	R/O, hidden and system files
0059DA 2F3900000600	move.l	\$600,-(A7)	Filename
0059E0 3F3C004E	move.w	#\$4E,-(A7)	Search first
0059E4 7E08	moveq.l	#8,D7	Bytes for stack correction
0059E6 487900000608	pea	\$608	DNA address for DOS
0059EC 3F3C001A	move.w	#\$1A,-(A7)	Setdta
0059F0 4E41	trap	#1	GEMDOS call
0059F2 5C4F	addq.w	#6,A7	Correct stack pointer
0059F4 4E41	trap	#1	GEMDOS call
0059F6 DEC7	add.w	D7,A7	Correct stack pointer
0059F8 4A40	tst.w	D0	File found ?
0059FA 6644	bne	\$5A40	No
0059FC 207900000600	move.l	\$600,A0	Pathname
005A02 247900000604	move.l	\$604,A2	Filename
005A08 43F900000634	lea	\$634,A1	Auto name
005A0E 12D8	move.b	(A0)+(A1)+	Copy path part
005A10 B5C8	cmp.l	A0,A2	
005A12 66FA	bne	\$5A0E	
005A14 41F900000626	lea	\$626,A0	Name from DMA buffer
005A1A 12D8	move.b	(A0)+(A1)+	Append to path
005A1C 66FC	bne	\$5A1A	
005A1E 487AF8CD	pea	\$52ED(PC)	Null name
005A22 487AF8C9	pea	\$52ED(PC)	Null name
005A26 487900000634	pea	\$634	Filename
005A2C 4267	clr.w	-(A7)	Load and start program

```

005A2E 3F3C004B      move.w  #54B,-(A7)
005A32 4E41          trap      #1
005A34 DEFC0010      add.w   #510,A7
005A38 7E02          moveq.l #2,D7
005A3A 3F3C004F      move.w  #54F,-(A7)
005A3E 60A6          bra      $59E6
005A40 4FF900003E2A  lea      $3E2A,A7
005A46 2F39000005FC  move.l   $5FC,-(A7)
005A4C 4E75          rts

*****
005A4E 4279000004EE  clr.w   $4EE
005A54 610A          bsr      $5A60
005A56 33FCFFFF00004EE  move.w  #-1,$4EE
005A5E 4E75          rts

*****
005A60 9BCD          sub.l   A5,A5
005A62 2B6D044E0654  move.l   $44E(A5),$654(A5)
005A68 426D0658      clr.w   $658(A5)
005A6C 4240          clr.w   D0
005A6E 102D044C      move.b   $44C(A5),D0
005A72 3B400662      move.w  D0,$662(A5)
005A76 D040          add.w   D0,D0
005A78 41FA005A      lea      $5AD4(PC),A0
005A7C 3B700000065A  move.w  0(A0,D0.w),$65A(A5)
005A82 3B700000065C  move.w  6(A0,D0.w),$65C(A5)
005A88 426D065E      clr.w   $65E(A5)
005A8C 426D0660      clr.w   $660(A5)
005A90 2B7C00FF82400666  move.l   #FF8240,$666(A5)
005A98 426D066E      clr.w   $66E(A5)
005A9C 322D0A8C      move.w  $A8C(A5),D1

```

Exec, load program  
GEMDOS call  
Correct stack pointer  
Bytes for stack pointer  
Search next  
Next file  
Stack pointer back to start value  
Load return address

Dumpit, screen hardcopy  
Set dumpflg  
Hardcopy  
Clear dumpflg

Scrdmp, hardcopy  
Clear A5  
v bs ad, screen output  
Offset to Null  
Sshiftmd, screen resolution  
Save  
Times 2  
Table for screen resolution  
Get screen width  
Get screen height  
Left  
And right to zero  
Address to color palette  
Clear mask pointer  
Get printer configuration

```

005AA0 E649      lsr.w      #3,D1
005AA2 C27C0001  and.w      #1,D1
005AA6 3B410664  move.w      D1,$664(A5)
005AAA 322D0A8C  move.w      $A8C(A5),D1
005AAE 3001      move.w      D1,D0
005AB0 E848      lsr.w      #4,D0
005AB2 C07C0001  and.w      #1,D0
005AB6 3B40066C  move.w      D0,$66C(A5)
005ABA C27C0007  and.w      #7,D1
005ABE 103B1020  move.b      $5AE0(PC,D1.w),D0
005AC2 33C00000066A  move.w      D0,$66A
005AC8 486D0654  pea        $654(A5)
005ACC 610022C4  bsr        $7D92
005AD0 584F      addq.w      #4,A7
005AD2 4E75      rts

*****
005AD4 014002800280  dc.w      320,640,640
005AD8 00C800C80190  dc.w      200,200,400

*****
005AE0 00      dc.b      0
005AE1 02      dc.b      2
005AE2 01      dc.b      1
005AE3 FF      dc.b      -1
005AE4 03      dc.b      3
005AE5 FF      dc.b      -1
005AE6 FF      dc.b      -1
005AE7 FF      dc.b      -1

Test / quality mode
Isolate bit
And save
Get printer configuration

Parallel / Serial
Isolate
And save for hardcopy
Isolate printer type
Get assignment from table
And save for hardcop
Address of the parameter block
Perform hardcopy
Correct stack pointer

Parameter table for hardcopy
Screen widths
Screen heights

Printer types (-1 = not implemented)
ATARI B/W matrix
ATARI B/W daisy wheel
ATARI color matrix
(ATARI color daisy wheel ?)
Epson B/W matrix
(Epson B/W daisy wheel)
(Epson color matrix)
(Epson color daisy wheel)

```

```

*****
005AE8 4E56FF0 link A6, #-16
005AEC 23FC000012C000025F6 move.l #300, $25F6
005AF6 4240 clr.w D0
005AF8 33C0000004A6 move.w D0, $4A6
005AFE 33C0000004692 move.w D0, $4692
005B04 3D40FFFE move.w D0, -2(A6)
005B08 604E bra $5B58
005B0A 207C00003E2A move.l #3E2A, A0
005B10 326EFFFEE move.w -2(A6), A1
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 3F2EFFFEE move.w -2(A6), -(A7)
005B22 42A7 clr.l -(A7)
005B24 42A7 clr.l -(A7)
005B26 4EB90000628C jsr $628C
005B2C DFFC0000000E add.l #$E, A7
005B32 3F00 move.w D0, -(A7)
005B34 306EFFFEE move.w -2(A6), A0
005B38 D1C8 add.l A0, A0
005B3A D1FC00004910 add.l #4910, A0
005B40 309F move.w (A7)+, (A0)
005B42 6610 bne $5B54
005B44 5279000004A6 addq.w #1, $4A6
005B4A 00B90000003000004C2 or.l #3, $4C2
005B54 526EFFFEE addq.w #1, -2(A6)
005B58 0C6E0002FFFE cmp.w #2, -2(A6)
005B5E 6DAA bit $5B0A
005B60 4E5E unlk A6
*****

hdv init
maxactim to 300*20 ms
nflops
Start with drive A
Address of the dsb
Drive number

Drive number

flopini
Correct stack pointer
Save error code
Drive number

Error code
Drive not present ?
Increment nflops
drvbits
Increment drive number
Not yet 2
Initialize next drive

```

```

005B62 4E75      rts

*****
005B64 4E56FFFC      link    A6,#-4
005B68 4280      clr.l    D0
005B6A 4E5E      unlk     A6
005B6C 4E75      rts

*****
005B6E 4E56FFF4      link    A6,#-12
005B72 48E7070C      movem.l D5-D7/A4-A5,-(A7)
005B76 0C6E00020008      cmp.w    #2,(A6)
005B7C 6D06      blt      $5B84
005B7E 4280      clr.l    D0
005B80 60000192      bra      $5D14:lnl:fp5
005B84 302E0008      move.w    8(A6),D0
005B88 EB40      asl.w    #5,D0
005B8A 48C0      ext.l    D0
005B8C 2A40      move.l    D0,A5
005B8E DBFC00003E3E      add.l    #53E3E,A5
005B94 284D      move.l    A5,A4
005B96 3EBC0001      move.w    #1,(A7)
005B9A 4267      clr.w    -(A7)
005B9C 4267      clr.w    -(A7)
005B9E 3F3C0001      move.w    #1,-(A7)
005BA2 3F2E0008      move.w    8(A6),-(A7)
005BA6 42A7      clr.l    -(A7)
005BA8 2F3C000012BC      move.l    #512BC,-(A7)
005BAE 4EB9000062D2      jsr      $62D2
005BB4 DFFC00000010      add.l    #510,A7
005BBA 2D40FFF4      move.l    D0,-12(A6)
005BBE 4AAEFFF4      tst.l    -12(A6)

*****
getbpb, get BIOS parameter block

Save registers
Drive number
< 2, ok
Else zero
Drive number
Timers 32

Plus base address
Count, read a sector
Side 0
Track 0
Sector 1
Drive number
Filler
Sector buffer
Read sector
Correct stack pointer
Save error code
And test

```



005BC2 6C16	bge \$5BDA	ok ?
005BC4 3EAE0008	move.w 8(A6), (A7)	Drive number
005BC8 202EFFF4	move.l -12(A6), D0	Error code
005BCC 3F00	move.w D0, -(A7)	
005BCE 4EB90000555C	jsr \$555C	Critical error handler
005BD4 548F	addq.l #2, A7	Correct stack pointer
005BD6 2D40FFF4	move.l D0, -12(A6)	Save error code
005BDA 202EFFF4	move.l -12(A6), D0	
005BDE B0BC00010000	cmp.l #\$10000, D0	Read boot sector again
005BE4 67B0	beq \$5B96	
005BE6 4AAEFFF4	tst.l -12(A6)	ok ?
005BEA 6C06	bge \$5BF2	
005BEC 4280	clr.l D0	
005BEE 60000124	bra \$5D14	
005BF2 2EBC000012C7	move.l #\$12C7, (A7)	Buffer+11, bytes per sector
005BF8 6100066C	bsr \$6266	u2i, convert 8086 integer to 68000 int
005BFC 3E00	move.w D0, D7	Save bytes per sector
005BFE 670E	beq \$5C0E	
005C00 1C39000012C9	move.b \$12C9, D6	Buffer+13, sectors per 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)	Buffer+22, sectors per FAT
005C20 61000644	bsr \$6266	u2i, convert 8086 integer to 68000 int
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	mults.w	2(A4),D0	Times clsiz
005C38 39400004	move.w	D0,4(A4)	Yields clsizb
005C3C 2EBC000012CD	move.l	#\$12CD,(A7)	Buffer+17, number of directory entries
005C42 61000622	bsr	\$6266	u2i, convert 8086 integer to 68000 int
005C46 EB40	asl.w	#5,D0	Times 32
005C48 48C0	ext.l	D0	
005C4A 81D4	divs.w	(A4),D0	Divided by recsiz
005C4C 39400006	move.w	D0,6(A4)	Yields 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)	Yields datrec
005C60 2EBC000012CF	move.l	#\$12CF,(A7)	Buffer+19, number of sectors
005C66 610005FE	bsr	\$6266	u2i, convert 8086 integer to 68000 int
005C6A 906C000C	sub.w	12(A4),D0	Minus datrec
005C6E 48C0	ext.l	D0	
005C70 81EC0002	divs.w	2(A4),D0	Divided by clsiz
005C74 3940000E	move.w	D0,14(A4)	Yields numcl
005C78 2EBC000012D6	move.l	#\$12D6,(A7)	Buffer+26, number of sides
005C7E 610005E6	bsr	\$6266	u2i, convert 8086 integer to 68000 int
005C82 3B400014	move.w	D0,20(A5)	dnside
005C86 2EBC000012D4	move.l	#\$12D4,(A7)	Buffer+24, sector per track
005C8C 610005D8	bsr	\$6266	u2i, convert 8086 integer to 68000 int
005C90 3B400018	move.w	D0,24(A5)	dspt
005C94 302D0014	move.w	20(A5),D0	dnside
005C98 C1ED0018	mults.w	24(A5),D0	Times dspt
005C9C 3B400016	move.w	D0,22(A5)	Yields dspt
005CA0 2EBC000012D8	move.l	#\$12D8,(A7)	Buffer+28, number of hidden sectors
005CA6 610005BE	bsr	\$6266	u2i, convert 8086 integer to 68000 int
005CAA 3B40001A	move.w	D0,26(A5)	dhidden

005CAE 2EBC000012CF	move.l	#\$12CF,(A7)	Buffer+19, number of sectors on disk
005CB4 610005B0	bsr	\$6266	u2i, convert 8086 integer to 68000 int
005CB8 48C0	ext.l	D0	
005CBA 81ED0016	divs.w	22(A5),D0	Divided by dspc
005CBE 3B400012	move.w	D0,18(A5)	yields dntracks
005CC2 4247	clr.w	D7	Counter to zero
005CC4 6016	bra	\$5CDC	Jump to end of loop
005CC6 204D	move.l	A5,A0	Buffer pointer
005CC8 3247	move.w	D7,A1	Loop counter
005CCA D1C9	add.l	A1,A0	Plus BPB address
005CCC 3247	move.w	D7,A1	Loop counter
005CCE D3FC000012BC	add.l	#\$12BC,A1	Plus buffer address
005CD4 116900008001C	move.b	8(A1),28(A0)	Copy byte of the serial number
005CDA 5247	addq.w	#1,D7	Next byte
005CDC BE7C0003	cmp.w	#3,D7	Three bytes already ?
005CE0 6DE4	blt	\$5CC6	No
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	Disk status uncertain
005D00 6002	bra	\$5D04	
005D02 4240	clr.w	D0	Status certain
005D04 227C000003E2A	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	Address of BPB as result

```

005D14 4A9F          tst.l    (A7) +
005D16 4CDF30C0      movem.l  (A7)+,D6-D7/A4-A5      Restore registers
005D1A 4E5E          unlk     A6
005D1C 4E75          rts

*****
005D1E 4E560000      link     A6,#0
005D22 48E70304      movem.l  D6-D7/A5,-(A7)      Save registers
005D26 0C6E0020008  cmp.w    #2,8(A6)          Drive number
005D2C 6D04          blt     $5D32             < 2?
005D2E 70F1          moveq.l  #-15,D0          'unknown device'
005D30 604C          bra     $5D7E             Error branch
005D32 3E2E0008      move.w    8(A6),D7          Drive number
005D36 3A47          move.w    D7,A5
005D38 DBFC0003E2A  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
005D66 A 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  move.l   (A7)+,D7/A5
005D84 4E5E      unlk     A6
005D86 4E75      rts

*****
005D88 4E56FFFC  link     A6,#-4
005D8C 48E70F04  movem.l  D4-D7/A5,-(A7)
005D90 0C6E00020012  cmp.w    #2,18(A6)
005D96 6D06      blt      $5D9E
005D98 70F1      moveq.l  #-15,D0
005D9A 6000010E  bra      $5EAA
005D9E 3C2E0012  move.w   18(A6),D6
005DA2 0C6E00020008  cmp.w    #2,8(A6)
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
005DC0 3E00      move.w   D0,D7
005DC2 BE7C0002  cmp.w    #2,D7
005DC6 660A      bne      $5DD2
005DC8 70F2      moveq.l  #-14,D0
005DCA 600000DE  bra      $5EAA

*****
rwabs, read/write sector(s)

Save registers
Drive number
< 2 ?
No, unknown device
Error branch
Save drive number
rwflag

Test media change
Save status
Disk changed ?
No
Media change error
Error branch

```

005DCE 600000A8	bra	\$5E78	
005DD2 BE7C0001	cmp.w	#1,D7	Disk possibly changed ?
005DD6 660000A0	bne	\$5E78	No
005DDA 3EBC0001	move.w	#1,(A7)	Read a sector (Boot sector)
005DDE 4267	clr.w	-(A7)	Side 0
005DE0 4267	clr.w	-(A7)	Track 0
005DE2 3F3C0001	move.w	#1, -(A7)	Sector 1
005DE6 3F06	move.w	D6, -(A7)	Drive number
005DE8 42A7	clr.l	-(A7)	Filler
005DEA 2F3C00012BC	move.l	#\$12BC, -(A7)	Sector buffer
005DF0 4EB9000062D2	jsr	\$62D2	floprd
005DF6 DFFC00000010	add.l	#\$10,A7	Correct stack pointer
005DFC 2D40FFFC	move.l	D0, -4(A6)	Save error number
005E00 4AAEFFFC	tst.l	-4(A6)	And test
005E04 6C14	bge	\$5E1A	Ok ?
005E06 3E86	move.w	D6, (A7)	Error
005E08 202EFFFC	move.l	-4(A6), D0	Pass an critical error handler
005E0C 3F00	move.w	D0, -(A7)	Correct stack pointer
005E0E 4EB90000555C	jsr	\$555C	
005E14 548F	addq.l	#2,A7	
005E16 2D40FFFC	move.l	D0, -4(A6)	
005E1A 202EFFFC	move.l	-4(A6), D0	
005E1E B0BC00010000	cmp.l	#\$10000, D0	
005E24 67B4	beq	\$5DDA	Read again
005E26 4AAEFFFC	tst.l	-4(A6)	Error number
005E2A 6C08	bge	\$5E34	Ok ?
005E2C 202EFFFC	move.l	-4(A6), D0	Error number
005E30 60000078	bra	\$5EAA	Error branch
005E34 4247	clr.w	D7	Clear media change status
005E36 601C	bra	\$5E54	

005E38 207C000012BC	move.l #12BC,A0	Address of the sector buffer
005E3E 10307008	move.b 8(A0,D7.w),D0	Serial number
005E42 4880	ext.w D0	
005E44 1235701C	move.b 28(A5,D7.w),D1	Compare
005E48 4881	ext.w D1	
005E4A B041	cmp.w D1,D0	With previous value
005E4C 6704	beq \$5E52	Ok ?
005E4E 70F2	moveq.l #-14,D0	Media change
005E50 6058	bra \$5EAA	Error branch
005E52 5247	addq.w #1,D7	Next byte of the serial number
005E54 BE7C0003	cmp.w #3,D7	All 3 bytes tested ?
005E58 6DDE	blt \$5E38	No
005E5A 3046	move.w D6,A0	Drive number
005E5C D1FC00000676	add.l #676,A0	wplatch
005E62 3246	move.w D6,A1	Drive number
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	Error branch
005E84 0C6E00010008	cmp.w #1,8(A6)	Drive number
005E8A 6F04	ble \$5E90	
005E8C 556E0008	subq.w #2,8(A6)	Drive number
005E90 3EAE000E	move.w 14(A6),(A7)	count
005E94 3F06	move.w D6,-(A7)	Drive number

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	Correct stack pointer
005EAA 4A9F	tst.l (A7)+	Restore registers
005EAC 4CDF20E0	movem.l (A7)+,D5-D7/A5	
005EB0 4E5E	unlk A6	
005EB2 4E75	rts	
*****		
005EB4 4E56FFFA	link A6,#-6	floprw, read/write sectors
005EB8 48E73F04	movem.l D2-D7/A5,-(A7)	Save registers
005EBC 302E0010	move.w 16(A6),D0	Drive number
005EC0 EB40	asl.w #5,D0	Times 32
005EC2 48C0	ext.l D0	
005EC4 2A40	move.l D0,A5	
005EC6 DBFC00003E3E	add.l #\$3E3E,A5	Plus base address BPB
005ECC 082E0000000D	btst #0,13(A6)	Buffer address not even ?
005ED2 6604	bne \$5ED8	Yes
005ED4 4240	clr.w D0	Clear oddflag
005ED6 6002	bra \$5EDA	
005ED8 7001	moveq.l #1,D0	Set oddflag
005EDA 3D40FFFE	move.w D0,-2(A6)	And save
005EDE 4A6D0016	tst.w 22(A5)	dspc set ?
005EE2 660A	bne \$5EEE	Yes
005EE4 7009	moveq.l #9,D0	Else take 9
005EE6 3B400016	move.w D0,22(A5)	As dspt
005EEA 3B400018	move.w D0,24(A5)	And dspc
005EEE 6000015E	bra \$604E	



005EF2 4A6EFFFFE	tst.w	-2(A6)	oddflag set ?
005EF6 6708	beq	\$5F00	No
005EF8 203C000012BC	move.l	#12BC,D0	Sector buffer
005EFE 6004	bra	\$5F04	
005F00 202E000A	move.l	10(A6),D0	Get buffer address
005F04 2D40FFFA	move.l	D0,-6(A6)	And save
005F08 3C2E000E	move.w	14(A6),D6	recno, logical sector number
005F0C 48C6	ext.l	D6	
005F0E 8DED0016	divs.w	22(A5),D6	Divided by dsptc yields track number
005F12 3B2E000E	move.w	14(A6),D4	recno, logical sector number
005F16 48C4	ext.l	D4	
005F18 89ED0016	divs.w	22(A5),D4	Divided by dsptc, sector per track
005F1C 4844	swap	D4	Remainder of division as sector number
005F1E B86D0018	cmp.w	24(A5),D4	Compare with dspt
005F22 6C04	bge	\$5F28	Greater than or equal ?
005F24 4245	clr.w	D5	Side 0
005F26 6006	bra	\$5F2E	
005F28 7A01	moveq.l	#1,D5	Side 1
005F2A 986D0018	sub.w	24(A5),D4	Subtract dspt
005F2E 4A6EFFFFE	tst.w	-2(A6)	oddflag set ?
005F32 6704	beq	\$5F38	No
005F34 7601	moveq.l	#1,D3	Set counter to 1
005F36 6018	bra	\$5F50	
005F38 302D0018	move.w	24(A5),D0	dspt
005F3C 9044	sub.w	D4,D0	Minus Sector number
005F3E B06E0012	cmp.w	18(A6),D0	Compare with number of sectors
005F42 6C08	bge	\$5F4C	Greater or equal ?
005F44 362D0018	move.w	24(A5),D3	dspt
005F48 9644	sub.w	D4,D3	Minus sector number equals counter
005F4A 6004	bra	\$5F50	
005F4C 362E0012	move.w	18(A6),D3	Number of sectors as counter

005F50 5244	addq.w #1,D4	Increment sector # (1st number is 1)
005F52 4A6E0008	tst.w 8(A6)	Test read-write flag
005F56 67000080	beq \$5FD8	Read
005F5A 202EFFFF	move.l -6(A6),D0	Buffer pointer
005F5E B0AE000A	cmp.l 10(A6),D0	Equals buffer address
005F62 6710	beq \$5F74	Yes
005F64 2EAEFFFF	move.l -6(A6),(A7)	Source address
005F68 2F2E000A	move.l 10(A6),-(A7)	Destination address
005F6C 4EB900005910	jsr \$5910	fastcopy, copy a sector
005F72 588F	addq.l #4,A7	Correct stack pointer
005F74 3E83	move.w D3,(A7)	Number of sectors
005F76 3F05	move.w D5,-(A7)	Side
005F78 3F06	move.w D6,-(A7)	Track
005F7A 3F04	move.w D4,-(A7)	Sector
005F7C 3F2E0010	move.w 16(A6),-(A7)	Drive
005F80 42A7	clr.l -(A7)	Filler
005F82 2F2EFFFF	move.l -6(A6),-(A7)	Sector buffer
005F86 4EB9000063B0	jsr \$63B0	flopwr, read sector
005F8C DFFC00000010	add.l #10,A7	Correct stack pointer
005F92 2E00	move.l D0,D7	Error code
005F94 4A87	tst.l D7	Ok ?
005F96 663E	bne \$5FD6	No
005F98 4A7900000444	tst.w \$444	fverify, verify required ?
005F9E 6736	beq \$5FD6	No
005FA0 3E83	move.w D3,(A7)	Number of sectors
005FA2 3F05	move.w D5,-(A7)	Side
005FA4 3F06	move.w D6,-(A7)	Track
005FA6 3F04	move.w D4,-(A7)	Sector
005FA8 3F2E0010	move.w 16(A6),-(A7)	Drive
005FAC 42A7	clr.l -(A7)	Filler
005FAE 2F3C000012BC	move.l #12BC,-(A7)	Sector buffer
005FB4 4EB900006602	jsr \$6602	flopver, verify sector

005FBA DFFC00000010	add.l \$S10,A7	Correct stack pointer
005FC0 2E00	move.l D0,D7	Error code
005FC2 4A87	tst.l D7	Ok ?
005FC4 6610	bne \$5FD6	No
005FC6 2EBC000012BC	move.l #512BC,(A7)	Sector buffer
005FCC 61000298	bsr \$6266	u2i, convert 8086 integer to 68000 int
005FD0 4A40	tst.w D0	Bad sector list
005FD2 6702	beq \$5FD6	Sectors OK ?
005FD4 7EF0	moveq.l #-16,D7	Bad sectors
005FD6 603A	bra \$6012	
005FD8 3E83	move.w D3,(A7)	Number of sectors
005FDA 3F05	move.w D5,-(A7)	Side
005FDC 3F06	move.w D6,-(A7)	Track
005FDE 3F04	move.w D4,-(A7)	Sector
005FE0 3F2E0010	move.w 16(A6),-(A7)	Drive
005FE4 42A7	clr.l -(A7)	Filler
005FE6 2F2EFFFF	move.l -6(A6),-(A7)	Sector buffer
005FEA 4E9000062D2	jsr \$62D2	flopdr, read sector
005FF0 DFFC00000010	add.l \$S10,A7	Correct stack pointer
005FF6 2E00	move.l D0,D7	Save error code
005FF8 202EFFFF	move.l -6(A6),D0	User buffer
005FFC B0AE000A	cmp.l 10(A6),D0	Equals desired buffer ?
006000 6710	beq \$6012	Yes
006002 2EAE000A	move.l 10(A6),(A7)	Source address
006006 2F2EFFFF	move.l -6(A6),-(A7)	Destination
00600A 4EB900005910	jsr \$5910	fastcpy, copy sector
006010 588F	addq.l #4,A7	Correct stack pointer
006012 4A87	tst.l D7	Test error code
006014 6C12	bge \$6028	Ok ?
006016 3EAE0010	move.w 16(A6),(A7)	Drive number
00601A 2007	move.l D7,D0	Error code

00601C 3F00	move.w	D0,-(A7)	On stack
00601E 4EB90000555C	jsr	\$555C	Critical error handler
006024 548F	addq.l	#2,A7	Correct stack pointer
006026 2E00	move.l	D0,D7	Get error code
006028 BEBC00010000	cmp.l	#\$10000,D7	Attempt again ?
00602E 6700FF22	beq	\$5F52	Yes
006032 4A87	tst.l	D7	Error code
006034 6C04	bge	\$603A	OK
006036 2007	move.l	D7,D0	Error code as result
006038 601E	bra	\$6058	
00603A 3003	move.w	D3,D0	Sector counter
00603C 48C0	ext.l	D0	
00603E 7209	moveq.l	#9,D1	Times 512
006040 E3A0	asl.l	D1,D0	
006042 D1AE000A	add.l	D0,10(A6)	Increment buffer address
006046 D76E000E	add.w	D3,14(A6)	Logical sector number plus counter
00604A 976E0012	sub.w	D3,18(A6)	Decrement number of sectors to process
00604E 4A6E0012	tst.w	18(A6)	Sectors yet to read/write ?
006052 6600FE9E	bne	\$5EF2	Yes
006056 4280	clr.l	D0	OK
006058 4A9F	tst.l	(A7)+	
00605A 4CDF20F8	movem.l	(A7)+,D3-D7/A5	Restore registers
00605E 4E5E	unlk	A6	
006060 4E75	rts		
*****			
006062 4E56FFFC	link	A6,#-4	random, generate random number
006066 4AB9000025FA	tst.l	\$25FA	Last random number
00606C 6616	bne	\$6084	Not zero ?
00606E 2039000004EA	move.l	\$4BA,D0	200 Hz counter
006074 7210	moveq.l	#16,D1	

```

006076 E3A0          asl.l    D1,D0          Bits 0-15 to 16-31
006078 80B9000004BA  or.l     $4BA,D0          Plus 200 Hz counter
00607E 23C0000025FA  move.l   D0,$25FA          Use as start value
006084 2F3CBB40E62D  move.l   #BB40E62D,-(A7)  3141592621
00608A 2F39000025FA  move.l   $25FA,-(A7)      Last random value
006090 4EB9000094FA  jsr      $94FA            32 * 32 bit multiplication
006096 508F          addq.l   #8,A7          Correct stack pointer
006098 5280          addq.l   #1,D0          Result plus 1
00609A 23C0000025FA  move.l   D0,$25FA          As new start value
0060A0 2039000025FA  move.l   $25FA,D0
0060A6 E080          asr.l     #8,D0          Bits 31-8 to 23-0
0060A8 C0BC00FFFF    and.l    #$FFFFFF,D0      24-bit random number as result
0060AE 4E5E          unlk     A6
0060B0 4E75          rts

*****
0060B2 4E560000      link     A6,#0          bootload, load boot sector
0060B6 48E70300      movem.l D6-D7,-(A7)      Save registers
0060BA 4EB900005932  jsr      $5932          hdv init
0060C0 4A79000004A6  tst.w    $4A6           nflops
0060C6 6704          beq      $60CC          No drive connected ?
0060C8 7001          moveq.l #1,D0          'couldn't load'
0060CA 6002          bra      $60CE          'no drive'
0060CC 7002          moveq.l #2,D0          Save errors
0060CE 3E00          move.w   D0,D7         nflops
0060D0 4A79000004A6  tst.w    $4A6           bootdev
0060D6 6744          beq      $611C          No diskette ?
0060D8 0C7900020000446  cmp.w    #2,$446       One sector
0060E0 6C3A          bge      $611C          Side 0
0060E2 3EBC0001      move.w   #1,(A7)       Track 0
0060E6 4267          clr.w    -(A7)
0060E8 4267          clr.w    -(A7)

```

0060EA 3F3C0001	move.w #1,-(A7)	Sector 1
0060EE 3F3900000446	move.w \$446,-(A7)	bootdev as drive number
0060F4 42A7	clr.l -(A7)	Filler
0060F6 2F3C000012BC	move.l #\$12BC,-(A7)	Sector buffer
0060FC 4EB90000062D2	jsr \$62D2	flopdr, read sector
006102 DFFC00000010	add.l #\$10,A7	Correct stack pointer
006108 4A80	tst.l D0	Error ?
00610A 6604	bne \$6110	No
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	Get old error code
006122 601C	bra \$6140	
006124 3EBC0100	move.w #\$100,(A7)	\$100 words
006128 2F3C000012BC	move.l #\$12BC,-(A7)	Sector buffer
00612E 61000106	bsr \$6236	Calculate checksum
006132 588F	addq.l #4,A7	Correct stack pointer
006134 B07C1234	cmp.w #\$1234,D0	Compare with checksum for boot sector
006138 6604	bne \$613E	Not equal ?
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	Restore registers

```

006146 4E5E      unlk  A6
006148 4E75      rts

*****
00614A 4E5EFFFF    link  A6,#-6
00614E 48E70704    movem.l D5-D7/A5,-(A7)
006152 4A6E0012    tst.w  18(A6)
006156 6C1E        bge    $6176
006158 3EBC0100    move.w  #$100,(A7)
00615C 2F2E0008    move.l  8(A6),-(A7)
006160 610000D4    bsr    $6236
006164 588F        addq.l  #4,A7
006166 B07C1234    cmp.w  #$1234,D0
00616A 6704        beq    $6170
00616C 4240        clr.w  D0
00616E 6002        bra    $6172
006170 7001        moveq.l #1,D0
006172 3D400012    move.w  D0,18(A6)
006176 4AAE000C    tst.l  12(A6)
00617A 6D3E        blt    $61BA
00617C 202E000C    move.l  12(A6),D0
006180 B0BC00FFFF    cmp.l  $FFFFFF,D0
006186 6F08        ble    $6190
006188 6100FED8    bsr    $6062
00618C 2D40000C    move.l  D0,12(A6)
006190 4247        clr.w  D7
006192 6020        bra    $61B4

006194 202E000C    move.l  12(A6),D0
006198 C0BC000000FF    and.l  $FFF,D0
00619E 3247        move.w  D7,A1

```

proto bt, generate boot sector

Save registers

Test execflg

Maintain executability

\$100 words

Address of the sector buffer

Calculate checksum

Correct stack pointer

Equals checksum for boot sector?

Yes

Not executable

Executable

execflg to 1 = boot sector executable

Serial number

Negative, don't change

Serial number

> \$FFFFFF ?

No

rand, generate random number

Save

Clear counter

Random number

Bits 0-7

Pointer to next byte in buffer

0061A0 D3EE0008	add.l	8(A6),A1	Plus base address
0061A4 13400008	move.b	D0,8(A1)	Write byte of serial number in buffer
0061A8 202E000C	move.l	12(A6),D0	Random number
0061AC E080	asr.l	#8,D0	In order to shift 8 bits right
0061AE 2D40000C	move.l	D0,12(A6)	And save new value
0061B2 5247	addq.w	#1,D7	Increment counter
0061B4 BE7C0003	cmp.w	#3,D7	Three bytes copied already ?
0061B8 6DDA	blt	\$6194	No
0061BA 4A6E0010	tst.w	16(A6)	Diskette size
0061BE 6D28	blt	\$61E8	Negative, don't change
0061C0 3C2E0010	move.w	16(A6),D6	Diskette size
0061C4 CDFC0013	mults.w	#13,D6	Times 19 equals pointer to prototype BPB
0061C8 4247	clr.w	D7	Clear counter
0061CA 6016	bra	\$61E2	
0061CC 3047	move.w	D7,A0	Counter
0061CE D1EE0008	add.l	8(A6),A0	Plus address of the buffer
0061D2 3246	move.w	D6,A1	
0061D4 D3FC00016D00	add.l	#\$16D00,A1	Address of the prototype BPB
0061DA 1151000B	move.b	(A1),11(A0)	Copy BPB
0061DE 5246	addq.w	#1,D6	
0061E0 5247	addq.w	#1,D7	Increment counter
0061E2 BE7C0013	cmp.w	#13,D7	Already 19 ?
0061E6 6DE4	blt	\$61CC	No
0061E8 426EFFFFA	clr.w	-6(A6)	
0061EC 2D6E0008FFFC	move.l	8(A6),-4(A6)	Buffer address
0061F2 600E	bra	\$6202	
0061F4 206EFFFFC	move.l	-4(A6),A0	
0061F8 3010	move.w	(A0),D0	Get word from buffer
0061FA D16EFFFFA	add.w	D0,-6(A6)	Sum for checksum generation
0061FE 54AEFFFFC	addq.l	#2,-4(A6)	Next word



006202 202E0008	move.l 8(A6),D0	Buffer address
006206 D0BC00001FE	add.l #1FE,D0	Plus 1FE
00620C B0AEFFFC	cmp.l -4(A6),D0	Last word already
006210 62E2	bhi \$61F4	
006212 303C1234	move.w #1234,D0	Checksum for boot sector
006216 906EFFFF	sub.w -6(A6),D0	Subtract from previous value
00621A 226EFFFF	move.l -4(A6),A1	
00621E 3280	move.w D0,(A1)	Checksum in buffer
006220 4A6E0012	tst.w 18(A6)	execflg
006224 6606	bne \$622C	Boot sector executable ?
006226 206EFFFF	move.l -4(A6),A0	
00622A 5250	addq.w #1,(A0)	Increment checksum, not executable
00622C 4A9F	tst.l (A7)+	
00622E 4CDF20C0	movem.l (A7)+,D6-D7/A5	Restore registers
006232 4E5E	unlk A6	
006234 4E75	rts	
*****		
006236 4E560000	link A6,#0	Calculate checksum
00623A 48E70300	movem.l D6-D7,-(A7)	Save registers
00623E 4247	clr.w D7	Clear sum
006240 600C	bra \$624E	
006242 206E0008	move.l 8(A6),A0	Address of the buffer
006246 3010	move.w (A0),D0	Get word
006248 DE40	add.w D0,D7	And sum
00624A 54AE0008	addq.l #2,8(A6)	Pointer to next word
00624E 302E000C	move.w 12(A6),D0	Number of words
006252 536E000C	subq.w #1,12(A6)	Minus 1
006256 4A40	tst.w D0	All words added already ?
006258 66E8	bne \$6242	No
00625A 3007	move.w D7,D0	Result to d0
00625C 4A9F	tst.l (A7)+	

```

00625E 4CDF0080      movem.l (A7)+,D7      Restore registers
006262 4E5E        unlk     A6
006264 4E75        rts

*****
006266 4E56FFFC      link     A6,#-4
00626A 206E0008      move.l  8(A6),A0
00626E 10280001      move.b  1(A0),D0
006272 4880          ext.w   D0
006274 C07C00FF      and.w   #$FF,D0
006278 E140          asl.w   #8,D0
00627A 226E0008      move.l  8(A6),A1
00627E 1211          move.b  (A1),D1
006280 4881          ext.w   D1
006282 C27C00FF      and.w   #$FF,D1
006286 8041          or.w   D1,D0
006288 4E5E        unlk     A6
00628A 4E75        rts

*****
00628C 43F9000006C8      lea     $6C8,A1
006292 4A6F000C      tst.w  12(A7)
006296 6706          beq     $629E
006298 43F9000006CC      lea     $6CC,A1
00629E 337900000400002  move.w  $440,2(A1)
0062A6 70FF          moveq.l #-1,D0
0062A8 42690000      clr.w  0(A1)
0062AC 610004BA      bsr     $6768
0062B0 61000696      bsr     $6948
0062B4 337CFF000000      move.w  #$FF00,0(A1)
0062BA 61000618      bsr     $68D4
0062BE 7E06          moveq.l #6,D7

*****
flop1n1, initialize drive
dsb0, pointer to DSB drive A
Drive A ?
Yes
dsb1, take DSB from drive B
seekrate
Default error number
Track number to zero
floplock, set parameters
Select drive and side
restore

```

```

0062C0 610005A0      bsr      $6862
0062C4 6608          bne      $62CE
0062C6 6100060C      bsr      $68D4
0062CA 67000542      beq      $680E
0062CE 60000530      bra      $6800

*****
0062D2 6100071E      bsr      $69F2
0062D6 70F5          moveq.l #-11,D0
0062D8 6100048E      bsr      $6768
0062DC 6100066A      bsr      $6948
0062E0 610005CC      bsr      $68AE
0062E4 66000090      bne      $6376
0062E8 33FCFF00006A2 move.w #-1,$6A2
0062F0 3CBC0090      move.w #$90,(A6)
0062F4 3CBC0190      move.w #$190,(A6)
0062F8 3CBC0090      move.w #$90,(A6)
0062FC 33ED068CFFFF8604 move.w $68C(A5),$FFFF8604
006304 3CBC0080      move.w #$80,(A6)
006308 3E3C0090      move.w #$90,D7
00630C 610006B6      bsr      $69C4
006310 2E3C00040000 move.l #$40000,D7
006316 246D0692      move.l $692(A5),A2
00631A 08390005FFFFFA01 btst    #5,$FFFFFA01
006322 6734          beq      $6358
006324 5387          subq.l #1,D7
006326 6724          beq      $634C
006328 1B79FFFF8609069D move.b $FFFF8609,$69D(A5)
006330 1B79FFFF860B069E move.b $FFFF860B,$69E(A5)
006338 1B79FFFF860D069F move.b $FFFF860D,$69F(A5)
006340 B5ED069C      cmp.l  $69C(A5),A2
006344 6ED4          bgt      $631A

*****
Restore
flopok, no error
flopfail, error

flopdr, read sector(s) from disk
Change, test for disk change
Error number to read error
floplock, set parameters
Select drive and side
go2track, search for track
Try again if error
currerr to default error
Clear DMA status

Data direction to READ
ccount to dsksctl, sector counter
Read sector command for 1772
Read multiple
wdiskctl, pass D7 to 1772
Initialize timeout counter
edma, destination address for DMA
mfp gpip, 1772 done ?
Yes
Decrement timeout counter
Timed-out ?
dmahigh
dmamid
dmalow
Current DMA address equal edma?
No, continue to wait

```

006346 610005E6	bsr \$692E	Reset, end transfer
00634A 600C	bra \$6358	
00634C 3E7CFFFE06A2	move.w #-2,\$6A2(A5)	currerr to timeout
006352 610005DA	bsr \$692E	Reset, end transfer
006356 601E	bra \$6376	Start next attempt
006358 3CBC0090	move.w #\$90,(A6)	Select DMA status register
00635C 3016	move.w (A6),D0	Read status
00635E 08000000	btst #0,D0	DMA error ?
006362 6712	beq \$6376	Yes, try again
006364 3CBC0080	move.w #\$80,(A6)	Select 1772 status register
006368 6100066E	bsr \$69D8	rdiskctl, read register
00636C C03C0018	and.b #\$18,D0	RNF, isolate checksum und lost data
006370 6700049C	beq \$680E	flopok, no error
006374 6118	bsr \$638E	errbits, determine error number
006376 0C6D00010672	cmp.w #1,\$672(A5)	retrycnt to second attempt ?
00637C 6604	bne \$6382	No
00637E 610004FA	bsr \$687A	reseek, home and reseek
006382 536D0672	subq.w #1,\$672(A5)	retrycnt, decrement attempt counter
006386 6A00FF54	bpl \$62DC	Another attempt ?
00638A 60000474	bra \$6800	No, flopfail
*****		
00638E 72F3	moveq.l #-13,D1	errbits, 1772 status in error number
006390 08000006	btst #6,D0	Write protect ?
006394 6614	bne \$63AA	Yes
006396 72F8	moveq.l #-8,D1	Record not found ?
006398 08000004	btst #4,D0	
00639C 660C	bne \$63AA	Yes
00639E 72FC	moveq.l #-4,D1	CRC error ?
0063A0 08000003	btst #3,D0	
0063A4 6704	beq \$63AA	Yes
0063A6 322D06A0	move.w \$6A0(A5),D1	defererror, take default error

```

0063AA 3B4106A2      move.w D1,$6A2(A5)
0063AE 4E75          rts

*****
0063B0 61000640      bsr $69F2
0063B4 70F6          moveq.l #-10,D0
0063B6 610003B0      bsr $6768
0063BA 302D0688      move.w $688(A5),D0
0063BE 5340          subq.w #1,D0
0063C0 806D0686      or.w $686(A5),D0
0063C4 806D068A      or.w $68A(A5),D0
0063C8 6606          bne $63D0
0063CA 7002          moveq.l #2,D0
0063CC 6100065C      bsr $6A2A
0063D0 61000576      bsr $6948
0063D4 610004D8      bsr $68AE
0063D8 6600007E      bne $6458
0063DC 3B7CFFF06A2   move.w #-1,$6A2(A5)
0063E2 3CBC0190      move.w #$190,(A6)
0063E6 3CBC0090      move.w #$90,(A6)
0063EA 3CBC0190      move.w #$190,(A6)
0063EE 3E3C0001      move.w #1,D7
0063F2 610005D0      bsr $69C4
0063F6 3CBC0180      move.w #$180,(A6)
0063FA 3E3C00AC      move.w #$A0,D7
0063FE 610005C4      bsr $69C4
006402 2E3C00040000   move.l #$40000,D7
006408 08390005FFFFFA01 btst #5,$FFFA01
006410 670A          beq $641C
006412 5387          subq.l #1,D7
006414 66F2          bne $6408
006416 61000516      bsr $692E

As currerr
flopwr, write sector(s) on disk
Change, test for disk change
Default error to write wrorr
floplock, set parameters
csect, sector number 1?

ctrack, track number 0?
cside, side 0?
No, not boot sector
Media change
Set to 'unsure'
select, select drive and seide
go2track, search for track
Error, try again
currerr to default error
Clear DMA status

Data direction to WRITE
Sector count register
wdiskctl, D7 to 1772
Selects1772
Write Sector
wdiskctl, D7 to 1772
Timeout counter
mfp, gpip, 1772 done ?
Yes
Decrement timeout counter
Not timed-out yet ?
reset, end transfer

```

```

000641A 6034
000641C 3CEC0180
0006420 610005B6
0006424 6100FF68
0006428 08000006
000642C 660003D2
0006430 C03C005C
0006434 661A
0006436 526D0688
000643A 06AD00000200068E
0006442 536D068C
0006446 670003C6
000644A 61000524
000644E 608C
0006450 0C6D00010672
0006456 6604
0006458 61000420
000645C 536D0672
0006460 6A00FF6E
0006464 6000039A
*****
0006468 0CAF876543210016
0006470 6600038E
0006474 6100057C
0006478 70FF
000647A 610002EC
000647E 610004C8
0006482 3B6F000E0696
0006488 3B6F00140698
000648E 3B6F001A069A
0006492 3B6F001E069C
0006496 3B6F0022069E
000649A 3B6F002806A0
000649E 3B6F002D06A4
00064A2 3B6F003206A8
00064A6 3B6F003806AC
00064AA 3B6F003D06B0
00064AE 3B6F004206B4
00064B2 3B6F004806B8
00064B6 3B6F004D06BC
00064BA 3B6F005206C0
00064BE 3B6F005806C4
00064C2 3B6F005D06C8
00064C6 3B6F006206CC
00064CA 3B6F006806D0
00064CE 3B6F006D06D4
00064D2 3B6F007206D8
00064D6 3B6F007806DC
00064DA 3B6F007D06E0
00064DE 3B6F008206E4
00064E2 3B6F008806E8
00064E6 3B6F008D06EC
00064EA 3B6F009206F0
00064EE 3B6F009806F4
00064F2 3B6F009D06F8
00064F6 3B6F00A206FC
00064FA 3B6F00A80600
00064FE 3B6F00AD0604
0006502 3B6F00B20608
0006506 3B6F00B8060C
000650A 3B6F00BD0610
000650E 3B6F00C20614
0006512 3B6F00C80618
0006516 3B6F00CD061C
000651A 3B6F00D20620
000651E 3B6F00D80624
0006522 3B6F00DE0628
0006526 3B6F00E2062C
000652A 3B6F00E80630
000652E 3B6F00ED0634
0006532 3B6F00F20638
0006536 3B6F00F8063C
000653A 3B6F00FD0640
000653E 3B6F00030644
0006542 3B6F00090648
0006546 3B6F000F064C
000654A 3B6F00150650
000654E 3B6F001B0654
0006552 3B6F00210658
0006556 3B6F0027065C
000655A 3B6F002D0660
000655E 3B6F00330664
0006562 3B6F00390668
0006566 3B6F003F066C
000656A 3B6F00450670
000656E 3B6F004B0674
0006572 3B6F00510678
0006576 3B6F0057067C
000657A 3B6F005D0680
000657E 3B6F00630684
0006582 3B6F00690688
0006586 3B6F006F068C
000658A 3B6F00750690
000658E 3B6F007B0694
0006592 3B6F00810698
0006596 3B6F0087069C
000659A 3B6F008D06A0
000659E 3B6F009306A4
00065A2 3B6F009906A8
00065A6 3B6F009F06AC
00065AA 3B6F00A506B0
00065AE 3B6F00AB06B4
00065B2 3B6F00B106B8
00065B6 3B6F00B706BC
00065BA 3B6F00BD06C0
00065BE 3B6F00C306C4
00065C2 3B6F00C906C8
00065C6 3B6F00CF06CC
00065CA 3B6F00D506D0
00065CE 3B6F00DB06D4
00065D2 3B6F00E106D8
00065D6 3B6F00E706DC
00065DA 3B6F00ED06E0
00065DE 3B6F00F306E4
00065E2 3B6F00F906E8
00065E6 3B6F00FF06EC
00065EA 3B6F010506F0
00065EE 3B6F010B06F4
00065F2 3B6F011106F8
00065F6 3B6F011706FC
00065FA 3B6F011D0600
00065FE 3B6F01230604
0006602 3B6F01290608
0006606 3B6F012F060C
000660A 3B6F01350610
000660E 3B6F013B0614
0006612 3B6F01410618
0006616 3B6F0147061C
000661A 3B6F014D0620
000661E 3B6F01530624
0006622 3B6F01590628
0006626 3B6F015F062C
000662A 3B6F01650630
000662E 3B6F016B0634
0006632 3B6F01710638
0006636 3B6F0177063C
000663A 3B6F017D0640
000663E 3B6F01830644
0006642 3B6F01890648
0006646 3B6F018F064C
000664A 3B6F01950650
000664E 3B6F019B0654
0006652 3B6F01A10658
0006656 3B6F01A7065C
000665A 3B6F01AD0660
000665E 3B6F01B30664
0006662 3B6F01B90668
0006666 3B6F01BF066C
000666A 3B6F01C50670
000666E 3B6F01CB0674
0006672 3B6F01D10678
0006676 3B6F01D7067C
000667A 3B6F01DD0680
000667E 3B6F01E30684
0006682 3B6F01E90688
0006686 3B6F01EF068C
000668A 3B6F01F50690
000668E 3B6F01FB0694
0006692 3B6F01010698
0006696 3B6F0107069C
000669A 3B6F010D06A0
000669E 3B6F011306A4
00066A2 3B6F011906A8
00066A6 3B6F011F06AC
00066AA 3B6F012506B0
00066AE 3B6F012B06B4
00066B2 3B6F013106B8
00066B6 3B6F013706BC
00066BA 3B6F013D06C0
00066BE 3B6F014306C4
00066C2 3B6F014906C8
00066C6 3B6F014F06CC
00066CA 3B6F015506D0
00066CE 3B6F015B06D4
00066D2 3B6F016106D8
00066D6 3B6F016706DC
00066DA 3B6F016D06E0
00066DE 3B6F017306E4
00066E2 3B6F017906E8
00066E6 3B6F017F06EC
00066EA 3B6F018506F0
00066EE 3B6F018B06F4
00066F2 3B6F019106F8
00066F6 3B6F019706FC
00066FA 3B6F019D0600
00066FE 3B6F01A30604
0006702 3B6F01A90608
0006706 3B6F01AF060C
000670A 3B6F01B50610
000670E 3B6F01BB0614
0006712 3B6F01C10618
0006716 3B6F01C7061C
000671A 3B6F01CD0620
000671E 3B6F01D30624
0006722 3B6F01D90628
0006726 3B6F01DF062C
000672A 3B6F01E50630
000672E 3B6F01EB0634
0006732 3B6F01F10638
0006736 3B6F01F7063C
000673A 3B6F01FD0640
000673E 3B6F01030644
0006742 3B6F01090648
0006746 3B6F010F064C
000674A 3B6F0115065
```

```

006494 7002      moveq.l #2,D0
006496 61000592   bsr    $6A2A
00649A 610003C0   bsr    $685C
00649E 66000360   bne    $6800
0064A2 336D06860000 move.w $686(A5),0(A1)
0064A8 3B7CFFF06A2 move.w #-1,$6A2(A5)
0064AE 6128      bsr    $64D8
0064B0 6600034E   bne    $6800
0064B4 3B6D0696068C move.w $696(A5),$68C(A5)
0064BA 3B7C00010688 move.w #1,$688(A5)
0064C0 6100015C   bsr    $661E
0064C4 246D068E   move.l $68E(A5),A2
0064C8 4A52      tst.w  (A2)
0064CA 67000342   beq    $680E
0064CE 3B7CFFF06A2 move.w #-16,$6A2(A5)
0064D4 6000032A   bra    $6800

*****
0064D8 3B7CFFF606A0 move.w #-10,$6A0(A5)
0064DE 363C0001   move.w #1,D3
0064E2 246D068E   move.l $68E(A5),A2
0064E6 323C003B   move.w #53B,D1
0064EA 103C004E   move.b #54E,D0
0064EE 6100010A   bsr    $65FA
0064F2 3803      move.w D3,D4
0064F4 323C000B   move.w #5B,D1
0064F8 4200      clr.b  D0
0064FA 610000FE   bsr    $65FA
0064FE 323C0002   move.w #2,D1
006502 103C00F5   move.b #5F5,D0
006506 610000F2   bsr    $65FA
00650A 14FC00FE   move.b #5FE,(A2)+

'changed'
Diskette changed
hseek, search for track
flopfail, not found
ctrack, write current track in DSB
currerr to default error
Format track
flopfail, error
spt sectors per track as ccount counter
csect, start with sector 1
verify, verify sector
cdma, list with bad sectors
Bad sector ?
No
currerr to 'Bad Sector'
flopfail, error

fmtrack, format track
defererror, default error number
Start with sector 1
cdma, buffer for track data
60 times
$4E, track header
wmult, write in buffer
Save sector number
12 times
0
wmult, write in buffer
3 times
$F5
wmult, write in buffer
$FE, address mark

```

00650E 14F900000687	move.b	\$687, (A2) +	Track
006514 14F90000068B	move.b	\$68B, (A2) +	Side
00651A 14C4	move.b	D4, (A2) +	Sector
00651C 14FC0002	move.b	#2, (A2) +	Sector size (512 bytes)
006520 14FC00F7	move.b	\$F7, (A2) +	Write checksum
006524 323C0015	move.w	#15, D1	22 times
006528 103C004E	move.b	\$4E, D0	\$4E
00652C 610000CC	bsr	\$5FA	wmult, write in buffer
006530 323C000B	move.w	\$B, D1	12 times
006534 4200	clr.b	D0	0
006536 610000C2	bsr	\$5FA	wmult, write in buffer
00653A 323C0002	move.w	#2, D1	3 times
00653E 103C00F5	move.b	\$F5, D0	\$F5
006542 610000B6	bsr	\$5FA	wmult, write in buffer
006546 14FC00FB	move.b	\$FB, (A2) +	\$FB, data block mark
00654A 323C00FF	move.w	\$FF, D1	256 times
00654E 14ED069A	move.b	\$9A(A5), (A2) +	virgin, initial data in buffer
006552 14ED069B	move.b	\$9B(A5), (A2) +	
006556 51C9FFF6	dbra	D1, \$654E	
00655A 14FC00F7	move.b	\$F7, (A2) +	Next word
00655E 323C0027	move.w	\$27, D1	Write checksum
006562 103C004E	move.b	\$4E, D0	40 times
006566 61000092	bsr	\$5FA	\$4E
00656A D86D0698	add.w	\$98(A5), D4	wmult, write in buffer
00656E B86D0696	cmp.w	\$96(A5), D4	Add interlv, next sector
006572 6F80	ble	\$64F4	spt, largest sector number ?
006574 5243	addq.w	#1, D3	No, next sector
006576 B6D0698	cmp.w	\$98(A5), D3	Start sector plus one
00657A 6F00FF76	ble	\$64F2	interlv
00657E 323C0578	move.w	\$578, D1	Next sector
006582 103C004E	move.b	\$4E, D0	1401 times (until track end)
006586 6172	bsr	\$5FA	\$4E
			wmult, write in buffer



```

006588 13ED0691FFFFF860D      move.b  $91(A5), $FFFF860D      dmaLow
006590 13ED0690FFFFF860B      move.b  $90(A5), $FFFF860B      dmaMid
006598 13ED068FFFFF8609      move.b  $8F(A5), $FFFF8609      dmaHigh
0065A0 3CBC0190                  move.w  #$190, (A6)            Clear DMA status
0065A4 3CBC0090                  move.w  #$90, (A6)
0065A8 3CBC0190                  move.w  #$190, (A6)            Data direction to WRITE
0065AC 3E3C001F                  move.w  #$1F, D7              Sector counter to 31
0065B0 61000412                  bsr     $69C4                 wdiskctl, D7 to 1772
0065B4 3CBC0180                  move.w  #$180, (A6)            Select 1772
0065B8 3E3C00F0                  move.w  #$F0, D7              Format track command
0065BC 61000406                  bsr     $69C4                 wdiskctl, D7 to 1772
0065C0 2E3C00040000             move.l  #$40000, D7           Timeout counter
0065C6 08390005FFFFFA01         btst    #5, $FFFFFA01         mfp gpip, 1772 done ?
0065CE 670C                      beq     $65DC                 Yes
0065D0 5387                      subq.l  #1, D7                Decrement timeout counter
0065D2 66F2                      bne     $65C6                 Not yet timed-out ?
0065D4 61000358                  bsr     $692E                 reset, terminate
0065D8 7E01                      moveq.l #1, D7                Clear 2-bit, error
0065DA 4E75                      rts

0065DC 3CBC0190                  move.w  #$190, (A6)            Select DMA status
0065E0 3016                      move.w  (A6), D0              Read status
0065E2 08000000             btst    #0, D0                DMA error ?
0065E6 67F0                      beq     $65D8                 Yes
0065E8 3CBC0180                  move.w  #$180, (A6)            Select 1772 status register
0065EC 610003EA             bsr     $69D8                 rdiskctl, read registers
0065F0 6100FD9C             bsr     $638E                 errbits, calculate error number
0065F4 C03C0044             and.b   #$44, D0              Test write protect and lost data
0065F8 4E75                      rts

0065FA 14C0                  move.b  D0, (A2)+             Write data in buffer
0065FC 51C9FFFC             dbra    D1, $65FA             Next byte

```

```

0006600 4E75
*****
0006602 61003EE      bsr      $69F2
0006606 70F5         moveq.l # -11,D0
0006608 610015E      bsr      $6768
000660C 610033A      bsr      $6948
0006610 610029C      bsr      $68AE
0006614 66001EA      bne      $6800
0006618 6104         bsr      $661E
000661A 600001F2     bra      $680E
*****
000661E 3B7CFFF506A0   move.w   #-11,$6A0(A5)
0006624 246D068E      move.l   $68E(A5),A2
0006628 06AD0000020068E  add.l   #$200,$68E(A5)
0006630 3B7C00020672   move.w   #2,$672(A5)
0006636 3CBC0084      move.w   #84,(A6)
000663A 3E2D0698      move.w   $688(A5),D7
000663E 6100384      bsr      $69C4
0006642 13ED0691FFFF860D  move.b   $691(A5),$FFFF860D
000664A 13ED0690FFFF860B  move.b   $690(A5),$FFFF860B
0006652 13ED068FFFFF8609  move.b   $68F(A5),$FFFF8609
000665A 3CBC0090      move.w   #$90,(A6)
000665E 3CBC0190      move.w   #$190,(A6)
0006662 3CBC0090      move.w   #$90,(A6)
0006666 3E3C0001     move.w   #1,D7
000666A 61000358      bsr      $69C4
000666E 3CBC0080      move.w   #$80,(A6)
0006672 3E3C0080      move.w   #$80,D7
0006676 6100034C      bsr      $69C4
000667A 2E3C00040000     move.l   #$40000,D7
*****
flopver, verify sector(s)
Change, test for disk change
'read error', as default error
floplock, set parameters
Select
go2track, search for track
flopfail, error
verify1, verify sectors
flopok, done

defererror to 'read error'
cdma, DMA buffer for bad sector list
cdma to next sector
retrycnt, 2 attempts
Select sector register
csect, sector number
wdiskctl,to disk controller
dmalow
dnamid
dmanhigh
Clear DMA status

Data direction to READ
Sector counter to 1
wdiskctl
Select 1772 command register
Read sector command
wdiskctl
Timeout counter

```

006680	08390005FFFFFFA01	btst	#5,\$FFFA01	mfp gpip, 1772 done ?
006688	670A	beq	\$6694	Yes
00668A	5387	subq.l	#1,D7	Decrement timeout counter
00668C	66F2	bne	\$6680	Timed-out yet ?
00668E	6100029E	bsr	\$692E	Reset 1772
006692	6036	bra	\$66CA	Next attempt
006694	3BC00090	move.w	#90,(A6)	Select DMA status register
006698	3016	move.w	(A6),D0	Reat status
00669A	08000000	btst	#0,D0	DMA error ?
00669E	672A	beq	\$66CA	Yes, try again
0066A0	3BC00080	move.w	#80,(A6)	Select 1772 status register
0066A4	61000332	bsr	\$69D8	rdiskctl, read status
0066A8	6100FCE4	bsr	\$638E	erbits, calculate error number
0066AC	C03C001C	and.b	#1C,D0	Test RNF, CRC and lost data
0066B0	6618	bne	\$66CA	Error, next attempt
0066B2	526D0688	addq.w	#1,\$688(A5)	csect, next sector
0066B6	536D068C	subq.w	#1,\$68C(A5)	ccount, decrement sector counter
0066BA	6600FF74	bne	\$6630	Another sector ?
0066BE	04AD0000200068E	sub.l	#200,\$68E(A5)	cdma, Reset DMA pointer
0066C6	4252	clr.w	(A2)	bad sector list mit Null abschlie@en
0066C8	4E75	rts		
0066CA	0C6D00010672	cmp.w	#1,\$672(A5)	retrycnt, 2ns attempt ?
0066D0	6604	bne	\$66D6	No
0066D2	610001A6	bsr	\$687A	reseek
0066D6	536D0672	subq.w	#1,\$672(A5)	Decrement retrycnt
0066DA	6A00FF66	bpl	\$6642	Another attempt ?
0066DE	34ED0688	move.w	\$688(A5),(A2)+	csect, sector number in bad sector list
0066E2	60CE	bra	\$66B2	Next sector

```

*****
0066E4 9BCD      sub.l    A5,A5
0066E6 4DF9FFFF8606  lea     $FFFF8606,A6
0066EC 50ED0680     st      $680(A5)
0066F0 4A6D043E     tst.w   $43E(A5)
0066F4 6670        bne     $6766
0066F6 203900000466  move.l  $466,D0
0066FC 1200        move.b  D0,D1
0066FE C23C0007     and.b   #7,D1
006702 6638        bne     $673C
006704 3C8C0080     move.w   $80,(A6)
006708 E608        lsr.b   #3,D0
00670A C07C0001     and.w   #1,D0
00670E 41ED0674     lea     $674(A5),A0
006712 D0C0        add.w   D0,A0
006714 B079000004A6  cmp.w   $4A6,D0
00671A 6602        bne     $671E
00671C 4240        clr.w   D0
00671E 5200        addq.b  #1,D0
006720 E308        lsl.b   #1,D0
006722 0A000007     eor.b   #7,D0
006726 6100026C     bsr     $6994
00672A 3039FFFF8604  move.w   $FFFF8604,D0
006730 08000006     btst    #6,D0
006734 56D0        sne      (A0)
006736 1002        move.b  D2,D0
006738 6100025A     bsr     $6994
00673C 302D0674     move.w   $674(A5),D0
006740 816D0676     or.w    D0,$676(A5)
006744 4A6D0682     tst.w   $682(A5)
006748 6618        bne     $6762
00674A 6100028C     bsr     $69D8
*****
flopvbl, floppy vertical blank handler
Clear A5
Address of the floppy register
Set motoron flag
Flock, disks busy ?
Yes, do nothing
frclock

Calculate mod 8
Not yet 8th interrupt ?
Select 1772 status register
Use bit 4 as drive number

wpstatus, write protect status table
Index with drive number
nflops, number of floppies

Drive select bit
Shift in position
Invert for hardware
Select drive
dskctl, read 1772 status
Test write rprotect bit
And save
Previous select status
Recreate
wpstatus
Write wplatch
desflg, floppies already deselected?
yes
Raed 1772 status register

```

```

00674E 08000007      btst    #7,D0      Motor on bit set ?
006752 6612         bne     $6766      Yes, then don't deselect
006754 103C0007      move.b  #7,D0      Both drives
006758 6100023A      bsr     $6994      Deselect
00675C 3B7C00010682  move.w  #1,$682(A5)  Set deslflg
006762 426D0680      clr.w  $680(A5)    Clear motoron flag
006766 4E75         rts

*****
006768 48F978F800006A4  movem.l D3-D7/A3-A6,$6A4  flopplock
006770 9BCD         sub.l  A5,A5      regsave
006772 4DF9FFF8606      lea     $FFFF8606,A6      Clear A5
006778 50F900000680      st      $680            Address of the floppy register
00677E 3B4006A0      move.w  D0,$6A0(A5)      Set motoron flag
006782 3B4006A2      move.w  D0,$6A2(A5)      deferror
006786 3B7C0001043E      move.w  #1,$43E(A5)     currerr
00678C 2B6F0008068E      move.l  8(A7),$68E(A5)   flock, disable floppy-vbl routine
006792 3B6F00100684      move.w  16(A7),$684(A5)  cdma
006798 3B6F00120688      move.w  18(A7),$688(A5)  cdev
00679E 3B6F00140686      move.w  20(A7),$686(A5)  csect
0067A4 3B6F0016068A      move.w  22(A7),$68A(A5)  ctrack
0067AA 3B6F0018068C      move.w  24(A7),$68C(A5)  cside
0067B0 3B7C00020672      move.w  #2,$672(A5)     ccount
0067B6 43ED06C8      lea     $6C8(A5),A1      retrycnt
0067BA 4A6D0684      tst.w  $684(A5)         dsb0
0067BE 6704         beq     $67C4            cdev
0067C0 43ED06CC      lea     $6CC(A5),A1      Drive 0 ?
0067C4 7E00         moveq.l #0,D7           dsbl
0067C6 3E2D068C      move.w  $68C(A5),D7      ccount, number of sectors
0067CA E14F      lsl.w  #8,D7            Times 512
0067CC E34F      lsl.w  #1,D7
0067CE 206D068E      move.l  $68E(A5),A0      cdma, start DMA address

```

```

0067D2 DIC7
0067D4 2B490692
0067D8 4A690000
0067DC 6A20
0067DE 61000168
0067E2 42690000
0067E6 610000EC
0067EA 6712
0067EC 7E0A
0067EE 6172
0067F0 6606
0067F2 610000E0
0067F6 6706
0067F8 337CFF000000
0067FE 4E75

*****
006800 7001
006802 61000226
006806 302D06A2
00680A 48C0
00680C 6002

*****
00680E 4280
006810 2F00
006812 3CBC0086
006816 3E290000
00681A 610001A8
00681E 3C3C0010
006822 610000C6
006826 303900000684

add.l D7,A0
move.l A0,$692(A5)
tst.w 0(A1)
bpl $67FE
bsr $6948
clr.w 0(A1)
bsr $68D4
beq $67FE
moveq.l #10,D7
bsr $6862
bne $67F8
bsr $68D4
beq $67FE
move.w #$FF00,0(A1)
rts

Plus length of sector
edma, yields end DMA address
dcurtrack, current track
>= 0 ?
No, select
Set track to zero
Restore, head to track zero
OK ?

Seek track 10
Error ?
Restore
OK ?
Recalibrate, error

*****
006800 7001
006802 61000226
006806 302D06A2
00680A 48C0
00680C 6002

*****
00680E 4280
006810 2F00
006812 3CBC0086
006816 3E290000
00681A 610001A8
00681E 3C3C0010
006822 610000C6
006826 303900000684

moveq.l #1,D0
bsr $6A2A
move.w $6A2(A5),D0
ext.l D0
bra $6810

*****
00680E 4280
006810 2F00
006812 3CBC0086
006816 3E290000
00681A 610001A8
00681E 3C3C0010
006822 610000C6
006826 303900000684

clr.l D0
move.l D0,-(A7)
move.w #586,(A6)
move.w 0(A1),D7
bsr $69C4
move.w #510,D6
bsr $68EA
move.w $684,D0

flopfail, error in floppy routine
Media change to 'unsure'
Set
currerr

flopok, error-free floppy routine

ok
Select 1772
Get track number
wdiskctl, send to disk controller
seek command
flopems
cdev, drive number

```

```

00682C E548          lsl.w    #2,D0          As index
00682E 41F900000678  lea      $678,A0          acctim
006834 21AD04BA0000  move.l   $4BA(A5),0(A0,D0,w)  200 Hz counter as last access time
00683A 0C790001000004A6 cmp.w    #1,$4A6          nflaps
006842 6606          bne      $684A
006844 216D04BA0004  move.l   $4BA(A5),4(A0)      20 Hz counter for other drives
00684A 201F          move.l   (A7)+,D0      Restore error number
00684C 4CF978F8000006A4 movem.l  $6A4,D3-D7/A3-A6    regsave
006854 42790000043E  clr.w    $43E              Clear flock, release vbl routine
00685A 4E75          rts

*****
00685C 3E3900000686  move.w   $686,D7          hseek, head to track
006862 33FCFFFA000006A2 move.w   #-10,$6A2        ctrack
00686A 3CBC0086          move.w   #$86,(A6)      currerr to 'seek error'
00686E 61000154          bsr      $69C4          Pass track number
006872 3C3C0010          move.w   #$10,D6        wdiskctl
006876 60000072          bra      $68EA          Seek command
                                flopcnds

*****
00687A 33FCFFFA000006A2 move.w   #-10,$6A2        resseek, home and seek
006882 6150          bsr      $68D4          currerr to 'seek error'
006884 664C          bne      $68D2          Restore
006886 42690000          clr.w    0(A1)          Error ?
00688A 3CBC0082          move.w   #$82,(A6)      Current track to zero
00688E 4247          clr.w    D7              Select track register
006890 61000132          bsr      $69C4          Track zero
006894 3CBC0086          move.w   #$86,(A6)      wdiskctl
006898 3E3C0005          move.w   #5,D7          Select data register
00689C 61000126          bsr      $69C4          Track 5
0068A0 3C3C0010          move.w   #$10,D6        wdiskctl
0068A4 6144          bsr      $68EA          Seek command
                                flopcnds

```

```

0068A6 662A      bne      $68D2
0068A8 337C00050000  move.w  #5,0(A1)

*****
0068AE 33FCFFFA00006A2  move.w  #-10,$6A2
0068B6 3CEC0086      move.w  #$86,(A6)
0068BA 3E2D0686      move.w  $686(A5),D7
0068BE 61000104      bsr      $69C4
0068C2 7C14      moveq.l  #$14,D6
0068C4 6124      bsr      $68EA
0068C6 660A      bne      $68D2
0068C8 336D06860000  move.w  $686(A5),0(A1)
0068CE CE3C0018      and.b   #$18,D7
0068D2 4E75      rts

*****
0068D4 4246      clr.w   D6
0068D6 6112      bsr      $68EA
0068D8 660E      bne      $68E8
0068DA 08070002      btst    #2,D7
0068DE 0A3C0004      eor.b   #4,SR
0068E2 6604      bne      $68E8
0068E4 42690000      clr.w   0(A1)
0068E8 4E75      rts

*****
0068EA 30290002      move.w  2(A1),D0
0068EE C03C0003      and.b   #3,D0
0068F2 8C00      or.b    D0,D6
0068F4 2E3C00040000  move.l  #$40000,D7
0068FA 3CBC0080      move.w  #$80,(A6)
0068FE 610000D8      bsr      $69D8

Error
Track number to 5

go2track, search for trak
current to 'seek error'
Select data register
ctrack, load track number
wdiskctl
Seek with verify command
flopcmds
Error
ctrack, Save track number
Test RNF, CRC lost data

restore, seek track zero
restore command
flopcmds
Error
Test track zero bit
Z-Flag invertieren
nicht Track Null ?
current Track auf Null

flopcmds
Seek rate
Bits 0 and 1
OR in command word
Timeout counter
Select 1772 register
rdiskctl

```



```

006902 08000007      btst    #7,D0
006906 6606          bne     $690E
006908 2E3C0060000   move.l  #$6000,D7
00690E 610000AA       bsr     $69BA
006912 5387          subq.l  #1,D7
006914 6712          beq     $6928
006916 08390005FFFFFA01 btst    #5,$FFFA01
00691E 66F2          bne     $6912
006920 610000AC       bsr     $69CE
006924 4246          clr.w  D6
006926 4E75          rts

006928 6104          bsr     $692E
00692A 7C01          moveq.l #1,D6
00692C 4E75          rts

*****
00692E 3C8C0080       move.w  #$80,(A6)
006932 3E3C00D0       move.w  #$D0,D7
006936 6100008C       bsr     $69C4
00693A 3E3C000F       move.w  #$F,D7
00693E 51CFFFE       dbra    D7,$693E
006946 4E75          rts

*****
006948 426D0682       clr.w  $682(A5)
00694C 302D0684       move.w  $684(A5),D0
006950 5200          addq.b  #1,D0
006952 E308          lsl.b   #1,D0
006954 806D068A       or.w    $68A(A5),D0
006958 0A000007       eor.b   #7,D0
00695C C03C0007       and.b   #7,D0

Motor on ?
Yes
Else longer timeout
wdiskctl6, write command in D6
Decrement timeout counter
Timed-out ?
mfp gpip, 1772 done ?
No, wait
rdiskctl7
OK

Reset 1772
Error

Reset 1772, reset floppy controller
Select command register
Reset command
wdiskctl
Delay counter
Timed-out ? read status

select, select drive and side
Clear desiflg
cdev, drive number

Calculate bit number
cside, side in bit 0
Invert bits for hardware

```

331

```

0069CC 6012      bra      $69E0      Delay loop for disk controller

*****
0069CE 6110      bsr      $69E0      rdiskctl7
0069D0 3E39FFFF8604  move.w $FFFF8604,D7      Delay loop for disk controller
0069D6 6008      bra      $69E0      dskctl
                                           Delay loop for disk controller

*****
0069D8 6106      bsr      $69E0      rdiskctl
0069DA 3039FFFF8604  move.w $FFFF8604,D0      Delay loop for disk controller
0069E0 40E7      move.w SR, -(A7)      dskctl
0069E2 3F07      move.w D7, -(A7)      Save status
0069E4 3E3C0020      move.w #$20,D7      Save D7
0069E8 51CFFFE      dora      D7,$69E8      Counter
0069EC 3E1F      move.w (A7)+,D7      Delay loop
0069EE 46DF      move.w (A7)+,SR      D7 back
0069F0 4E75      rts              Status back

*****
0069F2 0C79000100004A6  cmp.w #1,$4A6      change, tets disk change
0069FA 662C      bne      $6A28      nflops
0069FC 302F0010      move.w 16(A7),D0      None or 2 floppies, done
006A00 B07900004692  cmp.w $4692,D0      Drive number
006A06 671C      beq      $6A24      Equals diskette number ?
006A08 3F00      move.w D0,-(A7)      Yes
006A0A 3F3CFFFE      move.w #-17,-(A7)      Drive number
006A0E 6100EB4C      bsr      $555C      'insert disk'
006A12 584F      addq.w #4,A7      Critical error handler
006A14 33FCFFFF0000676  move.w $FFFF,$676      Correct stack pointer
006A1C 33EF001000004692  move.w 16(A7),$4692      wplatch, Status for both drives unsure
006A24 426F0010      clr.w 16(A7)      Save diskette number
006A28 4E75      rts              Drive number to zero

```

```

*****
006A2A 41F900003E2A      lea     $3E2A,A0
006A30 1F00              move.b D0,-(A7)
006A32 302D0684          move.w $684(A5),D0
006A36 119F0000          move.b (A7)+,0(A0,D0.w)
006A3A 4E75              rts

*****
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

*****
006A46 4BF900000000      lea     $0,A5
006A4C 41ED0A43          lea     $A43(A5),A0
006A50 610000DE          bsr     $6B30
006A54 04000050          sub.b  #80,D0
006A58 1400              move.b D0,D2
006A5A E982              asl.l  #4,D2
006A5C 610000D2          bsr     $6B30
006A60 D400              add.b  D0,D2
006A62 EB82              asl.l  #5,D2
006A64 610000CA          bsr     $6B30
006A68 D400              add.b  D0,D2

*****
jdstime, IKED format in DOS format
Clear A5
Pointer to clock-time buffer
bcdbin
Subtract offset of 80
Year
Shift in position
bcdbin
Add month
And shift in position
bcdbin
Add day

```

```

006A6A EB82          asl.l    #5,D2          And shift in position
006A6C 610000C2      bsr      $6B30        bcdbin
006A70 D400          add.b    D0,D2          Add hour
006A72 ED82          asl.l    #6,D2          And shift in position
006A74 610000BA      bsr      $6B30        bcdbin
006A78 D400          add.b    D0,D2          Add minute
006A7A EB82          asl.l    #5,D2          And shift in position
006A7C 610000B2      bsr      $6B30        bcdbin
006A80 E208          lsr.b    #1,D0          2-second resolution
006A82 D400          add.b    D0,D2          And add seconds
006A84 2B420A4C      move.l   D2,$A4C(A5)    Save new time
006A88 1B7C0000A8E   move.b   #0,$A8E(A5)    Clear handshake flag
006A8E 4E75          rts

*****
006A90 1B7CFFFF0A8E  move.b   #-1,$A8E(A5)    gettime, get current clock time and date
006A96 123C001C      move.b   #1C,D1        Request handshake flag for time
006A9A 61000234      bsr      $6CD0          get time of day command
006A9E 4A2D0A8E      tst.b    $A8E(A5)        Send
006AA2 66FA          bne     $A9E            New time arrived ?
006AA4 202D0A4C      move.l   $A4C(A5),D0    No, wait
006AA8 4E75          rts          Get time in D0

*****
006AAA 2B6F00040A50  move.l   4(A7),$A50(A5)    settime, set clock time and date
                                Pass time

*****
006AB0 41F900000A5A  lea      $A5A,A0          ikbdttime
006AB6 242D0A50      move.l   $A50(A5),D2    Pointer to end of time buffer
006ABA 1002          move.b   D2,D0          Get time to convert
006ABC 0200001F      and.b    #31,D0          To d0
006AC0 E300          asl.b    #1,D0          Isolate bits 0-4, seconds
                                2-second resolution

```

006AC2 6154	bsr	\$6B18	Convert
006AC4 EA8A	lsl.l	#5,D2	Minute
006AC6 1002	move.b	D2,D0	
006AC8 0200003F	and.b	#63,D0	Isolate bits 0-5
006ACC 614A	bsr	\$6B18	Convert
006ACE EC8A	lsl.l	#6,D2	Hours
006AD0 1002	move.b	D2,D0	
006AD2 0200001F	and.b	#31,D0	Isolate bits 0-4
006AD6 6140	bsr	\$6B18	Convert
006AD8 EA8A	lsl.l	#5,D2	Day
006ADA 1002	move.b	D2,D0	
006ADC 0200001F	and.b	#31,D0	Isolate bits 0-4
006AE0 6136	bsr	\$6B18	Convert
006AE2 EA8A	lsl.l	#5,D2	Month
006AE4 1002	move.b	D2,D0	
006AE6 0200000F	and.b	#15,D0	Isolate bits 0-3
006AEA 612C	bsr	\$6B18	Convert
006AEC E88A	lsl.l	#4,D2	Year
006AEE 1002	move.b	D2,D0	
006AF0 0200007F	and.b	#57F,D0	Isolate bits 0-7
006AF4 6122	bsr	\$6B18	Convert
006AF6 06100080	add.b	#\$80,(A0)	Add offset
006AFA 123C001B	move.b	#\$1B,D1	Set Time Of Day command
006AFE 610001D0	bsr	\$6CD0	Send to IKBD
006B02 7605	moveq.l	#5,D3	Number of bytes to send
006B04 45F90000A54	lea	\$A54,A2	Address of the parameter block
006B0A 610001E4	bsr	\$6CF0	Send
006B0E 123C001C	move.b	#\$1C,D1	Get Time Of Day command
006B12 610001BC	bsr	\$6CD0	Send to IKBD
006B16 4E75	rts		

```

*****
006B18 7200          moveq.l #0,D1
006B1A 760A          moveq.l #10,D3
006B1C 9003          sub.b D3,D0
006B1E 6B04          bml $6B24
006B20 5201          addq.b #1,D1
006B22 60F8          bra $6B1C
006B24 0600000A      add.b #10,D0
006B28 E901          asl.b #4,D1
006B2A D001          add.b D1,D0
006B2C 1100          move.b D0,~(A0)
006B2E 4E75          rts

*****
006B30 7000          moveq.l #0,D0
006B32 1010          move.b (A0),D0
006B34 E808          lsr.b #4,D0
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
006B44 D041          add.w D1,D0
006B46 4E75          rts

*****
006B48 70FF          moveq.l #-1,D0
006B4A 1439FFFFFFC04 move.b $FFFFFFC04,D2
006B50 08020001      btst #1,D2
006B54 6602          bne $6B58
006B56 7000          moveq.l #0,D0

*****
binbcd, convert byte to BCD
Ten's counter
Load 10
Subtract 10

Increment ten's counter

Generate one's counter
Tens in upper nibble
And add ones
Write result in buffer

bcdbin, convert BCD to binary
BCD byte
Ten's place

Isolate one's place
plus ten's place

midiost, MIDI output status
Default to OK
Read MIDI-ACIA status
And test
OK
Status not OK

```

337



```

*****
006BA4 61E2      bsr    $6B88
006BA6 4A40      tst.w  D0
006BA8 67FA      beq    $6BA4
006BA4 40E7      move.w  SR, ~(A7)
006BAC 007C0700  or.w   #$700,SR
006BB0 32280006  move.w  6(A0),D1
006BB4 B2680008  cmp.w   8(A0),D1
006BB8 6716      beq    $6BD0
006BBA 5241      addq.w  #1,D1
006BBC B2680004  cmp.w   4(A0),D1
006BC0 6502      bcs    $6BC4
006BC2 7200      moveq.l #0,D1
006BC4 22680000  move.l  0(A0),A1
006BC8 10311000  move.b  0(A1,D1.w),D0
006BCC 31410006  move.w  D1,6(A0)
006BD0 46DF      move.w  (A7)+,SR
006BD2 4E75      rts

*****
006BD4 242D04BA  move.l  $4BA(A5),D2
006BD8 94AD0A80  sub.l   $A80(A5),D2
006BDC 0C82000003E8  cmp.l   #1000,D2
006BE2 6518      bcs    $6BFC
006BE4 242D04BA  move.l  $4BA(A5),D2
006BE8 6172      bsr    $6C5C
006BEA 4A40      tst.w  D0
006BEC 6618      bne    $6C06
006BEE 262D04BA  move.l  $4BA(A5),D3
006BF2 9682      sub.l   D2,D3
006BF4 0C8300001770  cmp.l   #6000,D3
006BFA 6DEC      blt    $6BE8

*****
midin, get character from MIDI
midstat
Character ready ?
No
Save status
IPL 7, disable interrupts
Head index
Compare with tail index
Increment head index
Greater than buffer size ?
No
Begin at buffer start again
Pointer to MIDI buffer
Get character from buffer
Save new head index
Restore status

*****
lout, output character to Centronics
200 Hz counter
Minus printer timeout
Waited more than 5 seconds ?
Yes, time out
200 Hz counter, start time for this char
Get bust status
And test
Printer ready ?
200 Hz counter
Minus start time
30 seconds ?
Not yet reached

```

006BFC 7000	moveq.l #0,D0	Flag for time out
006BFE 2B6D04BA0A80	move.l \$4BA(A5), \$A80(A5)	200 Hz counter as last time-out time
006C04 4E75	rts	
006C06 40C3	move.w SR, D3	Save status
006C08 007C0700	or.w #\$700,SR	IPL 7, no interrupts
006C0C 7207	moveq.l #7,D1	Mixer
006C0E 61000E28	bsr \$7A38	Select registers
006C12 00000080	or.b #\$80,D0	Port B to output
006C16 7287	moveq.l #\$87,D1	Write enable
006C18 61000E1E	bsr \$7A38	
006C1C 46C3	move.w D3,SR	Restore status
006C1E 302F0006	move.w 6(A7),D0	Character to output
006C22 728F	moveq.l #\$8F,D1	Write to port B
006C24 61000E12	bsr \$7A38	
006C28 610C	bsr \$6C36	Strobe low
006C2A 6104	bsr \$6C30	Strobe high
006C2C 70FF	moveq.l #-1,D0	Flag for OK
006C2E 4E75	rts	
*****		
006C30 7420	moveq.l #\$20,D2	Strobe high
006C32 60000E46	bra \$7A7A	Bit 5
*****		
006C36 74DF	moveq.l #\$DF,D2	Strobe low
006C38 60000E66	bra \$7AA0	Bit 5
*****		
006C3C 7207	moveq.l #7,D1	listin, Get character from parallel port
006C3E 61000DF8	bsr \$7A38	Mixer
006C42 0200007F	and.b #\$7F,D0	Select register
006C46 7287	moveq.l #\$87,D1	Port B to input

```

006C48 61000DEE      bsr      $7A38
006C4C 61E2          bsr      $6C30
006C4E 610C          bsr      $6C5C
006C50 4A40          tst.w    D0
006C52 66FA          bne      $6C4E
006C54 61E0          bsr      $6C36
006C56 720F          moveq.l #15,D1
006C58 6000DDE      bra      $7A38

*****
006C5C 41F9FFFFFA01  lea      $FFFFFA01,A0
006C62 70FF          moveq.l #-1,D0
006C64 082800000000  btst     #0,0(A0)
006C6A 6702          beq      $6C6E
006C6C 7000          moveq.l #0,D0
006C6E 4E75          rts

*****
006C70 41ED09D0      lea      $9D0(A5),A0
006C74 70FF          moveq.l #-1,D0
006C76 45E80006      lea      6(A0),A2
006C7A 47E90008      lea      8(A0),A3
006C7E B54B          cmpm.w   (A3)+,(A2)+
006C80 6602          bne      $6C84
006C82 7000          moveq.l #0,D0
006C84 4E75          rts

*****
006C86 61E8          bsr      $6C70
006C88 4A40          tst.w    D0
006C8A 67FA          beq      $6C86
006C8C 610005D8      bsr      $7266

Strobe high = not busy
Get parallel port status

Wait until character arrives
Strobe low = busy
Select port B
Read byte from port

lstat, parallel port status
Address of the MFP
Default to ok
Test busy
OK
Still busy

auxistat, RS232 input status
iorec for RS232
Default to ok
Head index
Tail Index
Buffer leer ?
No
No character there

auxin, get character RS-232
auxin, character ready ?

No, wait
rs232get, get character

```

```

006C90 024000FF      and.w  #$FF,D0
006C94 4E75          rts

*****
006C96 41ED09D0      lea    $9D0(A5),A0
006C9A 70FF          moveq.l  #-1,D0
006C9C 34280016      move.w  22(A0),D2
006CA0 6100087C      bsr    $751E
006CA4 B4680014      cmp.w  20(A0),D2
006CA8 6602          bne    $6CAC
006CAA 7000          moveq.l  #0,D0
006CAC 4E75          rts

*****
006CAE 322F0006      move.w  6(A7),D1
006CB2 61000556      bsr    $720A
006CB6 65F6          bcs    $6CAE
006CB8 4E75          rts

*****
006CBA 70FF          moveq.l  #-1,D0
006CBC 1439FFFFFC00   move.b  $FFFFFC00,D2
006CC2 08020001      btst   #1,D2
006CC6 6602          bne    $6CCA
006CC8 7000          moveq.l  #0,D0
006CCA 4E75          rts

*****
006CCC 322F0006      move.w  6(A7),D1
006CD0 43F9FFFFFC00   lea    $FFFFFC00,A1
006CD6 14290000      move.b  0(A1),D2
006CDA 08020001      btst   #1,D2

```

Isolate bits 0-7

auxostat, RS232 output status  
iorec for RS232  
Default to ok  
Tail index  
Test for wrap around  
Compare with head index  
Not equal  
No more space in buffer

auxout, RS232 output routine  
Get data byte  
rs232put, and output  
Try again

ikbdost, IKBD output status  
Default to ok  
IKBD ACIA status  
Test  
OK  
Not ready

ikbdwc, send character to IKBD  
Get byte  
Address of the keyboard ACIA  
Get status  
Ready

```

006CDE 67F6      beq     $6CD6
006CE0 13410002  move.b  D1,2(A1)
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

*****
constat, keyboard input status
iorec for keyboard
Default to ok
Head index
Tail index
Buffer leer ?
No
No characters there

conin, get character from keyboard
constat, key pressed ?

No, wait
Save status
IPL 7, disable interrupts
Head index
Compare with tail index

```

```

006D26 5441      addq.w #2,D1
006D28 B280004   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
006D40 E048      lsr.w #8,D0
006D42 46DF      move.w (A7)+,SR
006D44 4E75      rts

*****
006D46 70FF      moveq.l #-1,D0
006D48 4E75      rts

*****
006D4A 082D0020484 btst #2,$484(A5)
006D50 670E      beq $6D60
006D52 2B7C00007D5A0A86 move.l #$7D5A,$A86(A5)
006D5A 1B7C00000A8A move.b #$0,$A8A(A5)
006D60 4E75      rts

*****
006D62 001B313233343536 dc.b $00,esc,'1','2','3','4','5','6'
006D6A 373839309E270809 dc.b '7','8','9','0','@',' ',' ','bs,tab
006D72 71776572747A7569 dc.b 'q','w','e','r','t','z','u','i'
006D7A 6F70812B0D006173 dc.b 'o','p','[','+',',',cx,$00,'a','s'
006D82 646667686A6B6C94 dc.b 'd','f','g','h','j','k','l','\
006D8A 8423007E79786376 dc.b ']' ,#,$00,'1','y','x','c','v'
006D92 626E6D2C2E2D0000 dc.b 'b','n','m',' ',' ',' ','-','$00,$00

```

Head index + 2  
Greater than or equal buffer size ?  
No  
Buffer pointer back to start  
Pointer to keyboard buffer  
Get character and scan code  
Save new head index  
Scan code to bits 16-23  
ASCII code to bits 0-7  
Restore status

conoutst, console output status  
Always OK

ringbel, tone after CTRL G  
Tone enabled ?  
No  
Pointer to sound table for bell  
Start sound timer

Keyboard table, unshifted

006D9A	0020000000000000	dc.b	\$00,'1,\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00
006DA2	0000000000000000	dc.b	\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00
006DAA	00002D00000002B00	dc.b	\$00,\$00,'1-',\$00,\$00,\$00,'1',\$00
006DB2	0000007F00000000	dc.b	\$00,\$00,\$00,del,\$00,\$00,\$00,\$00,\$00,\$00
006DBA	0000000000000000	dc.b	\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00
006DC2	3C000028292F2A37	dc.b	'<',\$00,\$00,'8','1','7','1','1','1','1','7,'
006DCA	3839343536313233	dc.b	'8','9','4','5','6','1','1','2','1','3,'
006DD2	302E0D0000000000	dc.b	'0','1',cr,\$00,\$00,\$00,\$00,\$00,\$00,\$00
006DDA	0000000000000000	dc.b	\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00,\$00

\*\*\*\*\*  
Keyboard table, shifted

006DE2	001E2122DD242526	dc.b	\$00,esc,'1','1','1','1','1','\$','&','&'
006DEA	2F28293D3F600809	dc.b	','('','')','='','?','1','bs,tab
006DF2	51574552545A5549	dc.b	'Q','W','E','R','T','Z','U','I'
006DFA	4F509A2A0D0004153	dc.b	'O','P','[','\','*','cr,\$00,'A','S'
006E02	44647484A584C99	dc.b	'D','F','G','H','J','K','L','\'
006E0A	8E5E007C59584356	dc.b	','\','\$00,'1','Y','X','C','V'
006E12	423E4D3B3A3F0000	dc.b	'B','N','M','Z',' ','1','\$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','14','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

\*\*\*\*\*  
Keyboard table, Caps Lock

06E62	001B3132333343536	dc.b	\$00,esc,'1','2','3','4','5','6'
06E6A	373839309E270809	dc.b	'7','8','9','0','@',' ','bs,tab
06E72	51574552545A5549	dc.b	'Q','W','E','R','T','Z','U','I'

```

006E7A 4F509A2B0D0004153      dc.b 'O','P','[','+',',',cr,$00,'A','S'
006E82 444647484A4B4C99      dc.b 'D','F','G','H','J','K','L','\','
006E8A 8E23007E59684356      dc.b ']',',',,$00,'1','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 00002D00000002B00      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
006EE8 7000      moveq.l #0,D0
006EEA 01C80000      movep.l D0,0(A0)
006EEE 01C80008      movep.l D0,8(A0)
006EF2 01C80010      movep.l D0,$10(A0)
006EF6 117C00480016      move.b #$48,22(A0)
006EFC 3B7C11110A84      move.w #$1111,$A84(A5)
006F02 3B7C00140442      move.w #20,$442(A5)
006F08 7002      moveq.l #2,D0
006F0A 7250      moveq.l #80,D1
006F0C 343C00C0      move.w #$C0,D2
006F10 6100017E      bsr $7090
006F14 45F900007C5C      lea $7C5C,A2
006F1A 7005      moveq.l #5,D0
006F1C 61000228      bsr $7146
006F20 7003      moveq.l #3,D0

*****
initnfp, initialize MFP 68901
Address of the MFP
Initialize registers with zero
grip to iera
ierb to isrb
isrb to vr
MFP Non-autovector # to $48, set S-bit

Timer ms to 20 milliseconds
Select timer C
/64 for 200 Hz
192
Initialize timer and interrupt vector
Timer C interrupt routine
Timer C interrupt number
initint, install interrupt
Select timer D

```



006F22 7201	moveq.l #1,D1	/4 for 9600 Baud
006F24 7402	moveq.l #2,D2	9600 baud
006F26 6100168	bsr \$7090	Initialize timer and interrupt vector
006F2A 203C00980101	move.l #980101,D0	\$00, \$98, \$01, \$01
006F30 01C80026	moveq.l D0,\$26(A0)	To scr, ucr, rsr, tsr
006F34 61000B3A	bsr \$7A70	DIR on
006F38 61000B2E	bsr \$7A68	RTS on
006F3C 41ED09D0	lea \$9D0(A5),A0	Pointer to iorec for RS232
006F40 43F90000705E	lea \$705E,A1	Start data for iorec
006F46 7021	moveq.l #33,D0	34 bytes
006F48 610000EC	bsr \$7036	Copy into RAM
006F4C 41ED0A00	lea \$A00(A5),A0	Pointer to iorec MIDI
006F50 43F900007050	lea \$7050,A1	Start data for iorec
006F56 700D	moveq.l #13,D0	14 bytes
006F58 610000DC	bsr \$7036	Copy into RAM
006F5C 203C0000759C	move.l #759C,D0	Keyboard and MIDI error vector
006F62 2B400A12	move.l D0,\$A12(A5)	Pointer to error routine keyboard
006F66 2B400A16	move.l D0,\$A16(A5)	Pointer to error routine MIDI
006F6A 2B7C000079C60A0E	move.l #79C6,\$A0E(A5)	MIDI interrupt vector
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 stop bit, no parity
006F92 1B7C000070484	move.b #7,\$484(A5)	Key click, 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	Clear sound variables
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 to high
006FC8 1B7C00F0A7E	move.b #\$F,\$A7E(A5)	
006FCE 1B7C00020A7F	move.b #2,\$A7F(A5)	
006FD4 41ED09F2	lea \$9F2(A5),A0	Pointer to iorec to keyboard
006FD8 43F900007042	lea \$7042,A1	Start data for iorec
006FDE 700D	moveq.l #13,D0	14 bytes
006FE0 6154	bsr \$7036	Copy into RAM
006FE2 6100C0E	bsr \$7BF2	Pointer to BIOS keyboard table
006FE6 13FC0003FFFFFC00	move.b #3,\$FFFFFC00	Reset keyboard ACIA
006FEE 13FC0096FFFFFC00	move.b #96,\$FFFFFC00	/64, 8 bits, 1 top bit, no arity
006FF6 267C00007080	move.l #\$7080,A3	Pointer to MFP Interrupt vectors
006FFC 7203	moveq.l #3,D1	Initialize four vectors
006FFE 2401	move.l D1,D2	
007000 2001	move.l D1,D0	Interrupt number
007002 06000009	add.b #9,D0	Plus offset
007006 E582	asl.l #2,D2	
007008 24732000	move.l 0(A3,D2.w),A2	Get vector from table
00700C 61000138	bsr \$7146	initint, install interrupt
007010 51C9FFEC	dbra D1,\$6FFE	Next vector
007014 45ED752A	lea \$752A(A5),A2	MIDI and keyboard vector
007018 7006	moveq.l #6,D0	Vector number 6
00701A 6100012A	bsr \$7146	initint, install interrupt vector
00701E 45ED73C0	lea \$73C0(A5),A2	CTS interrupt routine
007022 7002	moveq.l #2,D0	Vector number 2
007024 61000120	bsr \$7146	initint, install interrupt
007028 247C0000703E	move.l #\$703E,A2	Pointer to init data for IKBD
00702E 7603	moveq.l #3,D3	4 bytes
007030 6100FCDE	bsr \$6CF0	Send string to IKBD
007034 4E75	rts	

```

007036 10D9      move.b  (A1)+, (A0)+
007038 51C8FFFC  dbra    D0, $7036
00703C 4E75      rts

*****
00703E 8001121A  dc.b    $80, $01, $12, $1A
Reset keyboard, disable mouse and joystick

*****
007042 000008D0  dc.l    $8D0
007046 0080      dc.w    128
007048 0000      dc.w    0
00704A 0000      dc.w    0
00704C 0020      dc.w    32
00704E 0060      dc.w    96
iorec for keyboard
Keyboard buffer
Length of the keyboard buffer
Head index
Tail index
Low water mark, 1/4 buffer length
High water mark, 3/4 buffer length

*****
007050 00000950  dc.l    $950
007054 0080      dc.w    128
007056 0000      dc.w    0
007058 0000      dc.w    0
00705A 0020      dc.w    32
00705C 0060      dc.w    96
iorec for MIDI
MIDI buffer
Length of the MIDI buffer
Head index
Tail index
Low water mark, 1/4 buffer length
High water mark, 3/4 buffer length

*****
00705E 000006D0  dc.l    $6D0
007062 0100      dc.w    256
007064 0000      dc.w    0
007066 0000      dc.w    0
007068 0040      dc.w    64
00706A 00C0      dc.w    192
00706C 000007D0  dc.l    $7D0
007070 0100      dc.w    256
iorec for RS232
RS232 input buffer
Length of the input buffer
Head index
Tail index
Low water mark, 1/4 buffer length
High water mark, 3/4 buffer length
RS232 output buffer
Length of the output buffer

```

```

007072 0000      dc.w      0      Head index
007074 0000      dc.w      0      Tail index
007076 0040      dc.w      64      Low water mark, 1/4 buffer length
007078 00C0      dc.w      192     High water mark, 3/4 buffer length
00707A 00        dc.b      0      rsrbyte, receiver status
00707B 00        dc.b      0      tsrbyte, transmitter status
00707C 00        dc.b      0      rxoff,
00707D 00        dc.b      0      txoff
00707E 01        dc.b      1      rsmode, XON/XOFF mode
00707F 00        dc.b      0      filler

*****
007080 00007426      dc.l      $7426      Interrupt vectors for 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, initialize timer in MFP
007094 207CFFFFFA01      move.l   #$FFFFFA01,A0      Save registers
00709A 267C00007124      move.l   #$7124,A3      Address MFP
0070A0 247C00007128      move.l   #$7128,A2
0070A6 615A          bsr        $7102
0070AB 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)
0070F2 B4303000          cmp.b  0(A0,D3.w),D2
0070F6 66F6              bne     $70EE
0070F8 C749              exg     A3,A1
0070FA 8313              or.b   D1,(A3)
0070FC 4CDF0F1F          movem.l (A7)+,D0-D4/A0-A3
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

```

Write data in MFP  
And compare  
Until match

Restore registers

```

007120 0E0E1010          dc.b      14,14,16,16
007124 12121414          dc.b      18,18,20,20

007128 DFFEDFEF          dc.b      $DF,$FE,$DF,$FE
00712C 181A1C1C          dc.b      $18,$1A,$1C,$1C
007130 0008FF8          dc.b      0,0,$8F,$F8
007134 1E202224          dc.b      $1E,$20,$22,$24

*****
007138 302F0004          move.w    4(A7),D0
00713C 246F0006          move.l    6(A7),A2
007140 0280000000F       and.l     #15,D0

*****
007146 48E7E0E0          movem.l   D0-D2/A0-A2,-(A7)
00714A 6120             bsr        $716C
00714C 2400             move.l    D0,D2
00714E E542             asl.w     #2,D2
007150 068200000100      add.l     #$100,D2
007156 2242             move.l    D2,A1
007158 228A             move.l    A2,(A1)
00715A 614A             bsr        $71A6
00715C 4CDF0707          movem.l   (A7)+,D0-D2/A0-A2
007160 4E75             rts

*****
007162 302F0004          move.w    4(A7),D0
007166 02800000000F       and.l     #15,D0
00716C 48E7C0C0          movem.l   D0-D1/A0-A1,-(A7)
007170 41F9FFFFFA01      lea        $FFFFFA01,A0
007176 43E80012          lea        18(A0),A1
00717A 614A             bsr        $71C6

mfpint, set MFP interrupt vector
Interrupt number
Interrupt vector
Number 0-15, long word

initint, set MFP interrupt vector
Save registers
Disable interrupts
Vector number
As index for long word
Plus base of the MFP vectors
Vector address
Set new vector
Enable interrupts
Restore registers

disint, disable MFP interrupts
Get interrupt number
Convert to long word index
Save registers
Address the MFP
Address imra
Calculate bit number to clear

```

00717C 0391	bclr	D1, (A1)	And clear bit
00717E 43E80006	lea	6(A0), A1	Address iera
007182 6142	bsr	\$71C6	Calculate bit number to clear
007184 0391	bclr	D1, (A1)	And clear bit
007186 43E8000A	lea	10(A0), A1	Address ipra
00718A 613A	bsr	\$71C6	Calculate bit number to clear
00718C 0391	bclr	D1, (A1)	And clear bit
00718E 43E8000E	lea	14(A0), A1	Address isra
007192 6132	bsr	\$71C6	Calculate bit number to clear
007194 0391	bclr	D1, (A1)	And clear bit
007196 4CDF0303	movem.l	(A7)+, D0-D1/A0-A1	Restore registers
00719A 4E75	rts		
*****			
00719C 302F0004	move.w	4(A7), D0	jenabint, enable MFP interrupts
0071A0 0280000000F	and.l	#15, D0	Vector number
0071A6 48E7C0C0	movem.l	D0-D1/A0-A1, -(A7)	In long word index
0071AA 41F9FFFA01	lea	\$FFFA01, A0	Restore registers
0071B0 43E80006	lea	6(A0), A1	Address MFP
0071B4 6110	bsr	\$71C6	Address iera
0071B6 03D1	bset	D1, (A1)	Calculate bit number to set
0071B8 43E80012	lea	18(A0), A1	And set bit
0071BC 6108	bsr	\$71C6	Address imra
0071BE 03D1	bset	D1, (A1)	Calculate bit number to set
0071C0 4CDF0303	movem.l	(A7)+, D0-D1/A0-A1	And set bit
0071C4 4E75	rts		Restore registers
*****			
0071C6 1200	move.b	D0, D1	bselect, calculate bit and register number
0071C8 0C000008	cmp.b	#8, D0	Save interrupt number
0071CC 6D02	blt	\$71D0	Greater than 8 ?
			No

```

0071CE 5141      subq.w  #8,D1      Else subtract offset
0071D0 0C000008  cmp.b   #8,D0      Greater than 8 ?
0071D4 6C02      bge    $71D8      Yes
0071D6 5449      addq.w  #2,A1      Pointer from a to b register
0071D8 4E75      rts

*****
0071DA 41F9000009D0  lea    $9D0,A0      rs232ptr
0071E0 43F9FFFFFA01  lea    $FFFFFA01,A1      Pointer to RS232 iorec
0071E6 4E75      rts      Pointer to MFP

*****
0071E8 34280008      move.w  8(A0),D2      rs232lbuf
0071EC 36280006      move.w  6(A0),D3      Tail index
0071F0 B443      cmp.w   D3,D2      Head index
0071F2 6204      bhi    $71F8      Head > tail ?
0071F4 D4680004      add.w  4(A0),D2      No
0071F8 9443      sub.w  D3,D2      Add buffer size
0071FA 4E75      rts      Tail - head

*****
0071FC 082800010020  btst   #1,32(A0)      rtschk
007202 6704      beq    $7208      RTS/CTS mode ?
007204 61000862      bsr    $7A68      No
007208 4E75      rts      rtson

*****
00720A 40E7      move.w  SR, -(A7)      rs232put, RS232 output
00720C 007C0700      or.w   #$700,SR      Save status
007210 61C8      bsr    $71DA      IPL 7, disable interrupts
007212 082800000020  btst   #0,32(A0)      rs232ptr, get buffer pointer
007218 6706      beq    $7220      XON/XOFF mode ?
                        No

```



```

00721A 4A28001F      tst.b      31(A0)      XON active ?
00721E 6618          bne      $7238      Yes
007220 08290007002C  btst      #7,44(A1)      Is MFP still sending ?
007226 6710          beq      $7238      Yes
007228 34280014      move.w     20(A0),D2      Head index
00722C B4680016      cmp.w      22(A0),D2      Tail index
007230 6606          bne      $7238      Still characters in buffer?
007232 1341002E      move.b     D1,46(A1)      Pass byte to MFP
007236 601A          bra      $7252

007238 34280016      move.w     22(A0),D2      Tail index
00723C 610002E0      bsr      $751E      Test for wrap around
007240 B4680014      cmp.w      20(A0),D2      Compare with head index
007244 6716          beq      $725C      Same, buffer full
007246 2268000E      move.l     14(A0),A1      Get current buffer address
00724A 13812000      move.b     D1,0(A1,D2.w)  Write character in buffer
00724E 31420016      move.w     D2,22(A0)      Save new tail index
007252 61A8          bsr      $71FC      rtschk, set RTS ?
007254 46DF          move.w     (A7)+,SR      Restore status flag
007256 023C00FE      and.b      #254,SR      Clear carry flag
00725A 4E75          rts

00725C 619E          bsr      $71FC      rtschk, set RTS ?
00725E 46DF          move.w     (A7)+,SR      Restore status
007260 003C0001      or.b       #1,SR      Set carry flag, don't output char
007264 4E75          rts

*****
007266 40E7          move.w     SR, -(A7)      rs232get, RS232 input
007268 007C0700      or.w       #$700,SR      Save status
00726C 6100FF6C      bsr      $71DA      IPL 7, disable interrupts
007270 32280006      move.w     6(A0),D1      rs232ptr, get RS232 pointer
                                Head index

```

007274 B2680008	cmp.w 8(A0),D1	Tail index
007278 671A	beq \$7294	No character in buffer ?
00727A 61000296	bsr \$7512	Test for wrap around
00727E 22680000	move.l 0(A0),A1	Get buffer address
007282 7000	moveq.l #0,D0	
007284 10311000	move.b 0(A1,D1.w),D0	Get character from buffer
007288 31410006	move.w D1,6(A0)	Set new head index
00728C 46DF	move.w (A7)+,SR	Restore status
00728E 023C00FE	and.b #254,SR	Clear carry flag, OK
007292 6006	bra \$729A	
007294 46DF	move.w (A7)+,SR	Restore status
007296 003C0001	or.b #1,SR	Set carry flag, no character there
00729A 082800000020	btst #0,32(A0)	XON/XOFF mode ?
0072A0 671C	beq \$72BE	No
0072A2 4A28001E	tst.b 30(A0)	XON active?
0072A6 6716	beq \$72BE	No
0072A8 6100FF3E	bsr \$71E8	Get input buffer length used
0072AC B468000A	cmp.w 10(A0),D2	Equal low water mark ?
0072B0 660C	bne \$72BE	No
0072B2 123C0011	move.b #11,D1	XON
0072B6 6100FF52	bsr \$720A	Send
0072BA 4228001E	clr.b 30(A0)	Clear XON flag
0072BE 4E75	rts	
*****	*****	rcvint, RS232 receiver interrupt routine
0072C0 48E7F0E0	movem.l D0-D3/A0-A2,-(A7)	Save registers
0072C4 6100FF14	bsr \$71DA	rs232ptr, get RS232 pointer
0072C8 1169002A001C	move.b 42(A1),28(A0)	Read receiver status register
0072CE 08280007001C	btst #7,28(A0)	Interrupt through receiver buffer full ?
0072D4 67000092	beq \$7368	No
0072D8 082800010020	btst #1,32(A0)	RTS/CTS mode ?

0072DE 6704	beq \$72E4	No
0072E0 61000782	bsr \$7A64	rtsoff
0072E4 1029002E	move.b 46(A1),D0	Read data from receiver register
0072E8 082800010020	btst #1,32(A0)	RTS/CTS mode ?
0072EE 6624	bne \$7314	Yes
0072F0 082800000020	btst #0,32(A0)	XON/XOFF mode ?
0072F6 671C	beq \$7314	No
0072F8 0C000011	cmp.b #17,D0	XON received ?
0072FC 6608	bne \$7306	No
0072FE 117C0000001F	move.b #0,31(A0)	Clear XOFF flag
007304 6062	bra \$7368	Character not in buffer
007306 0C000013	cmp.b #19,D0	Receive XOFF ?
00730A 6608	bne \$7314	No
00730C 117C00FF001F	move.b #FF,31(A0)	Set XOFF flag
007312 6054	bra \$7368	Character not in buffer
007314 32280008	move.w 8(A0),D1	Tail index
007318 610001F8	bsr \$7512	Test for wrap around
00731C B2680006	cmp.w 6(A0),D1	Head equal tail ?
007320 6746	beq \$7368	Yes
007322 24680000	move.l 0(A0),A2	Get buffer address
007326 15801000	move.b D0,0(A2,D1.w)	Received byte in buffer
00732A 31410008	move.w D1,8(A0)	Save new tail index
00732E 6100FEB8	bsr \$71E8	Get input buffer length used
007332 B468000C	cmp.w 12(A0),D2	Equal high water mark ?
007336 6624	bne \$735C	No
007338 082800010020	btst #1,32(A0)	RTS/CTS mode ?
00733E 6628	bne \$7368	Yes
007340 082800000020	btst #0,32(A0)	XON/XOFF mode ?
007346 6714	beq \$735C	No
007348 4A28001E	tst.b 30(A0)	XOFF already sent ?

```

00734C 660E      bne      $735C
00734E 117C00FF001E  move.b  #$FF,30(A0)
007354 123C0013    move.b  #$13,D1
007358 6100FE00    bsr      $720A
00735C 082800010020 btst     #1,32(A0)
007362 6704      beq      $7368
007364 61000702    bsr      $7A68
007368 08A90004000E bclr     #4,14(A1)
00736E 4CDF070F    movem.l  (A7)+,D0-D3/A0-A2
007372 4E73      rte

*****
007374 48E720E0    movem.l  D2/A0-A2,-(A7)
007378 6100FE60    bsr      $71DA
00737C 082800010020 btst     #1,32(A0)
007382 6630      bne      $73B4
007384 082800000020 btst     #0,32(A0)
00738A 6706      beq      $7392
00738C 4A28001F    tst.b    31(A0)
007390 6622      bne      $73B4
007392 1169002C001D  move.b  44(A1),29(A0)
007398 34280014    move.w  20(A0),D2
00739C B4680016    cmp.w   22(A0),D2
0073A0 6712      beq      $73B4
0073A2 6100017A    bsr      $751E
0073A6 2468000E    move.l  14(A0),A2
0073AA 13722000002E  move.b  0(A2,D2.w),46(A1)
0073B0 31420014    move.w  D2,20(A0)
0073B4 08A90002000E bclr     #2,14(A1)
0073BA 4CDF0704    movem.l  (A7)+,D2/A0-A2
0073BE 4E73      rte

txrint, transmitter buffer empty
Save registers ?
rs232ptr, get RS232 pointer
RTS/CTS mode ?
Yes, then use this interrupt
XON/XOFF mode ?
No
XOFF active ?
Yes
Save transmitter status register
Head index
Compare with tail index
Send buffer empty
Test for wrap around
Pointer to send buffer
Pass byte to MPP for sending
Save new head index
Clear interrupt service bit
Restore registers

```

```

*****
0073C0 48E720E0      movem.l D2/A0-A2,-(A7)
0073C4 6100FE14      bsr $71DA
0073C8 082800010020  btst #1,32(A0)
0073CE 672A          beq $73FA
0073D0 1169002C001D  move.b 44(A1),29(A0)
0073D6 08280007001D  btst #7,29(A0)
0073DC 67F8          beq $73D6
0073DE 34280014      move.w 20(A0),D2
0073E2 B4680016      cmp.w 22(A0),D2
0073E6 671E          beq $7406
0073E8 61000134      bsr $751E
0073EC 2468000E      move.l 14(A0),A2
0073F0 13722000002E  move.b 0(A2,D2.w),46(A1)
0073F6 31420014      move.w D2,20(A0)
0073FA 08A900020010  bclr #2,16(A1)
007400 4CDF0704      movem.l (A7)+,D2/A0-A2
007404 4E73          rte

007406 60F2      bra $73FA

*****
007408 48E780C0      movem.l D0/A0-A1,-(A7)
00740C 6100FDCC      bsr $71DA
007410 1169002A001C  move.b 42(A1),28(A0)
007416 1029002E      move.b 46(A1),D0
00741A 08A90003000E  bclr #3,14(A1)
007420 4CDF0301      movem.l (A7)+,D0/A0-A1
007424 4E73          rte

*****
007426 48E700C0      movem.l A0-A1,-(A7)

```

ctsint, CTS interrupt routine  
Save registers  
rs232ptr, get RS232 pointer  
RTS/CTS mode ?  
No, then ignore interrupt  
Save transmitter status  
Transmitter buffer empty ?  
No, wait (must jump to \$73D0!)  
Head index  
Compare with tail index  
Transmit buffer empty  
Test for wrap around  
Pointer to transmit buffer  
Pass byte to MFP for sending  
Save new head index  
Clear interrupt service bit  
Restore registers  
Transmit buffer empty  
rxerror, receive error  
Save registers  
rs232ptr, get RS232 pointer  
Save receiver status  
Read data register (clears status)  
Clear interrupt service bit  
Restore registers  
txerror, transmit error  
Save registers

```

00742A 6100FDAE      bsr      $71DA
00742E 1169002C001D  move.b 44(A1),29(A0)
007434 08A90001000E  bclr   #1,14(A1)
00743A 4CDF0300      movem.l (A7)+/A0-A1
00743E 4E73         rte

*****
007440 7200      moveq.l #0,D1
007442 322F0004  move.w 4(A7),D1
007446 40E7      move.w SR, -(A7)
007448 007C0700  or.w   #$700,SR
00744C 45F90000745C  lea    $745C,A2
007452 E581      asl.l  #2,D1
007454 20321800  move.l 0(A2,D1.1),D0
007458 46DF      move.w (A7)+,SR
00745A 4E75      rts

*****
00745C 000009D0      dc.l  $9D0
007460 000009F2      dc.l  $9F2
007464 00000A00      dc.l  $A00

*****
007468 007C0700      or.w   #$700,SR
00746C 6100FD6C      bsr    $71DA
007470 0F490028      movep.l $28(A1),D7
007474 7000      moveq.l #0,D0
007476 13400002A      move.b D0,42(A1)
00747A 13400002C      move.b D0,44(A1)
00747E 4A6F0006      tst.w  6(A7)
007482 6B0A      bmi    $748E
007484 116F00070020  move.b 7(A7),32(A0)

rs232ptr, get RS232 pointer
Save transmitter status
Clear interrupt service bit
Restore registers

iorec, get pointer to table

Get device number
Save status
IPL 7, disable interrupts
Base address of the table
Times 4 for long word
Get pointer to iorec
Restore status

iorec table
RS232
IKBD
MIDI

rsconf, configure RS232
IPL 7, disable interrupts
rs232ptr, get RS232 pointer
Save ucr, isr, tsr, scr

Disable receiver
Disable transmitter
Mode
Negative, then don't set
Reset rsmode

```

00748A 7000	moveq.l #0,D0	Baud rate
00748C 7400	moveq.l #0,D2	Negative, then don't change
00748E 4A6F0004	tst.w 4(A7)	Get baud rate number
007492 6B20	bmi \$74B4	Table for baud rate control
007494 322F0004	move.w 4(A7),D1	Get value from table
007498 45F9000074F2	lea \$74F2,A2	Table for baud rate data
00749E 10321000	move.b 0(A2,D1.w),D0	Get value from table
0074A2 45F900007502	lea \$7502,A2	
0074A8 14321000	move.b 0(A2,D1.w),D2	
0074AC 2200	move.l D0,D1	
0074AE 7003	moveq.l #3,D0	Pointer to timer D
0074B0 6100FBDE	bsr \$7090	Set timer D for new baud rate
0074B4 4A6F0008	tst.w 8(A7)	Set ucr ?
0074B8 6B06	bmi \$74C0	No
0074BA 136F00090028	move.b 9(A7),40(A1)	Set ucr
0074C0 4A6F000A	tst.w 10(A7)	Set rsr ?
0074C4 6B06	bmi \$74CC	No
0074C6 136F000B002A	move.b 11(A7),42(A1)	Set rsr
0074CC 4A6F000C	tst.w 12(A7)	Set tsr ?
0074D0 6B06	bmi \$74D8	No
0074D2 136F000D002C	move.b 13(A7),44(A1)	Set tsr
0074D8 4A6F000E	tst.w 14(A7)	Set scr ?
0074DC 6B06	bmi \$74E4	No
0074DE 136F000F0026	move.b 15(A7),38(A1)	Set scr
0074E4 7001	moveq.l #1,D0	Enable receiver
0074E6 1340002A	move.b D0,42(A1)	Enable transmitter
0074EA 1340002C	move.b D0,44(A1)	Old value of the control register
0074EE 2007	move.l D7,D0	
0074F0 4E75	rts	
*****		
0074F2 01010101010101	dc.b 1,1,1,1,1,1,1	Timer values for RS-232 baud rates
		High byte

```

0074FA 0101010101010202      dc.b      1,1,1,1,1,2,2
007502 01020405080A0E10      dc.b      1,2,4,5,8,10,11,16      Low byte
00750B 204060808FAF4060      dc.b      32,64,96,128,143,175,64,96

*****
007512 5241                  addq.w      #1,D1
007514 B2680004              cmp.w      4(A0),D1
007518 6502                  bcs       $751C
00751A 7200                  moveq.l   #0,D1
00751C 4E75                  rts

*****
00751E 5242                  addq.w      #1,D2
007520 B4680012              cmp.w      18(A0),D2
007524 6502                  bcs       $7528
007526 7400                  moveq.l   #0,D2
007528 4E75                  rts

*****
00752A 48E7FFFE              movem.l   D0-D7/A0-A6,-(A7)
00752E 4BF900000000          lea       $0,A5
007534 246D0A2A              move.l   $A2(A5),A2
007538 4E92                  jsr       (A2)
00753A 246D0A2E              move.l   $A2E(A5),A2
00753E 4E92                  jsr       (A2)
007540 08390004FFFFFA01      btst      #4,$FFFFFA01
007548 67EA                  beq       $7534
00754A 08B90006FFFFFA11      bclr     #6,$FFFFFA11
007552 4CDF7FFF              movem.l   (A7)+,D0-D7/A0-A6
007556 4E73                  rts

*****
wrapin, test for wrap around
Head index + 1
Equal to buffer size ?
No
Else start with zero again

wrapout, test for wrap around
Tail index + 1
Equals buffer size ?
No
Else start with zero again

midkey, keyboard + MIDI interrupt
Save registers
Clear A5
mbufrec, $7558
Interrupt from MIDI-ACIA ?
kbufrec, $7568
Interrupt vfromon Keyboard-ACIA ?
Still interrupt ?
Yes
Clear interrupt service bit
Restore registers

```



```

*****
007558 41ED0A00      lea    $A00(A5),A0
00755C 43F9FFFC04      lea    $FFFFC04,A1
007562 246D0A16      move.l $A16(A5),A2
007566 600E          bra     $7576

*****
007568 41ED09F2      lea    $9F2(A5),A0
00756C 43F9FFFC00      lea    $FFFFC00,A1
007572 246D0A12      move.l $A12(A5),A2
007576 14290000      move.b 0(A1),D2
00757A 08020007      btst   #7,D2
00757E 671C          beq     $759C
007580 08020000      btst   #0,D2
007584 670A          beq     $7590
007586 48E720E0      movem.l D2/A0-A2,-(A7)
00758A 6112          bsr     $759E
00758C 4CDF0704      movem.l (A7)+,D2/A0-A2
007590 02020020      and.b  $20,D2
007594 6706          beq     $759C
007596 10290002      move.b 2(A1),D0
00759A 4ED2          jmp     (A2)
00759C 4E75          rts

*****
00759E 10290002      move.b 2(A1),D0
0075A2 B1FC00009F2      cmp.l  #$9F2,A0
0075A8 66000416      bne     $79C0
0075AC 4A2D0A32      tst.b  $A32(A5)
0075B0 6654          bne     $7606
0075B2 0C0000F6      cmp.b  #$F6,D0
0075B6 650000F4      bcs     $76AC

*****
MIDI interrupt
iorec for MIDI
Address of the MIDI ACIA
$759C, MIDI error routine

*****
Keyboard interrupt
iorec for keyboard
Address of the keyboard-ACIA
$759C, Keyboard error routine
Get ACIA status
Interrupt request ?
No
Receiver buffer full ?
No
Save registers
arcvint, get byte
Restore registers
Mark bit tested
No error
Read data clear status
Execute error routine

*****
arcvint, get byte from ACIA
Get data from ACIA
Keyboard ACIA ?
No, MIDI
Keyboard state
KeyPress ?
Yes

```

0075BA 040000F6	sub.b	#\$F6,D0	Subtract offset
0075BE 02800000FF	and.l	#\$FF,D0	
0075C4 47F9000075F2	lea	\$75F2,A3	Pointer to IKBD code table
0075CA 1B7300000A32	move.b	0(A3,D0.w), \$A32(A5)	Save IKBD merken
0075D0 47F9000075FC	lea	\$75FC,A3	Pointer to IKBD lengths table
0075D6 1B7300000A33	move.b	0(A3,D0.w), \$A33(A5)	IKBD index
0075DC 064000F6	add.w	#\$F6,D0	Add offset again
0075E0 0C0000F8	cmp.b	#\$F8,D0	Mouse position record ?
0075E4 6D0A	b.lt	\$75F0	No
0075E6 0C0000FB	cmp.b	#\$FB,D0	Mouse position record ?
0075EA 6E04	b.gt	\$75F0	No
0075EC 1B400A40	move.b	D0, \$A40(A5)	Save mouse position
0075F0 4E75	rts		

*****	IKBD parameter
0075F2 01020303030304050607	Status codes
0075FC 070502020206020101	Lengths
*****	

*****	Joystick record ?	
007606 0C2D000060A32	cmp.b	#6, \$A32(A5)
00760C 64000084	bcc	\$7692
007610 45F900007656	lea	\$7656,A2
007616 7400	moveq.l	#0,D2
007618 142D0A32	move.b	\$A32(A5), D2
00761C 5302	subq.b	#1,D2
00761E E342	asl.w	#1,D2
007620 D42D0A32	add.b	\$A32(A5), D2
007624 5302	subq.b	#1,D2
007626 E542	asl.w	#2,D2
007628 20722000	move.l	0(A2,D2.w), A0
00762C 22722004	move.l	4(A2,D2.w), A1
007630 24722008	move.l	8(A2,D2.w), A2

007634 2452	move.l (A2),A2	Get interrupt vector
007636 7400	moveq.l #0,D2	
007638 142D0A33	move.b \$A33(A5),D2	Get IKBD index
00763C 93C2	sub.l D2,A1	Minus base
00763E 1280	move.b D0,(A1)	
007640 532D0A33	subq.b #1,\$A33(A5)	IKBD index minus 1
007644 4A2D0A33	tst.b \$A33(A5)	Test index
007648 660A	bne \$7654	
00764A 2F08	move.l A0,-(A7)	Pass record pointer
00764C 4E92	jsr (A2)	Execute interrupt routine
00764E 584F	addq.w #4,A7	Correct stack pointer
007650 422D0A32	clr.b \$A32(A5)	Clear IKBD state
007654 4E75	rts	

*****	Parameter table for IKBD
007656 0000A34	dc.l \$A34
00765A 0000A3B	dc.l \$A3B
00765E 0000A1A	dc.l \$A1A
007662 0000A3B	dc.l \$A3B
007666 0000A40	dc.l \$A40
00766A 0000A1E	dc.l \$A1E
00766E 0000A40	dc.l \$A40
007672 0000A43	dc.l \$A43
007676 0000A1E	dc.l \$A1E
00767A 0000A43	dc.l \$A43
00767E 0000A49	dc.l \$A49
007682 0000A22	dc.l \$A22
007686 0000A49	dc.l \$A49
00768A 0000A4B	dc.l \$A4B
00768E 0000A26	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
0076A6 41ED0A49       lea     $A49(A5),A0
0076AA 609E           bra     $764A

Joystick interrupt routine
Address of joystick data

*****
0076AC 122D0A5D       move.b  $A5D(A5),D1
0076B0 0C00002A       cmp.b  #$2A,D0
0076B4 6606           bne     $76BC
0076B6 08C10001       bset   #1,D1
0076BA 6074           bra     $7730
0076BC 0C0000AA       cmp.b  #$AA,D0
0076C0 6606           bne     $76C8
0076C2 08810001       bclr   #1,D1
0076C6 6068           bra     $7730
0076C8 0C000036       cmp.b  #$36,D0
0076CC 6606           bne     $76D4
0076CE 08C10000       bset   #0,D1
0076D2 605C           bra     $7730
0076D4 0C0000B6       cmp.b  #$B6,D0
0076D8 6606           bne     $76E0
0076DA 08810000       bclr   #0,D1
0076DE 6050           bra     $7730
0076E0 0C00001D       cmp.b  #$1D,D0
0076E4 6606           bne     $76EC
0076E6 08C10002       bset   #2,D1
0076EA 6044           bra     $7730
0076EC 0C00009D       cmp.b  #$9D,D0

Process received keyboard codes
Load shift status
Left shift key pressed ?
No
Set bit for left shift key
Left shift key released ?
No
Clear bit for left shift key
Right shift key pressed ?
No
Set bit for right shift key
Right shift key released ?
No
Clear bit for right shift key
CTRL key pressed ?
No
Set bit for CTRL key
CTRL key released ?

```

0076F0 6606	bne \$76F8	No
0076F2 08810002	bclr #2,D1	Clear bit for CTRL key
0076F6 6038	bra \$7730	
0076F8 0C000038	cmp.b #538,D0	ALT key pressed ?
0076FC 6606	bne \$7704	No
0076FE 08C10003	bset #3,D1	Set bit for ALT key
007702 602C	bra \$7730	
007704 0C0000B8	cmp.b #5B8,D0	ALT key released ?
007708 6606	bne \$7710	No
00770A 08810003	bclr #3,D1	Clear bit for ALT key
00770E 6020	bra \$7730	
007710 0C00003A	cmp.b #53A,D0	CAPS LOCK key pressed ?
007714 6620	bne \$7736	No
007716 082D00000484	btst #0,\$A84(A5)	Get console configuration
00771C 670E	beq \$772C	No key click
00771E 2B7C00007D780A86	move.l #5D78,\$A86(A5)	Addrss of the key click sound table
007726 1B7C00000A8A	move.b #50,\$A8A(A5)	Start sound
00772C 08410004	bchg #4,D1	Invert CAPS LOCK status
007730 1B410A5D	move.b D1,\$A5D(A5)	Save new shift status
007734 4E75	rts	
007736 08000007	btst #7,D0	Was key released ?
00773A 662A	bne \$7766	Yes
00773C 4A2D0A7B	tst.b \$A7B(A5)	Repeat ?
007740 6616	bne \$7758	Yes
007742 1B400A7B	move.b D0,\$A7B(A5)	Save key code for repeat
007746 1B7900000A7E0A7C	move.b \$A7E,\$A7C(A5)	Delay 1
00774E 1B7900000A7F0A7D	move.b \$A7F,\$A7D(A5)	Delay 2
007756 603A	bra \$7792	
007758 1B7C00000A7C	move.b #50,\$A7C(A5)	Clear delay counter
00775E 1B7C00000A7D	move.b #50,\$A7D(A5)	Clear delay counter

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 key released ?
00777E 6708	beq	\$7788	Yes
007780 0C0000D2	cmp.b	#\$D2,D0	INSERT key released ?
007784 66000238	bne	\$79BE	No
007788 082D00030A5D	btst	#3,\$A5D(A5)	ALTkey still pressed ?
00778E 6700022E	beq	\$79BE	No
007792 082D00000484	btst	#0,\$484(A5)	Console status
007798 670E	beq	\$77A8	No key click
00779A 2B7C00007D780A86	move.l	#\$7D78,\$A86(A5)	Address of the sound table for key click
0077A2 1B7C00000A8A	move.b	#\$0,\$A8A(A5)	Start sound
0077A8 2F08	move.l	A0,-(A7)	Save iorec for keyboard
0077AA 7200	moveq.l	#0,D1	
0077AC 1200	move.b	D0,D1	Scan code to D1
0077AE 206D0A5E	move.l	\$A5E(A5),A0	Address of the standard keyboard table
0077B2 0240007F	and.w	#\$7F,D0	
0077B6 082D00040A5D	btst	#4,\$A5D(A5)	CAPS LOCK active ?
0077BC 6704	beq	\$77C2	No
0077BE 206D0A66	move.l	\$A66(A5),A0	Address of the CAPS LOCK keyboard table
0077C2 082D00000A5D	btst	#0,\$A5D(A5)	Right shift key pressed ?
0077C8 6608	bne	\$77D2	Yes
0077CA 082D00010A5D	btst	#1,\$A5D(A5)	Left shift key pressed ?
0077D0 671A	beq	\$77EC	No

0077D2 0C00003B	cmp.b #59,D0	Function key ?
0077D6 6510	bcs \$77E8	No
0077D8 0C000044	cmp.b #68,D0	Function key ?
0077DC 620A	bhi \$77E8	No
0077DE 06410019	add.w #25,D1	Add offset to GSX standard
0077E2 7000	moveq.l #0,D0	
0077E4 600001B2	bra \$7998	
0077E8 206D0A62	move.l \$A62(A5),A0	Address of the shift keyboard table
0077EC 10300000	move.b 0(A0,D0.w),D0	Get ASCII code from the table
0077F0 082D00020A5D	btst #2,\$A5D(A5)	CTRL key pressed ?
0077F6 6760	beq \$7858	No
0077F8 0C00000D	cmp.b #13,D0	Carriage return
0077FC 6604	bne \$7802	No
0077FE 700A	moveq.l #10,D0	Convert to linefeed
007800 672A	beq \$782C	
007802 0C010047	cmp.b #547,D1	CTRL HOME ?
007806 6608	bne \$7810	No
007808 06410030	add.w #530,D1	Add \$30 to GSX standard
00780C 6000018A	bra \$7998	
007810 0C01004B	cmp.b #54B,D1	CTRL cursor left ?
007814 6608	bne \$781E	
007816 7273	moveq.l #573,D1	Convert to GSX standard
007818 7000	moveq.l #0,D0	
00781A 6000017C	bra \$7998	
00781E 0C01004D	cmp.b #54D,D1	CTRL cursor right ?
007822 6608	bne \$782C	
007824 7274	moveq.l #574,D1	Convert to GSX standard
007826 7000	moveq.l #0,D0	
007828 6000016E	bra \$7998	
00782C 0C000032	cmp.b #532,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 key pressed ?
00785E 67000138	beq \$7998	No	
007862 0C01001A	cmp.b #\$1A,D1	'0' ?	
007866 6618	bne \$7880	No	
007868 103C0040	move.b #\$40,D0	'@'	
00786C 142D0A5D	move.b \$A5D(A5),D2	Shift status	
007870 02020003	and.b #3,D2	Left and/or right shift key pressed?	
007874 67000122	beq \$7998	No	
007878 103C005C	move.b #\$5C,D0	'\'	
00787C 6000011A	bra \$7998		
007880 0C010027	cmp.b #\$27,D1	'0' ?	
007884 6618	bne \$789E	No	
007886 103C005B	move.b #\$5B,D0	'['	
00788A 142D0A5D	move.b \$A5D(A5),D2	Shift status	
00788E 02020003	and.b #3,D2	Left and/or right shift key pressed?	
007892 67000104	beq \$7998	No	
007896 103C007B	move.b #\$7B,D0	'{'	
00789A 600000FC	bra \$7998		
00789E 0C010028	cmp.b #\$28,D1	'A' ?	
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	Left and/or right shift key pressed?
0078E0 670000E6	beq \$7998	No
0078B4 103C007D	move.b #\$7D,D0	','
0078B8 600000DE	bra \$7998	
0078EC 0C010062	cmp.b #\$62,D1	ATL HELP ?
0078C0 660A	bne \$78CC	No
0078C2 526D04EE	addq.w #1,\$4EE(A5)	Set dumpflg for hardcopy
0078C6 205F	move.l (A7)+,A0	Restore keyboard iorec
0078C8 600000F4	bra \$79BE	Done
0078CC 45F900007A2C	lea \$7A2C,A2	Pointer to mouse scancode table
0078D2 7403	moveq.l #3,D2	Test 4 values
0078D4 B2322000	cmp.b 0(A2,D2.w),D1	Value found ?
0078D8 6700010E	beq \$79E8	Yes
0078DC 51CAFFF6	dbra D2,\$78D4	Next value
0078E0 0C010048	cmp.b #\$48,D1	Cursor up ?
0078E4 661C	bne \$7902	No
0078E6 123C0000	move.b #0,D1	X offset for cursor up
0078EA 143CFFF8	move.b #-8,D2	Y offset for cursor up
0078EE 102D0A5D	move.b \$A5D(A5),D0	Get shift status
0078F2 02000003	and.b #3,D0	Left and/or right shift key pressed?
0078F6 6700010E	beq \$7A06	No
0078FA 143CFFF8	move.b #-1,D2	Only one pixel up with shift
0078FE 60000106	bra \$7A06	
007902 0C01004B	cmp.b #\$4B,D1	Cursor left ?
007906 661C	bne \$7924	No
007908 143C0000	move.b #0,D2	Y offset for cursor left
00790C 123CFFF8	move.b #-8,D1	X offset for cursor left
007910 102D0A5D	move.b \$A5D(A5),D0	Get shift status
007914 02000003	and.b #3,D0	Left and/or right shift key pressed?
007918 670000EC	beq \$7A06	No

00791C 123CFFF	move.b #0,D1	Only one pixel left with shift
007920 60000E4	bra \$7A06	
007924 0C01004D	cmp.b #0,D1	Cursor right ?
007928 661C	bne \$7946	No
00792A 123C0008	move.b #0,D1	X offset for cursor right
00792E 143C0000	move.b #0,D2	Y offset for cursor right
007932 102D0A5D	move.b \$A5D(A5),D0	Shift status
007936 02000003	and.b #0,D0	Left and/or right shift key pressed?
00793A 670000CA	beq \$7A06	No
00793E 123C0001	move.b #0,D1	Only one pixel right with shift
007942 600000C2	bra \$7A06	
007946 0C010050	cmp.b #0,D1	Cursor down ?
00794A 661C	bne \$7968	No
00794C 123C0000	move.b #0,D1	X offset for cursor down
007950 143C0008	move.b #0,D2	Y offset for cursor down
007954 102D0A5D	move.b \$A5D(A5),D0	Shift status
007958 02000003	and.b #0,D0	Left and/or right shift key pressed?
00795C 670000A8	beq \$7A06	No
007960 143C0001	move.b #0,D2	Only one pixel down with shift
007964 600000A0	bra \$7A06	
007968 0C010002	cmp.b #0,D1	'1'
00796C 650C	bcs \$797A	Not greater or equal
00796E 0C01000D	cmp.b #0,D1	'='
007972 6206	bhi \$797A	Not less or equal
007974 06010076	add.b #0,D1	
007978 600C	bra \$7986	
00797A 0C000041	cmp.b #0,D0	'A'
00797E 650A	bcs \$798A	
007980 0C00005A	cmp.b #0,D0	'Z'
007984 6204	bhi \$798A	
007986 7000	moveq.l #0,D0	
007988 600E	bra \$7998	

```

00798A 0C000061      cmp.b  #561,D0
00798E 6508          bcs  $7998
007990 0C00007A      cmp.b  #57A,D0
007994 6202          bhi  $7998
007996 60EE          bra  $7986
007998 E141          asl.w  #8,D1
00799A D041          add.w  D1,D0
00799C 205F          move.l (A7)+,A0
00799E 32280008      move.w 8(A0),D1
0079A2 5441          addq.w #2,D1
0079A4 B2680004      cmp.w  4(A0),D1
0079A8 6502          bcs  $79AC
0079AA 7200          moveq.l #0,D1
0079AC B2680006      cmp.w  6(A0),D1
0079B0 670C          beq  $79BE
0079B2 24680000      move.l 0(A0),A2
0079B6 35801000      move.w D0,0(A2,D1.w)
0079BA 31410008      move.w D1,8(A0)
0079BE 4E75          rts

*****
0079C0 246D0A0E      move.l $A0E(A5),A2
0079C4 4ED2          jmp  (A2)

*****
0079C6 32280008      move.w 8(A0),D1
0079CA 5241          addq.w #1,D1
0079CC B2680004      cmp.w  4(A0),D1
0079D0 6502          bcs  $79D4
0079D2 7200          moveq.l #0,D1
0079D4 B2680006      cmp.w  6(A0),D1
0079D8 670C          beq  $79E6

*****
midibyte
Pointer to MIDI interrupt handler
Execute routine

sysmidi
Tail index
Increment
End of buffer reached ?
No
Buffer pointer back to start
Head equal tail ?
Yes, buffer full

*****

```

```

0079DA 24680000      move.l 0(A0),A2      Buffer address
0079DE 15801000      move.b D0,0(A2,D1.w)      Write received byte in buffer
0079E2 31410008      move.w D1,8(A0)      New tail index
0079E6 4E75          rts

*****
0079E8 7605          moveq.l #5,D3
0079EA 08010004      btst     #4,D1
0079EE 6702          beq      $79F2
0079F0 7606          moveq.l #6,D3
0079F2 08010007      btst     #7,D1
0079F6 6706          beq      $79FE
0079F8 07AD0A5D      bclr    D3,$A5D(A5)
0079FC 6004          bra     $7A02
0079FE 07ED0A5D      bset    D3,$A5D(A5)
007A02 7200          moveq.l #0,D1
007A04 7400          moveq.l #0,D2

*****
007A06 41ED0A5A      lea     $A5A(A5),A0      Pointer to mouse-emulator buffer
007A0A 246D0A1E      move.l $A1E(A5),A2      Mouse interrupt vector
007A0E 4280          clr.l  D0
007A10 102D0A5D      move.b $A5D(A5),D0      Get status of the "mouse" buttons
007A14 EA08          lsr.b  #5,D0            Bit for right button to bit 0
007A16 060000F8      add.b  #SF8,D0          plus relative mouse header
007A1A 11400000      move.b D0,0(A0)        Byte in buffer
007A1E 11410001      move.b D1,1(A0)        Store X value
007A22 11420002      move.b D2,2(A0)        Store Y value
007A26 4E92          jsr     (A2)           Call mouse interrupt routine
007A28 205F          move.l (A7)+,A0        Restore iorec for keyboard
007A2A 4E75          rts

```

```

*****
007A2C 47C7      dc.b    $47,$C7,$52,$D2
*****
*****
007A30 302F0004      move.w 4(A7),D0
007A38 40E7      move.w SR, -(A7)
007A3A 007C0700      or.w    $700,SR
007A3E 48E76080      movem.l D1-D2/A0,-(A7)
007A42 41F9FFF800      lea     $FFFF800,A0
007A48 1401      move.b D1,D2
007A4A 0201000F      and.b  #15,D1
007A4E 1081      move.b D1,(A0)
007A50 E302      asl.b  #1,D2
007A52 6404      bcc   $7A58
007A54 11400002      move.b D0,2(A0)
007A58 7000      moveq.l #0,D0
007A5A 1010      move.b (A0),D0
007A5C 4CDF0106      movem.l (A7)+,D1-D2/A0
007A60 46DF      move.w (A7)+,SR
007A62 4E75      rts
*****
007A64 7408      moveq.l #8,D2
007A66 6012      bra   $7A7A
*****
007A68 74F7      moveq.l #$F7,D2
007A6A 6034      bra   $7AA0
*****
007A6C 7410      moveq.l #16,D2
007A6E 600A      bra   $7A7A
*****
muskeyl
Scan code mouse substitute

giaccess, read/write sound chip
Data
Save status
IPL 7, disable interrupts
Save regss of the sound chip
Address of the sound chip
Get register number
Registers 0-15
Select register
Test read/write bit
Read
Write data byte in sound chip register

Read byte from sound chip register
Restore registers
Restore status

rts off, disable RTS

Set bit in port A

rts on, enable RTS

Clear bit in port A

dtr off, disable DTR

Set bit in port A

```

```

*****
007A70 74EF      moveq.l #-17,D2
007A72 602C      bra      $7AA0
*****
dtroff, enable DTR
Clear bit in port A
*****
007A74 7400      moveq.l #0,D2
007A76 342F0004   move.w  4(A7),D2
007A7A 48E7E000   movem.l D0-D2,-(A7)
007A7E 40E7      move.w  SR,-(A7)
007A80 007C0700   or.w    #$700,SR
007A84 720E      moveq.l #14,D1
007A86 2F02      move.l  D2,-(A7)
007A88 61AE      bsr     $7A38
007A8A 241F      move.l  (A7)+,D2
007A8C 8002      or.b    D2,D0
007A8E 728E      moveq.l #$8E,D1
007A90 61A6      bsr     $7A38
007A92 46DF      move.w  (A7)+,SR
007A94 4CDF0007   movem.l (A7)+,D0-D2
007A98 4E75      rts
*****
007A9A 7400      moveq.l #0,D2
007A9C 342F0004   move.w  4(A7),D2
007AA0 48E7E000   movem.l D0-D2,-(A7)
007AA4 40E7      move.w  SR,-(A7)
007AA6 007C0700   or.w    #$700,SR
007AAA 720E      moveq.l #14,D1
007AAC 2F02      move.l  D2,-(A7)
007AAE 6188      bsr     $7A38
007AB0 241F      move.l  (A7)+,D2
*****
offgibit, clear bit in sound chip port A
*****
Get bit pattern
Save registers
Save status
IPL 7, disable interrupts
Port A
Save bit number
Select port A
Bit number back
Set bit(s)
Write to port A
Write new value
Restore status
Restore registers
*****
Get bit pattern
Save registers
Save status
IPL 7, disable interrupts
Port A
Save bit pattern
Select port A
Restore bit pattern

```

```

007AB2 C002          and.b      D2,D0
007AB4 728E          moveq.l  #$8E,D1
007AB6 6180          bsr       $7A38
007AB8 46DF          move.w   (A7)+,SR
007ABA 4CDF0007      movem.l  (A7)+,D0-D2
007ABE 4E75          rts

*****
007AC0 4A6F0004      tst.w    4(A7)
007AC4 6726          beq       $7AEC
007AC6 2B6F000A0A1E move.l   10(A7),$A1E(A5)
007ACC 266F0006      move.l   6(A7),A3
007AD0 0C6F00010004 cmp.w    #1,4(A7)
007AD6 6724          beq       $7AFC
007AD8 0C6F00020004 cmp.w    #2,4(A7)
007ADE 6736          beq       $7B16
007AE0 0C6F00040004 cmp.w    #4,4(A7)
007AE6 6770          beq       $7B58
007AE8 7000          moveq.l  #0,D0
007AEA 4E75          rts

*****
007AEC 7212          moveq.l  #$12,D1
007AEE 6100F1E0      bsr       $6CD0
007AF2 2B7C00007BC0A1E move.l   #$7BC0,$A1E(A5)
007AFA 6070          bra       $7B6C

*****
007AFC 45ED0A6A      lea       $A6A(A5),A2
007B00 14FC0008      move.b   #$8,(A2)+
007B04 14FC000B      move.b   #$B,(A2)+
007B08 6166          bsr       $7B70

Clear bit(s)
Write to port A
Write new value
Restore status
Restore registers

initmouse
Disable mouse ?
Yes disable mouse
Mouse interrupt vector
Address of the parameter block
Relative mouse ?
Yes
Absolute mouse ?
Yes
Keycode mouse ?
Yes
Error, invalid

Disable mouse
Disable mouse command
Send to IKBD
Mouse interrupt vector to RTS

Relative mouse
Transfer buffer pointer
Relative mouse
Relative mouse threshold x, y
Set mouse parameters

```

```

007B0A 7606      moveq.l #6,D3
007B0C 45ED0A6A   lea    $A6(A5),A2
007B10 6100F1DE   bsr    $6CF0
007B14 6056      bra    $7B6C

*****
007B16 45ED0A6A   lea    $A6(A5),A2
007B1A 14FC0009   move.b #9,(A2)+
007B1E 14EB0004   move.b 4(A3),(A2)+
007B22 14EB0005   move.b 5(A3),(A2)+
007B26 14EB0006   move.b 6(A3),(A2)+
007B2A 14EB0007   move.b 7(A3),(A2)+
007B2E 14FC000C   move.b #5C,(A2)+
007B32 613C      bsr    $7B70
007B34 14FC000E   move.b #5E,(A2)+
007B38 14FC0000   move.b #0,(A2)+
007B3C 14EB0008   move.b 8(A3),(A2)+
007B40 14EB0009   move.b 9(A3),(A2)+
007B44 14EB000A   move.b 10(A3),(A2)+
007B48 14EB000B   move.b 11(A3),(A2)+
007B4C 7610      moveq.l #16,D3
007B4E 45ED0A6A   lea    $A6(A5),A2
007B52 6100F19C   bsr    $6CF0
007B56 6014      bra    $7B6C
007B58 45ED0A6A   lea    $A6(A5),A2
007B5C 14FC000A   move.b #5A,(A2)+
007B60 610E      bsr    $7B70
007B62 7605      moveq.l #5,D3
007B64 45ED0A6A   lea    $A6(A5),A2
007B68 6100F186   bsr    $6CF0
007B6C 70FF      moveq.l #-1,D0
007B6E 4E75      rts

Length of the strings - 1
Transfer buffer pointer
Send string to IKBD

Absolute mouse
Transfer buffer pointer
Absolute mouse
xmax msb
xmax lsb
ymax msb
ymax lsb
Absolute mouse scale
Set mouse parameters
Initial absolute mouse position
Fill byte
Start position x msb
Start position x lsb
Start position y msb
Start position y lsb
String length -1
Transfer buffer pointer
Send string to IKBD

Transfer buffer pointer
Mouse keycode mode
Set mouse parameters
Length of the string -1
Transfer buffer pointer
Send string to IKBD
Flag for OK

```



```

*****
007B70 14EB0002      move.b 2(A3), (A2)+
007B74 14EB0003      move.b 3(A3), (A2)+
007B78 7210          moveq.l #16, D1
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

*****
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
007B94 322F0006      move.w 6(A7), D1
007B98 342F0008      move.w 8(A7), D2
007B9C 6100F4F2      bsr $7090
007BA0 4AAF000A      tst.l 10(A7)
007BA4 6E1A          bml $7BC0
007BA6 246F000A      move.l 10(A7), A2
007BAA 7200          moveq.l #0, D1
007BAC 43F900007BC2  lea $7BC2, A1
007BB2 0280000000FF  and.l #$FF, D0
007BB8 10310000      move.b 0(A1, D0.w), D0
007BBC 6100F588      bsr $7146
007BC0 4E75          rts

*****
007BC2 0D080504      dc.b 13, 8, 5, 4
*****

setmouse, set mouse parameters
x threshold, scale, delta
y threshold, scale, delta
Top/bottom ?

xbtimer, initialize timer

Timer number (0-3 => A - D)
Value for control register
Value for data register
Set timer values
Interrupt vector
Not used ?
Interrupt vector

Table for determining interrupt number

Get interrupt number
initint, set MFP interrupt

Interrupt numbers of the MFP timers

```

```

*****
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 2B7C000006D620A5E  move.l  #$D62, $A5E(A5)
007BFA 2B7C000006DE20A62  move.l  #$DE2, $A62(A5)
007C02 2B7C000006EE20A66  move.l  #$EE2, $A66(A5)
007C0A 4E75          rts

*****
007C0C 202D00A86      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)

*****
keytrans, set keyboard table
Change standard table ?
No
Address of the standard table
Change shift table ?
No
Address of the shift table
Change CAPS LOCK table ?
No
Address of the CAPS LOCK table
Pointer to address of the table

*****
bioskeys, set standard keyboard table
Standard table
Shift table
CAPS LOCK table

*****
dosound, start sound
Get sound status
Address of the sound table
Negative, don't set
New sound table
Start sound table

*****
setprt, set/get printer configuration
Get printer configuration
New value
Negative, don't set
Set printer configuration

```

```

007C30 4E75      rts
*****
007C32 302D0A7E      move.w  $A7E(A5),D0
007C36 4A6F0004      tst.w  4(A7)
007C3A 6B16      bmi  $7C52
007C3C 322F0004      move.w  4(A7),D1
007C40 1B410A7E      move.b  D1,$A7E(A5)
007C44 4A6F0006      tst.w  6(A7)
007C48 6B08      bmi  $7C52
007C4A 322F0006      move.w  6(A7),D1
007C4E 1B410A7F      move.b  D1,$A7F(A5)
007C52 4E75      rts
*****
007C54 203C0000A0E      move.l  $A0E,D0
007C5A 4E75      rts
*****
007C5C 52B9000004BA      addq.l  #1,$4BA
007C62 E7F900000A84      rol.w  $A84
007C68 6A4E      bpl  $7CB8
007C6A 48E7FFFE      movem.l D0-D7/A0-A6,-(A7)
007C6E 4BF900000000      lea  $0,A5
007C74 614C      bsr  $7CC2
007C76 082D000104B4      btst  #1,$484(A5)
007C7C 672A      beq  $7CA8
007C7E 4A2D0A7B      tst.b  $A7B(A5)
007C82 6724      beq  $7CA8
007C84 4A2D0A7C      tst.b  $A7C(A5)
007C88 6706      beq  $7C90
007C8A 532D0A7C      subq.b  #1,$A7C(A5)
*****
kbrate, set/get keyboard repeat
Delay before key repeat
Test new value
Negative, don't set

Set new value
Repeat rate
Negative, don't set

Set new value

ikbdrvcs, pointers to IKB + MIDI vectors
Address of the vector table

timerclnt, timer C interrupt
Increment 200 Hz counter
Rotate divisor bit
Not fourth interrupt, then done
Save registers
Clear A5
Process sound
Key repeat enabled ?
No
Key pressed ?
No
Counter for start delay
Not active
Decrement counter

```

```

007C8E 6618      bne      $7CA8
007C90 532D0A7D  subq.b  #1,$A7D(A5)
007C94 6612      bne      $7CA8
007C96 1B6D0A7F0A7D  move.b  $A7F(A5),$A7D(A5)
007C9C 102D0A7B  move.b  $A7B(A5),D0
007CA0 41ED09F2  lea     $9F2(A5),A0
007CA4 6100FAEC  bsr     $7792
007CA8 3F2D0442  move.w  $442(A5),-(A7)
007CAC 206D0400  move.l  $400(A5),A0
007CB0 4E90      jsr     (A0)
007CB2 544F      addq.w  #2,A7
007CB4 4CDF7FFF  movem.l (A7)+,D0-D7/A0-A6
007CB8 08B90005FFFFFALL  bclr    #5,$FFFFFALL
007CC0 4E73      rte

*****
007CC2 48E7C080  movem.l D0-D1/A0,-(A7)
007CC6 202D0A86  move.l  $A86(A5),D0
007CCA 67000088  beq     $7D54
007CCE 2040      move.l  D0,A0
007CD0 102D0A8A  move.b  $A8A(A5),D0
007CD4 6708      beq     $7CDE
007CD6 5300      subq.b  #1,D0
007CD8 1B400A8A  move.b  D0,$A8A(A5)
007CDC 6076      bra     $7D54

007CDE 1018      move.b  (A0)+,D0
007CE0 6B2E      bmi     $7D10
007CE2 13C0FFFFF8900  move.b  D0,$FFFFF8900
007CE8 0C000007  cmp.b   #7,D0
007CEC 661A      bne     $7D08
007CEE 1218      move.b  (A0)+,D1

```

Not run out ?  
 Decrement counter for repeat delay  
 Not run out  
 Reload counter  
 Key to repeat  
 Pointer to iorec keyboard  
 Key code in keyboard buffer  
 20 milliseconds per call  
 Get event timer vector  
 Execute routine  
 Correct stack pointer  
 Restore registers  
 Clear interrupt service bit

sndirq, sound interrupt routine  
 Restore registers  
 Pointer to sound table  
 No sound active ?  
 Pointer to A0  
 Load timer value  
 New sound started ?  
 Else decrement timer  
 And save value  
 Done

Get sound command  
 Bit 7 set, then special command  
 Select register in sound chip  
 Register 7?  
 No  
 Data for register 7

007CF0 0201003F	and.b #3F,D1	Isolate bits 0 ~ 5
007CF4 1039FFFF8800	move.b \$FFFF8800,D0	Read mixer
007CFA 020000C0	and.b #80,D0	Isolate bits 6-7
007CFE 8001	or.b D1,D0	OR data
007D00 13C0FFFF8802	move.b D0,\$FFFF8802	Write byte in sound chip register
007D06 60D6	bra \$7CDE	New sound command
007D08 13D8FFFF8802	move.b (A0)+,\$FFFF8802	Write byte directly in sound chip
007D0E 60CE	bra \$7CDE	Next sound command
007D10 5200	addq.b #1,D0	Was command \$FF ?
007D12 6A32	bpl \$7D46	Yes
007D14 0C000081	cmp.b #81,D0	Was command \$80 ?
007D18 6606	bne \$7D20	No
007D1A 1B580A8B	move.b (A0)+,\$A8B(A5)	Save byte for later
007D1E 60BE	bra \$7CDE	Next sound command
007D20 0C000082	cmp.b #82,D0	Command > \$81 ?
007D24 6620	bne \$7D46	Yes, set timer
007D26 13D8FFFF8800	move.b (A0)+,\$FFFF8800	Select register
007D2C 1018	move.b (A0)+,D0	Add increment-value
007D2E D12D0A8B	add.b D0,\$A8B(A5)	
007D32 1018	move.b (A0)+,D0	End value
007D34 13ED0A8BFFFF8802	move.b \$A8B(A5),\$FFFF8802	Write value in sound chip register
007D3C B02D0A8B	cmp.b \$A8B(A5),D0	End value reached ?
007D40 670E	beq \$7D50	Yes
007D42 5948	subq.w #4,A0	Sound pointer back to same command
007D44 600A	bra \$7D50	
007D46 1B580A8A	move.b (A0)+,\$A8A(A5)	Next value as delay timer
007D4A 6604	bne \$7D50	
007D4C 307C0000	move.w #0,A0	Sound pointer to zero
007D50 2B480A86	move.l A0,\$A86(A5)	And save
007D54 4CDF0103	movem.l (A7)+,D0-D1/A0	Restore registers

[illegible]

007D90 FF00	dc.b	\$FF,0	
*****			
007D92 4E560000	link	A6,#0	Hardcopy
007D96 48E7070C	movem.l	D5-D7/A4-A5,-(A7)	Save registers
007D9A 2A6E0008	move.l	8(A6),A5	Address if the parameter block
007D9E 287C0002600	move.l	#2600,A4	Address of the working memory
007DA4 7E1E	moveq.l	#30,D7	30 bytes
007DA6 6004	bra	\$7DAC	
007DA8 18DD	move.b	(A5)+,(A4)+	Put parameters in working memory
007DAA 5347	subq.w	#1,D7	
007DAC 4A47	tst.w	D7	Next byte
007DAE 6EF8	bgt	\$7DA8	
007DB0 4AB90000261A	tst.l	\$261A	p masks, half-tone mask
007DB6 6708	beq	\$7DC0	Not used ?
007DB8 20390000261A	move.l	\$261A,D0	Load mask
007DBE 6006	bra	\$7DC6	
007DC0 203C00016D4C	move.l	#516D4C,D0	Default mask
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	Set flag
007DD8 60000EC6	bra	\$89A0	Error, stop
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	Printer port
007DF0 4A7900002608	tst.w	\$2608	Height
007DF6 6642	bne	\$7E3A	Not zero ?

007DF8 6028	bra \$7E22	
007DFA 4A7900004EE	tst.w \$4EE	scrdump flag, ALT HELP pressed again?
007E00 6632	bne \$7E34	Yes, stop
007E02 207900002600	move.l \$2600,A0	blkptr
007E08 1010	move.b (A0),D0	Get byte from video RAM
007E0A 4880	ext.w D0	
007E0C 3E80	move.w D0,(A7)	On the stack
007E0E 61000B9A	bsr \$89AA	Output byte
007E12 52B900002600	addq.l #1,\$2600	Increment blkptr
007E18 4A40	tst.w D0	Output OK?
007E1A 6706	beq \$7E22	Yes
007E1C 70FF	moveq.l #-1,D0	Set flag
007E1E 60000B80	bra \$89A0	Error, stop
007E22 4240	clr.w D0	
007E24 303900002606	move.w \$2606,D0	width, screen width
007E2A 537900002606	subq.w #1,\$2606	Decrement width
007E30 4A40	tst.w D0	Already zero?
007E32 66C6	bne \$7DFA	No, output next byte
007E34 4240	clr.w D0	Flag for OK
007E36 60000B68	bra \$89A0	Done
007E3A 0C79000300002616	cmp.w #3,\$2616	p type, Epson B/W matrix
007E42 6306	bls \$7E4A	
007E44 70FF	moveq.l #-1,D0	Set flag
007E46 60000B58	bra \$89A0	Error, stop
007E4A 0C79000100002610	cmp.w #1,\$2610	distres, printer resolution
007E52 6306	bls \$7E5A	
007E54 70FF	moveq.l #-1,D0	Set flag
007E56 60000B48	bra \$89A0	Error, stop



007E5A 0C7900020000260E	cmp.w #2,\$260E	srcres, screen resolution
007E62 6306	bis \$7E6A	
007E64 70FF	moveq.l #-1,D0	Set flag
007E66 60000B38	bra \$89A0	Error, stop
007E6A 0C79000700002604	cmp.w #7,\$2604	Test offset
007E72 6306	bis \$7E7A	
007E74 70FF	moveq.l #-1,D0	Set flag
007E76 60000B28	bra \$89A0	Error, stop
007E7A 4A790000260E	tst.w \$260E	srcres, screen resolution
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 for 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 for medium resolution
007EA4 0C7900020000260E	cmp.w #2,\$260E	srcres, screen resolution
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 for high resolution
007EBA 4A7900002610	tst.w \$2610	dstres, printer resolution
007EC0 6704	beq \$7EC6	Test mode ?
007EC2 4240	clr.w D0	Quality mode
007EC4 6002	bra \$7EC8	

007EC6 7001	moveq.l #1,D0	Printer mode
007EC8 13C000004EE8	move.b D0,\$4EE8	p type, ATARI color matrix
007ECE 0C79000100002616	cmp.w #1,\$2616	
007ED6 6704	beq \$7EDC	
007ED8 4240	clr.w D0	
007EDA 6002	bra \$7EDE	
007EDC 7001	moveq.l #1,D0	ATARI color dot-matrix printer
007EDE 13C0000047CE	move.b D0,\$47CE	p type, ATARI daisy wheel ?
007EE4 0C79000200002616	cmp.w #2,\$2616	Yes
007EEC 6704	beq \$7EF2	
007EEE 4240	clr.w D0	
007EFO 6002	bra \$7EF4	
007EF2 7001	moveq.l #1,D0	Flag for ATARI daisy wheel printer
007EF4 13C000004F84	move.b D0,\$4F84	p type, Epson B/W Matrix ?
007EFA 0C79000300002616	cmp.w #3,\$2616	Yes
007F02 6704	beq \$7F08	
007F04 4240	clr.w D0	
007F06 6002	bra \$7F0A	
007F08 7001	moveq.l #1,D0	Flag for Epson B/W dot matrix
007FOA 13C0000047D0	move.b D0,\$47D0	ATARI daisy wheel printer ?
007F10 4A3900004F84	tst.b \$4F84	No
007F16 6706	beq \$7F1E	
007F18 70FF	moveq.l #-1,D0	Error, stop
007F1A 60000A84	bra \$89A0	
007F1E 4A39000047D0	tst.b \$47D0	Epson B/W dot-matrix printer ?
007F24 670C	beq \$7F32	No
007F26 4A3900004EE8	tst.b \$4EE8	Printer test mode ?
007F2C 6604	bne \$7F32	No
007F2E 7001	moveq.l #1,D0	Printer resolution
007F30 6008	bra \$7F3A	
007F32 103900004EE8	move.b \$4EE8,D0	

007F38 4880	ext.w D0				
007F3A 13C000004EE8	move.b D0,\$4EE8				Printer resolution
007F40 4A3900004F82	tst.b \$4F82				Low resolution ?
007F46 6726	beq \$7F6E				No
007F48 0C79014000002606	cmp.w #320,\$2606				width, 320 points per line
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 points per line
007F6C 6024	bra \$7F92				
007F6E 0C79028000002606	cmp.w #640,\$2606				width, 640 points per line
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 to 640
007F92 4A3900004ED8	tst.b \$4ED8				High resolution ?
007F98 660001E6	bne \$8180				Yes
007F9C 4247	clr.w D7				Clear color counter
007F9E 600001D8	bra \$8178:ln1				
007FA2 207900002612	move.l \$2612,A0				colpal
007FA8 4240	clr.w D0				
007FAA 3010	move.w (A0),D0				Get color
007FAC C07C0777	and.w #\$777,D0				Mask irrelevant bits
007FB0 33C0000047BA	move.w D0,\$47BA				Save color
007FB6 54B900002612	addq.l #2,\$2612				colpal pointer to next color
007FBC 0C790777000047BA	cmp.w #\$777,\$47BA				Color equal white ?
007FC4 67000194	beq \$815A				Yes
007FC8 3039000047BA	move.w \$47BA,D0				Load color

007FCE C07C0007	and.w #7,D0	Isolate blue level
007FD2 33C0000035C2	move.w D0,\$35C2	And save
007FD8 3039000047BA	move.w \$47BA,D0	Load color
007FDE E840	asr.w #4,D0	
007FE0 C07C0007	and.w #7,D0	Isolate green level
007FE4 33C000004EDA	move.w D0,\$4EDA	And save
007FEA 3039000047BA	move.w \$47BA,D0	Load color
007FF0 E040	asr.w #8,D0	
007FF2 C07C0007	and.w #7,D0	Isolate red level
007FF6 33C000004694	move.w D0,\$4694	And save
007FFC 4A39000047CE	tst.b \$47CE	ATARI color dot-matrix printer ?
008002 67000114	beq \$8118	No
008006 3047	move.w D7,A0	
008008 D1C8	add.l A0,A0	
00800A D1FC00004EEC	add.l #\$4EEC,A0	
008010 30B900004694	move.w \$4694,(A0)	
008016 3047	move.w D7,A0	Red level
008018 D1C8	add.l A0,A0	
00801A 227C00004EEC	move.l #\$4EEC,A1	
008020 30309800	move.w 0(A0,A1.1),D0	
008024 B07900004EDA	cmp.w \$4EDA,D0	
00802A 6F08	ble \$8034	Green level
00802C 303900004EDA	move.w \$4EDA,D0	
008032 600E	bra \$8042	Green level
008034 3047	move.w D7,A0	
008036 D1C8	add.l A0,A0	
008038 227C00004EEC	move.l #\$4EEC,A1	
00803E 30309800	move.w 0(A0,A1.1),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,A1.1),D0	
00805C B079000035C2	cmp.w	\$35C2,D0	Blue level
008062 6F08	ble	\$806C	
008064 3039000035C2	move.w	\$35C2,D0	Red 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,A1.1),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	Red 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	
0080AC 6002	bra	\$80A4	
0080A2 7001	moveq.l	#1,D0	
0080A4 33C000004694	move.w	D0,\$4694	Red level
0080AA 303900004EDA	move.w	\$4EDA,D0	Green 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	Green level
0080CE 3039000035C2	move.w \$35C2,D0	Red 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	Blue level
0080F2 303900004694	move.w \$4694,D0	Red level
0080F8 E540	asl.w #2,D0	
0080FA 323900004EDA	move.w \$4EDA,D1	Green level
008100 E341	asl.w #1,D1	
008102 D041	add.w D1,D0	
008104 D079000035C2	add.w \$35C2,D0	Blue 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	Red level

00811E C1FC001E	mul.s.w	\$s1E,D0	Times 30, 30% weighting
008122 323900004EDA	move.w	\$4EDA,D1	Green level
008128 C3FC003B	mul.s.w	\$s3B,D1	Times 59, 59% weighting
00812C D041	add.w	D1,D0	
00812E 3239000035C2	move.w	\$35C2,D1	Blue level
008134 C3FC000B	mul.s.w	\$sB,D1	Times 11, 11% weighting
008138 D041	add.w	D1,D0	
00813A 48C0	ext.l	D0	
00813C 81FC0064	div.s.w	\$s64,D0	
008140 3247	move.w	D7,A1	Divide by 100, scaling
008142 D3C9	add.l	A1,A1	
008144 D3FC00004EEC	add.l	\$s4EEC,A1	
00814A 3280	move.w	D0,(A1)	
00814C 3047	move.w	D7,A0	
00814E D1C8	add.l	A0,A0	
008150 D1FC00004698	add.l	\$s4698,A0	
008156 4250	clr.w	(A0)	
008158 601C	bra	\$8176:ln1	
00815A 3047	move.w	D7,A0	
00815C D1C8	add.l	A0,A0	
00815E D1FC00004698	add.l	\$s4698,A0	
008164 30BC0007	move.w	#7,(A0)	
008168 3047	move.w	D7,A0	
00816A D1C8	add.l	A0,A0	
00816C D1FC00004EEC	add.l	\$s4EEC,A0	
008172 30BC0008	move.w	#8,(A0)	
008176 5247	addq.w	#1,D7	Increment color counter
008178 BE7C0010	cmp.w	#10,D7	16 colors handled already ?
00817C 6D00FE24	blt	\$7FA2	No, next color
008180 4A3900004F82	tst.b	\$4F82	Low resolution ?
008186 6716	beq	\$819E	No
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	No	
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:ln1		
0081BE 33FC000100004768	move.w #1,\$4768		
0081C6 33FC000800004EE2	move.w #8,\$4EE2		
0081CE 33FC000200004F0C	move.w #2,\$4F0C		
0081D6 4A39000047D0	tst.b \$47D0	Epson B/W dot-matrix printer ?	
0081DC 6706	beq \$81E4	No	
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	Height	
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	mul.s.w \$4EE2,D0		



```

008226 33C0000003E80      move.w D0,$E80      blkptr
00822C 2039000002600      move.l $2600,D0      Create even address
008232 C0ECFFFFFFFE      and.l #$FFFFFFFE,D0  And save
008238 23C00000046B8      move.l D0,$46B8      blkptr
00823E 2039000002600      move.l $2600,D0
008244 B0B90000046B8      cmp.l $46B8,D0
00824A 660A                bne $8256
00824C 4240                clr.w D0
00824E 3039000002604      move.w $2604,D0      Offset
008254 600A                bra $8260
008256 4240                clr.w D0
008258 3039000002604      move.w $2604,D0      Offset
00825E 5040                addq.w #8,D0
008260 33C00000047BC      move.w D0,$47BC
008266 4279000012EA      clr.w $12EA
00826C 60000716            bra $8984
008270 4A79000004EE      tst.w $4EE
008276 6600071E            bne $8996
00827A 13FC000100003628  move.b #1,$3628
008282 4240                clr.w D0
008284 3039000002606      move.w $2606,D0      width
00828A C0F9000004768      mulu.w $4768,D0
008290 E848                lsr.w #4,D0
008292 9079000004768      sub.w $4768,D0
008298 E348                lsl.w #1,D0
00829A 4840                swap D0
00829C 4240                clr.w D0
00829E 4840                swap D0
0082A0 D0B90000046B8      add.l $46B8,D0      Pointer to video RAM
0082A6 23C0000004EDC      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	33F9000026060003E2C	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	height
0082F6	303900002608	move.w	\$2608,D0	
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	mults.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	
008372 6712	beq	\$8386	
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	

High resolution ?  
No

```

0083C6 537900004F12      subq.w  #1,$4F12
0083CC 4A7900004F12      tst.w  $4F12
0083D2 6C18              bge    $93EC
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
00840E 670A              beq    $841A
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 6000004A6         bra    $88EA

```

Epson B/W dot-matrix printer ?  
No

008446 427900004F88	clr.w	\$4F88	
00844C 60000450	bra	\$889E	
008450 4A39000047CE	tst.b	\$47CE	ATARI color dot matrix printer ?
008456 675A	beq	\$84B2	No
008458 4A3900004ED8	tst.b	\$4ED8	High resolution ?
00845E 6652	bne	\$84B2	Yes
008460 4A7900004F88	tst.w	\$4F88	
008466 6616	bne	\$847E	
008468 2EBC00016D5E	move.l	#\$16D5E, (A7)	ESC 'X' 6
00846E 6100058E	bsr	\$89FE	Output string to the printer
008472 4A40	tst.w	D0	Error ?
008474 6706	beq	\$847C	No
008476 70FF	moveq.l	#-1, D0	Flag for error
008478 60000526	bra	\$89A0	Error, terminate
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	Output string to printer
008492 4A40	tst.w	D0	Error ?
008494 6706	beq	\$849C	No
008496 70FF	moveq.l	#-1, D0	Flag for error
008498 60000506	bra	\$89A0	Error, terminate
00849C 6014	bra	\$84B2	
00849E 2EBC00016D68	move.l	#\$16D68, (A7)	ESC 'X' 3
0084A4 61000558	bsr	\$89FE	Output string to printer
0084A8 4A40	tst.w	D0	Error ?
0084AA 6706	beq	\$84B2	No
0084AC 70FF	moveq.l	#-1, D0	Flag for error

0084AE 600004F0	bra \$89A0	Error, terminate
0084B2 4A39000047D0	tst.b \$47D0	Epson B/W dot-matrix printer?
0084B8 6708	beq \$84C2	No
0084BA 2EBC00016D6D	move.l #\$16D6D, (A7)	ESC 'L', Bit image 960 points/line
0084C0 6006	bra \$84C8	
0084C2 2EBC00016D71	move.l #\$16D71, (A7)	ESC 'Y', Bit image 960 points/line
0084C8 61000534	bsr \$89FE	Output string to printer
0084CC 4A40	tst.w D0	Error ?
0084CE 6706	beq \$84D6	No
0084D0 70FF	moveq.l #-1, D0	Flag for error
0084D2 600004CC	bra \$89A0	Error, terminate
0084D6 103900003E86	move.b \$3E86, D0	Get byte
0084DC 4880	ext.w D0	Low byte of the number
0084DE 3E80	move.w D0, (A7)	On stack
0084E0 610004C8	bsr \$89AA	Output
0084E4 4A40	tst.w D0	Error ?
0084E6 6706	beq \$84EE	No
0084E8 70FF	moveq.l #-1, D0	Flag for error
0084EA 600004B4	bra \$89A0	Error terminate
0084EE 103900003E88	move.b \$3E88, D0	Get byte
0084F4 4880	ext.w D0	High byte of the number
0084F6 3E80	move.w D0, (A7)	On stack
0084F8 610004B0	bsr \$89AA	Output
0084FC 4A40	tst.w D0	Error ?
0084FE 6706	beq \$8506	No
008500 70FF	moveq.l #-1, D0	Flag for error
008502 6000049C	bra \$89A0	Error, terminate
008506 13FC000100004EEA	move.b #1, \$4EEA	

```

00850E 23F9000046B800004EDC move.l $46B8,$4EDC
008518 33F9000047BC00004F12 move.w $47BC,$4F12
008522 4279000012E8 clr.w $12E8
008528 60000034C 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
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

```

height

008580 4240	clr.w	D0		height
008582 303900002608	move.w	\$2608,D0		
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		
0085FE 6722	beq	\$8622		
008600 4A7900004F1A	tst.w	\$4F1A		
008606 670C	beq	\$8614		
008608 20790000261A	move.l	\$261A,A0		
00860E 1010	move.b	(A0),D0		

High resolution ?  
No

p masks



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	
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	

p masks

00867A D2C9	add.w	A1,A1		
00867C D3F90000261A	add.l	\$261A,A1		
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	\$5A4		
0086A4 4A39000047CE	tst.b	\$47CE		
0086AA 670000DE	beq	\$878A		
0086AE 4A3900004ED8	tst.b	\$4ED8		
0086B4 660000D4	bne	\$878A		
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,A1.1),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		

p masks

ATARI color dot-matrix printer ?

No

High resolution ?

Yes

```

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 610001E8  bsr     $89AA
0087F4 4A40      tst.w  D0
0087F6 6706      beq     $87FE
0087F8 70FF      moveq.l #-1,D0
0087FA 600001A4  bra     $89A0

                                Output
                                Error, terminate

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
008828 6720      beq     $884A
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
008840 4A40      tst.w  D0
008842 6706      beq     $884A
008844 70FF      moveq.l #-1,D0

                                Epson B/W dot-matrix printer ?
                                No

                                Output
                                OK ?
                                Yes
                                Set flag

```

008846 60000158	bra \$89A0	Error, terminate
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	Output
00888E 4A40	tst.w D0	OK ?
008890 6706	beq \$8898	Yes
008892 70FF	moveq.l #-1,D0	Set flag
008894 6000010A	bra \$89A0	Error, terminate
008898 527900004F88	addq.w #1,\$4F88	
00889E 4A39000047CE	tst.b \$47CE	ATARI color dot-matrix printer ?
0088A4 670C	beq \$88B2	No
0088A6 4A3900004ED8	tst.b \$4ED8	High resolution ?
0088AC 6604	bne \$88B2	Yes
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" line spacing

0088C4 61000138	bsr \$89FE	Output string to printer
0088C8 4A40	tst.w D0	
0088CA 6706	beq \$88D2	
0088CC 70FF	moveq.l #-1,D0	Set flag
0088CE 600000D0	bra \$89A0	Error, terminate
0088D2 3EBC000A	move.w #\$A,(A7)	LF
0088D6 610000D2	bsr \$89AA	Output
0088DA 4A40	tst.w D0	OK ?
0088DC 6706	beq \$88E4	Yes
0088DE 70FF	moveq.l #-1,D0	Set flag
0088E0 600000BE	bra \$89A0	Error, terminate
0088E4 5279000047D2	addq.w #1,\$47D2	
0088EA 4A3900004EE8	tst.b \$4EE8	Printer resolution
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	Printer resolution
008908 673E	beq \$8948	
00890A 4247	clr.w D7	
00890C 6028	bra \$8936	
00890E 2EBC00016D7A	move.l #\$16D7A,(A7)	ESC '3' 1, 1/216" line spacing
008914 610000E8	bsr \$89FE	Output string to printer
008918 4A40	tst.w D0	Error on output ?
00891A 6706	beq \$8922	No
00891C 70FF	moveq.l #-1,D0	Set flag
00891E 60000080	bra \$89A0	Error, terminate

008922 3EBC000A	move.w #A,(A7)	LF
008926 61000082	bsr \$89AA	Output
00892A 4A40	tst.w D0	Error during output ?
00892C 6706	beq \$8934	No
00892E 70FF	moveq.l #-1,D0	Set flag
008930 6000006E	bra \$89A0	Error, terminate
008934 5247	addq.w #1,D7	Epson B/W dot-matrix printer ?
008936 4A39000047D0	tst.b \$47D0	No
00893C 6704	beq \$8942	
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" line spacing
00894E 610000AE	bsr \$89FE	Output string to printer
008952 4A40	tst.w D0	Error during output ?
008954 6704	beq \$895A	No
008956 70FF	moveq.l #-1,D0	Set flag
008958 6046	bra \$89A0	Error, terminate
00895A 3EBC000A	move.w #A,(A7)	LF
00895E 614A	bsr \$89AA	Output
008960 4A40	tst.w D0	Error during output ?
008962 6704	beq \$8968	No
008964 70FF	moveq.l #-1,D0	
008966 6038	bra \$89A0	Error, terminate
008968 3039000003E80	move.w \$3E80,D0	
00896E E340	asl.w #1,D0	
008970 48C0	ext.l D0	



```

008972 D1B9000046B8      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
00898C B079000012EA      cmp.w  $12EA,D0
008992 6200F8DC          bhi     $8270
008996 2EBC00016D83      move.l  #$16D83,(A7)
00899C 6160              bsr     $89FE
00899E 4240              clr.w  D0
0089A0 4A9F              tst.l  (A7)+
0089A2 4CDF30C0          movem.l (A7)+,D6-D7/A4-A5
0089A6 4E5E              unlk   A6
0089A8 4E75              rts

*****
0089AA 4E56FFFC          link   A6,#-4
0089AE 4A39000025FE      tst.b  $25FE
0089B4 6722              beq     $89D8
0089B6 102E0009          move.b  9(A6),D0
0089BA 4880              ext.w  D0
0089BC 3E80              move.w  D0,(A7)
0089BE 102E0009          move.b  9(A6),D0
0089C2 4880              ext.w  D0
0089C4 3F00              move.w  D0,-(A7)
0089C6 4EB9000008A2C     jsr     $8A2C
0089CC 548F              addq.l #2,A7
0089CE 4A40              tst.w  D0
0089D0 6604              bne     $89D6
0089D2 70FF              moveq.l #-1,D0
0089D4 6024              bra     $89FA
0089D6 6020              bra     $89F8

*****
Output a character

Printer port
RS232 ?
Character to output
Extend to word
On the stack
Character to output
Extend to word
And back on stack (?)
Output via Centronics port
Correct stack pointer
OK ?
Yes
Flag for error
Done
Error-free termination

```

0089D8 102E0009	move.b 9(A6),D0	Character to output
0089DC 4880	ext.w D0	Extend to word
0089DE 3E80	move.w D0,(A7)	And on stack
0089E0 102E0009	move.b 9(A6),D0	Character to output
0089E4 4880	ext.w D0	Extend to word
0089E6 3F00	move.w D0,-(A7)	And back on stack (?)
0089E8 4EB900008A46	jsr \$8A46	Output via RS-232
0089EE 548F	addq.l #2,A7	Correct stack pointer
0089F0 4A40	tst.w D0	OK ?
0089F2 6604	bne \$89F8	Yes
0089F4 70FF	moveq.l #-1,D0	Set error flag
0089F6 6002	bra \$89FA	
0089F8 4240	clr.w D0	
0089FA 4E5E	unlk A6	
0089FC 4E75	rts	
*****		
0089FE 4E56FFFC	link A6,#-4	Output string to printer
008A02 6018	bra \$8A1C	
008A04 206E0008	move.l 8(A6),A0	Get string address
008A08 1010	move.b (A0),D0	String character
008A0A 4880	ext.w D0	Extend to word
008A0C 3E80	move.w D0,(A7)	
008A0E 619A	bsr \$89AA	Output character
008A10 52AE0008	addq.l #1,8(A6)	Pointer to next character
008A14 4A40	tst.w D0	Error-free output ?
008A16 6704	beq \$8A1C	Yes
008A18 70FF	moveq.l #-1,D0	Error
008A1A 600C	bra \$8A28	Terminate output
008A1C 206E0008	move.l 8(A6),A0	Pointer to string
008A20 0C1000FF	cmp.b #\$FF,(A0)	\$FF as end indicator

008A24 66DE	bne \$8A04	Next character
008A26 4240	clr.w D0	OK
008A28 4E5E	unlk A6	
008A2A 4E75	rts	
*****		
008A2C 302F0006	move.w 6(A7),D0	Centronics output
008A30 48E71F1E	movem.l D3-D7/A3-A6,-(A7)	Character to output
008A34 3F00	move.w D0,-(A7)	Save registers
008A36 3F00	move.w D0,-(A7)	Character to output
008A38 9BCD	sub.l A5,A5	Clear A5
008A3A 6100E198	bsr \$6BD4	Output character to Centronics port
008A3E 584F	addq.w #4,A7	Correct stack pointer
008A40 4CDF78F8	movem.l (A7)+,D3-D7/A3-A6	Restore registers
008A44 4E75	rts	
*****		
008A46 302F0006	move.w 6(A7),D0	RS232 output
008A4A 48E71F1E	movem.l D3-D7/A3-A6,-(A7)	Character to output
008A4E 3F00	move.w D0,-(A7)	Save registers
008A50 3F00	move.w D0,-(A7)	Character to output
008A52 9BCD	sub.l A5,A5	Clear A5
008A54 6100E258	bsr \$6CAE	Output character via RS-232
008A58 584F	addq.w #4,A7	Correct stack pointer
008A5A 4CDF78F8	movem.l (A7)+,D3-D7/A3-A6	Restore registers
008A5E 4E75	rts	
*****		
008A60 207900002580	move.l \$2580,A0	VDI ESCAPE functions
008A66 3028000A	move.w 10(A0),D0	Pointer to CONTRL array
008A6A B07C0013	cmp.w #\$13,D0	Function number
008A6E 6236	bhl \$8AA6	Greater than 19 ?



```

*****
008AB6 61000448      bsr      $8F00
008ABA 207900002584  move.l  $2584,A0
008AC0 3010          move.w  (A0),D0
008AC2 C0F90000257E  mulu.w  $257E,D0
008AC8 33C00000255E  move.w  D0,$255E
008ACE 6000041E      bra      $8EEE

*****
008AD2 322F0006      move.w  6(A7),D1
008AD6 024100FF      and.w   $FF,D1
008ADA 600005D2      bra      $90AE

*****
008ADE 322F0006      move.w  6(A7),D1
008AE2 024100FF      and.w   $FF,D1
008AE6 2079000004A8  move.l  $4A8,A0
008AEC 4ED0          jmp      (A0)

*****
008AEE B27C0020      cmp.w   $20,D1
008AF2 6C0005BA      bge     $90AE
008AF6 B23C001B      cmp.b   $1B,D1
008AFA 660C          bne     $8B08
008AFC 23FC0000B4A000004A8  move.l  $8B4A,$4A8
008B06 4E75          rts

*****
008B08 5F41          subq.w  #7,D1
008B0A 6B22          bmi     $8B2E
008B0C B27C0006      cmp.w   #6,D1
008B10 6E1C          bgt     $8B2E

VDI ESC 101
Cursor off
Pointer to INTIN array
Get parameters
Times number of bytes per screen line

ascout
Get character from stack
Process only low byte
Output character

conout
Get character from stack
Process only low byte
Get conout vector
And execute routine

Standard conout
Control code ?
No, output character
ESC ?
No, process other CTRL codes
Conout vector to ESC processing

Process CTRL codes
Less than 7 ?
Ignore, RTS
Greater than 13 ?
Ignore, RTS

```

```

008B12 E349      lsl.w    #1,D1
008B14 307B100A   move.w   $B20(PC,D1.w),A0
008B18 D1FC00008B30 add.l    #8B30,A0
008B1E 4ED0      jmp      (A0)

*****
008B20 0000      dc.w     $8B30-$8B30
008B22 01AC      dc.w     $8CDC-$8B30
008B24 0004      dc.w     $8B34-$8B30
008B26 04AA      dc.w     $8FDA-$8B30
008B28 04AA      dc.w     $8FDA-$8B30
008B2A 04AA      dc.w     $8FDA-$8B30
008B2C 049E      dc.w     $8FCE-$8B30

*****
008B2E 4E75      rts

*****
008B30 6000E218   bra      $6D4A

*****
008B34 303900002560 move.w   $2560,D0
008B3A 0240FF78   and.w    #$FFF8,D0
008B3E 5040      addq.w   #8,D0
008B40 323900002562 move.w   $2562,D1
008B46 60000764   bra      $92AC

*****
008B4A 23FC00008AEE000004A8 move.l   #$8AEE,$4A8
008B54 927C0041   sub.w    #$41,D1
008B58 6BD4      bmi      $8B2E
008B5A B27C000C   cmp.w    #5C,D1

```

Bring to word processing  
Get relative address from table  
Add base address to it  
And execute corresponding routine

Jump table for control codes

7, BEL  
8, BS  
9, TAB  
10, LF  
11, VT  
12, FF  
13, CR

BEL  
Output tone

TAB  
Cursor column  
Convert to number divisible by 8  
And add 8  
Cursor line  
Reset cursor

Process character after ESC  
convert vector back to standard  
Minus 'A'  
Less, then ignore  
'M'

```

008B5E 6F50      ble    $8B80
008B60 B27C0018  cmp.w   #$18,D1
008B64 663C      bne    $8BA2
008B66 23FC0008B72000004A8 move.l  #$8B72,$4A8
008B70 4E75      rts

*****
008B72 927C0020      sub.w   #$20,D1
008B76 33C1000004AC  move.w   D1,$4AC
008B7C 23FC0008B88000004A8 move.l  #$8B88,$4A8
008B86 4E75      rts

*****
008B88 927C0020      sub.w   #$20,D1
008B8C 3001      move.w   D1,D0
008B8E 3239000004AC  move.w   $4AC,D1
008B94 23FC0008AAE000004A8 move.l  #$8AEE,$4A8
008B9E 6000070C      bra     $92AC

*****
008BA2 927C0021      sub.w   #$21,D1
008BA6 6B86      bml     $8B2E
008BA8 B27C0015      cmp.w   #$15,D1
008BAC 6F10      ble     $8BBE
008BAE 4E75      rts

*****
008BB0 E349      lsl.w   #1,D1
008BB2 307B1058      move.w   $8C0C(PC,D1.w),A0
008BB6 D1FC0008B2E  add.l   #$8B2E,A0
008BBC 4ED0      jmp     (A0)

```

To ESC table for capital letters  
'y' for set cursor ?  
No, test for lowercase letters  
conout vector for ESC Y

Process line after ESC Y  
Subtract offset  
And save line value  
Process conout vector to column

Process column after ESC Y  
Subtract offset  
Column value  
And line  
conout vector back to standard  
And set cursor

Test for ESC lowercase  
Subtract offset  
Ignore lowercase 'b'  
'w'  
Less than or equal, process sequence

ESC uppercase  
Code times 2 for word acces  
Get relative address from table  
Add base address  
And execute routine

```

*****
008BE E349      lsl.w  #1,D1
008BC 307B1064   move.w $8C26(PC,D1.w),A0
008BC4 D1FC0008B2E add.l  #$8B2E,A0
008BCA 4ED0      jmp   (A0)

*****
008BCC 23FC0008BD8000004A8 move.l  #$8BD8,$4A8
008BD6 4E75      rts

*****
008BD8 23FC0008BAEE000004A8 move.l  #$8AEE,$4A8
008BE2 927C0020   sub.w  #20,D1
008BE6 3001      move.w  D1,D0
008BE8 60000292   bra    $8E7C

*****
008BEC 23FC0008BF80000004A8 move.l  #$8BF8,$4A8
008BF6 4E75      rts

*****
008BF8 23FC0008BAEE000004A8 move.l  #$8AEE,$4A8
008C02 927C0020   sub.w  #20,D1
008C06 3001      move.w  D1,D0
008C08 6000027E   bra    $8E88

*****
008C0C 0166      dc.w    $8B2E-$8B2E
008C0E 017A      dc.w    $8B2E-$8B2E
008C10 0194      dc.w    $8B2E-$8B2E
008C12 01AE      dc.w    $8B2E-$8B2E
008C14 0162      dc.w    $8B2E-$8B2E

*****

```

ESC lowercase  
Code times 2 for word access  
Get relative address from table  
Add base address  
And execute routine

ESC b set character color  
Set conout vector

Set standrad conout vector  
Subtract offset

Set character color

ESC c set background color  
Set conout vector

Standard conout vector  
Subtract offset

Set background color

Address table for ESC upper case  
ESC A  
ESC B  
ESC C  
ESC D  
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
*****			
008C26 009E	dc.w	\$8B2E-\$8B2E	Address table for ESC lowercase
008C28 00BE	dc.w	\$8B2E-\$8B2E	ESC b
008C2A 0366	dc.w	\$8B2E-\$8B2E	ESC c
008C2C 0382	dc.w	\$8B2E-\$8B2E	ESC d
008C2E 03D2	dc.w	\$8B2E-\$8B2E	ESC e
008C30 0000	dc.w	\$8B2E-\$8B2E	ESC f
008C32 0000	dc.w	\$8B2E-\$8B2E	ESC g, rts
008C34 0000	dc.w	\$8B2E-\$8B2E	ESC h, rts
008C36 03F2	dc.w	\$8B2E-\$8B2E	ESC i, rts
008C38 040E	dc.w	\$8B2E-\$8B2E	ESC j
008C3A 0428	dc.w	\$8B2E-\$8B2E	ESC k
008C3C 0000	dc.w	\$8B2E-\$8B2E	ESC l
008C3E 0000	dc.w	\$8B2E-\$8B2E	ESC m, rts
008C40 0446	dc.w	\$8B2E-\$8B2E	ESC n, rts
008C42 029A	dc.w	\$8B2E-\$8B2E	ESC o
008C44 02A4	dc.w	\$8B2E-\$8B2E	ESC p
008C46 0000	dc.w	\$8B2E-\$8B2E	ESC q
008C48 0000	dc.w	\$8B2E-\$8B2E	ESC r, rts
008C4A 0000	dc.w	\$8B2E-\$8B2E	ESC s, rts
008C4C 0000	dc.w	\$8B2E-\$8B2E	ESC t, rts
008C4E 048C	dc.w	\$8B2E-\$8B2E	ESC u, rts
008C50 0496	dc.w	\$8B2E-\$8B2E	ESC v
			ESC w

```

*****
008C52 207900002580      move.l $2580,A0
008C58 317C00002008      move.w #2,8(A0)
008C5E 20790000258C      move.l $258C,A0
008C64 303900002550      move.w $2550,D0
008C6A 5240              addq.w #1,D0
008C6C 31400002          move.w D0,2(A0)
008C70 303900002552      move.w $2552,D0
008C76 5240              addq.w #1,D0
008C78 3080              move.w D0,(A0)
008C7A 4E75              rts

*****
008C7C 3F3C0014          move.w #$14,-(A7)
008C80 4E4E              trap #14
008C82 548F              addq.l #2,A7
008C84 4E75              rts

*****
008C86 6108              bsr $8C90
008C88 60000226          bra $8EB0

*****
008C8C 61000272          bsr $8F00

*****
008C90 615E              bsr $8CF0
008C92 6064              bra $8CF8

*****
008C94 323900002562      move.w $2562,D1
008C9A 67DE              beq $8C7A

*****
VDI ESC 1, get screen size
Pointer to CONTRL array
2 result values
Pointer to INTOUT array
Maximum cursor column (79)
plus 1 equals number of columns
As second result
Maximum cursor line (24)
plus 1 equals number of lines
As first result

*****
VDI ESC 17
Hardcopy

Put stack back in order

*****
VDI ESC 3, Enter alpha mode
ESC E, Clear Home
ESC e, Cursor on

*****
VDI ESC 2, Exit alpha mode
ESC f, Cursor off

*****
ESC E, Clear Home
ESC H, Cursor Home
ESC J, Clear rest of screen

*****
ESC A, Cursor up, VDI ESC 4
Cursor line
Zero, then done already

```

008C9C 5341	subq.w #1,D1	Subtract one
008C9E 303900002560	move.w \$2560,D0	Get cursor column
008CA4 60000606	bra \$92AC	And set cursor
*****		
008CA8 323900002562	move.w \$2562,D1	ESC B, Cursor down, VDI ESC 5
008CAE B27900002552	cmp.w \$2552,D1	Cursor line
008CB4 67C4	beq \$8C7A	Compare to maximum cursor line
008CB6 5241	addq.w #1,D1	Already in lowest line, done
008CB8 303900002560	move.w \$2560,D0	Increment by one
008CBE 600005EC	bra \$92AC	Cursor column
*****		
008CC2 303900002560	move.w \$2560,D0	Set cursor
008CC8 B07900002550	cmp.w \$2550,D0	ESC C, Cursor right, VDI ESC 6
008CCE 67AA	beq \$8C7A	Cursor column
008CD0 5240	addq.w #1,D0	Compare with maximum value (79)
008CD2 323900002562	move.w \$2562,D1	Increment by one
008CD8 600005D2	bra \$92AC	Get cursor line
*****		
008CDC 303900002560	move.w \$2560,D0	And set cursor
008CE2 6796	beq \$8C7A	ESC D, DEL, Cursor left, VDI ESC 7
008CE4 5340	subq.w #1,D0	Cursor column
008CE6 323900002562	move.w \$2562,D1	Already zero ?
008CEC 600005BE	bra \$92AC	Decrement by one
*****		
008CF0 7000	moveq.l #0,D0	Cursor line
008CF2 3200	move.w D0,D1	Set cursor
008CF4 600005B6	bra \$92AC	ESC H, Cursor Home, VDI ESC 8
*****		
008CF0 7000	moveq.l #0,D0	Column
008CF2 3200	move.w D0,D1	And line to zero
008CF4 600005B6	bra \$92AC	Set cursor

*****			ESC J, Clear rest of screen, VDI 9
008CF8 612A	bsr	\$8D24	ESC K, Clear rest of line
008CF9 323900002562	move.w	\$2562,D1	Cursor line
008D00 B27900002552	cmp.w	\$2552,D1	Compare with maximum cursor line
008D06 6700FF72	beq	\$8C7A	
008D0A 5241	addq.w	#1,D1	
008D0C 4841	swap	D1	
008D0E 323C0000	move.w	#0,D1	Maximum cursor line
008D12 343900002552	move.w	\$2552,D2	
008D18 4842	swap	D2	
008D1A 343900002550	move.w	\$2550,D2	Maximum cursor column (79)
008D20 60000436	bra	\$9158	
*****			ESC K, Clear rest of line, VDI ESC 10
008D24 08B9000300002576	bclr	#3,\$2576	Clear flag for line overflow
008D2C 40E7	move.w	SR, -(A7)	Save old value
008D2E 610001D0	bsr	\$8F00	ESC f, Cursor off
008D32 610001EC	bsr	\$8F20	ESC j, Save cursor position
008D36 323900002560	move.w	\$2560,D1	Cursor column
008D3C 08010000	btst	#0,D1	
008D40 6716	beq	\$8D58	
008D42 B27900002550	cmp.w	\$2550,D1	Compare with maximum value (79)
008D48 673A	beq	\$8D84	In last column, then output space
008D4A 323C0020	move.w	#20,D1	Space
008D4E 6100035E	bsr	\$90AE	Output
008D52 323900002560	move.w	\$2560,D1	Cursor column
008D58 4841	swap	D1	
008D5A 323900002562	move.w	\$2562,D1	Cursor line
008D60 3401	move.w	D1,D2	
008D62 4841	swap	D1	
008D64 4842	swap	D2	
008D66 343900002550	move.w	\$2550,D2	Maximum cursor column

008D6C 610003EA	bsr	\$9158	Restore flag
008D70 44DF	move.w	(A7)+,CCR	Not set ?
008D72 6708	beq	\$8D7C	Reset
008D74 08F9000300002576	bset	#3,\$2576	ESC k, Restore cursor position
008D7C 610001BE	bsr	\$8F3C	Re-enable cursor
008D80 6000016C	bra	\$8EEE	
008D84 323C0020	move.w	#\$20,D1	Space
008D88 61000324	bsr	\$90AE	Output
008D8C 60E2	bra	\$8D70	
*****			
008D8E 207900002584	move.l	\$2584,A0	VDI ESC 11, Set cursor
008D94 3210	move.w	(A0),D1	Pointer to INTIN array
008D96 5341	subq.w	#1,D1	Get line
008D98 30280002	move.w	2(A0),D0	Minus offset
008D9C 5340	subq.w	#1,D0	Get column
008D9E 6000050C	bra	\$92AC	Minus offset
*****			
008DA2 207900002580	move.l	\$2580,A0	VDI ESC 12, Text output
008DA8 30280006	move.w	6(A0),D0	Pointer to CONTRL array
008DAC 207900002584	move.l	\$2584,A0	Number of characters
008DB2 600E	bra	\$8DC2	Pointer to INTIN array
*****			
008DB4 3218	move.w	(A0)+,D1	Get pointer to D1
008DB6 48E78080	movem.l	D0/A0,-(A7)	Restore registers
008DBA 6100FD26	bsr	\$8AE2	Output character in D1
008DBE 4CDF0101	movem.l	(A7)+,D0/A0	Registers back
008DC2 51C8FFFO	dbra	D0,\$8DB4	Output next character
008DC6 4E75	rts		

```

*****
008DC8 08F9000400002576      bset    #4,$2576
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

```

ESC p, reverse on, VDI ESC 13  
Set flag for reverse

ESC q, reverse off, VDI ESC 14  
Clear flag for reverse

VDI ESC 15, Get cursor position  
Pointer to CONTRL array  
2 result values  
Pointer to INTOUT array  
Cursor line  
Plus offset  
As first result  
Cursor column  
Plus offset  
As second result

VDI ESC 16, Inquire tablet status  
Pointer to CONTRL array  
A result value  
Pointer to INTOUT array  
Tablet available

Pointer to INTIN array  
No result values  
Set graphic cursor

```

*****
008E2E 4EF9000FF78      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
008E48 7200            moveq.l #0,D1
008E4A 60000460         bra     $92AC
*****
*****
008E4E 610000B0         bsr     $8F00
008E52 323900002562     move.w  $2562,D1
008E58 6100056E         bsr     $93C8
008E5C 4240            clr.w  D0
008E5E 323900002562     move.w  $2562,D1
008E64 61000446         bsr     $92AC
008E68 60000084         bra     $8EEE
*****
*****
008E6C 61000092         bsr     $8F00
008E70 323900002562     move.w  $2562,D1
008E76 61000508         bsr     $9380
008E7A 60E0            bra     $8E5C
*****
*****
008E7C C07C000F         and.w  #5F,D0
008E80 33C000002558     move.w  D0,$2558
008E86 4E75            rts
*****

```

VDI ESC 19, Clear graphic cursor

ESC I, Cursor up, scroll if needed

Cursor line

Not in line 0, then cursor up

Save cursor column

ESC L, Insert line

Restore cursor column

Cursor line to zero

Set cursor

ESC L, Insert line

ESC f, Cursor off

Cursor line

Shift remainder of screen down

Cursor in column 0

Cursor line

Set cursor

Re-enable cursor

ESC M, Clear line

ESC f, Cursor off

Cursor line

Shift remainder of screen up

See above

Set character color

mod 16, 0..15

Save character color

```

*****
008E88 C07C000F          and.w  #$F,D0
008E8C 33C00002556      move.w  D0,$2556
008E92 4E75             rts

*****
008E94 610000DE          bsr     $8F74
008E98 343900002562      move.w  $2562,D2
008E9E 67F2             beq     $8E92
008EA0 5342             subq.w  #1,D2
008EA2 4842             swap    D2
008EA4 343900002550      move.w  $2550,D2
008EAA 7200             moveq.l  #0,D1
008EAC 600002AA          bra     $9158

*****
008EB0 4A7900002422      tst.w   $2422
008EB6 67DA             beq     $8E92
008EB8 427900002422      clr.w   $2422
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
008ED4 323900002562      move.w  $2562,D1
008EDA 61000031C        bsr     $91F8
008EDE 6000043E          bra     $931E

*****
008EE2 61EA             bsr     $8ECE
008EE4 08D00001          bset    #1,(A0)
008EE8 08D00002          bset    #2,(A0)
008EEC 4E75             rts

*****
Set background color
mod 16, 0..15
Save background color

ESC d, Clear screen to cursor
ESC o, Clear line to cursor
Cursor line
Already zero, done

Maximum cursor column

ESC e, Cursor on
Cursor already on ?
Yes, done
Flag for cursor on

Cursor column
Cursor line
Calculate cursor position
Invert character at cursor position

Invert character at cursor position

```



```

*****
008EEE 4A7900002422      tst.w  $2422      Cursor enabled ?
008EF4 679C             beq    $8E92      Yes, rts
008EF6 537900002422      subq.w #1,$2422
008EFC 67C0             beq    $8EBE      See above
008EFE 4E75             rts

*****
008F00 527900002422      addq.w #1,$2422      ESC f, Cursor off
008F06 41F900002576      lea    $2576,A0      Flag for cursor off
008F0C 089000002       bclr   #2,(A0)      Clear flag for cursor
008F10 6780             beq    $8E92      Cursor was already off, rts
008F12 08100000       btst   #0,(A0)
008F16 67B6             beq    $8ECE
008F18 089000001       bclr   #1,(A0)
008F1C 66B0             bne    $8ECE
008F1E 4E75             rts

*****
008F20 08F9000500002576 bset   #5,$2576      ESC j, Save cursor position
008F28 41F90000242E      lea    $242E,A0      Flag for cursor saved
008F2E 30F900002560      move.w $2560,(A0)+      Address of the temp. storage
008F34 30B900002562      move.w $2562,(A0)      Cursor column
008F3A 4E75             rts      Cursor line

*****
008F3C 08B9000500002576 bclr   #5,$2576      ESC k, Cursor to saved position
008F44 6700FDAA          beq    $8CF0      Was cursor position saved ?
008F48 41F90000242E      lea    $242E,A0      No
008F4E 3018             move.w (A0)+,D0      Address of the temp. storage
008F50 3210             move.w (A0),D1      Cursor column
008F52 60000358          bra    $92AC      Cursor line
                                Set cursor

```

*****	bsr	\$8F00	ESC l, delete line
0008F56 61A8			ESC f, turn cursor off
0008F58 323900002562	move.w	\$2562,D1	Cursor line
0008F5E 3401	move.w	D1,D2	
0008F60 4841	swap	D1	
0008F62 4241	clr.w	D1	
0008F64 4842	swap	D2	
0008F66 343900002550	move.w	\$2550,D2	Maximum cursor column
0008F6C 610001EA	bsr	\$9158	
0008F70 6000FEEA	bra	\$8E5C	Cursor in column zero
*****			ESC o, Clear line to cursor
0008F74 61A8	bsr	\$8F00	ESC f, Turn cursor off
0008F76 61A8	bsr	\$8F20	ESC j, Save cursor position
0008F78 343900002560	move.w	\$2560,D2	Cursor column
0008F7E 6730	beq	\$8FB0	Zero, then done
0008F80 08020000	btst	#0,D2	
0008F84 6610	bne	\$8F96	
0008F86 323C0020	move.w	#20,D1	Space
0008F8A 61000122	bsr	\$90AE	Output
0008F8E 343900002560	move.w	\$2560,D2	Cursor column
0008F94 5542	subq.w	#2,D2	
0008F96 4842	swap	D2	Cursor line
0008F98 343900002562	move.w	\$2562,D2	
0008F9E 3202	move.w	D2,D1	
0008FA0 4842	swap	D2	
0008FA2 4841	swap	D1	
0008FA4 4241	clr.w	D1	
0008FA6 610001B0	bsr	\$9158	ESC k, Restore cursor position
0008FAA 6190	bsr	\$8F3C	And turn cursor back on
0008FAC 6000FF40	bra	\$8EEE	Space :
0008FB0 323C0020	move.w	#20,D1	

008FB4 610000F8	bsr \$90AE	Output
008FB8 60F0	bra \$8FAA	
*****		
008FBA 08F9000300002576	bset #3,\$2576	ESC v, New line at end of line
008FC2 4E75	rts	Set flag
*****		
008FC4 08B9000300002576	bclr #3,\$2576	ESC w, No new line at end of line
008FCC 4E75	rts	Clear flag
*****		
008FCE 323900002562	move.w \$2562,D1	CR, Cursor in column zero
008FD4 4240	clr.w D0	Cursor line
008FD6 600002D4	bra \$92AC	Column to zero
*****		
008FDA 303900002562	move.w \$2562,D0	Set cursor
008FE0 B07900002552	cmp.w \$2552,D0	LF, (VT, FF), Cursor down
008FE6 6600FCC0	bne \$8CA8	Cursor line
008FEA 6100FF14	bsr \$8F00	Compare with maximum cursor line
008FEE 4241	clr.w D1	Not in lowest linme, just cursor down
008FF0 6100038E	bsr \$9380	ESC f, Turn cursor off
008FF4 6000FEF8	bra \$8EEE	Scroll screen down
*****		
008FF8 41F900002576	lea \$2576,A0	And turn cursor back on
008FFE 08100002	btst #2,(A0)	Flash cursor
009002 671E	beq \$9022	Address of the flag word
009004 08100000	btst #0,(A0)	Cursor on ?
009008 6718	beq \$9022	No
00900A 43F900002565	lea \$2565,A1	Cursor flashing ?
*****		
		No
		Address of the flash counter

```

009010 5311      subq.b #1,(A1)      Decrement counter
009012 660E      bne $9022          Not yet run down ?
009014 12B900002564 move.b $2564,(A1) Reload flash counter
00901A 08500001  bchg #1,(A0)      Invert cursor phase
00901E 6000FEAE  bra $8ECE          Turn cursor off

009022 4E75      rts

*****
009024 302F0004  move.w 4(A7),D0      Cursor configuration, XBIOS No. 21
009028 6BF8      bmi $9022          Get number from stack
00902A B07C0005  cmp.w #5,D0          Negative, ignore
00902E 6EF2      bgt $9022          Greater than 5 ?
009030 E340      asl.w #1,D0          Yes, ignore
009032 41F90000904A lea $904A,A0      Base address of the table
009038 D0FE0004  add.w $903E(PC,D0.w),A0 Plus relative address
00903C 4ED0      jmp (A0)          Execute function

*****
00903E 0000      dc.w $904A-$904A      Address of the routines
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          0 ESC f, Turn cursor off
*****
00904E 6000FE60  bra $8EB0          1 ESC e, Turn cursor on
*****

```

```

*****
009052 6100FEAC      bsr      $F00
009056 08ED0002576   bset     #0,$2576(A5)
00905C 6000FE90      bra      $EEEE

*****
009060 6100FE9E      bsr      $F00
009064 08AD0002576   bclr     #0,$2576(A5)
00906A 6000FE82      bra      $EEEE

*****
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          bhl      $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
0090A6 4243          clr.w    D3

```

2      ESC f, Turn cursor off  
Turn cursor back on

3      ESC f, Turn cursor off  
Turn cursor back on

4      Set cursor flash rate

5      Load cursor flash rate

Calculate font data for character in D1  
Smallest ASCII code in font  
Compare with character to output  
Character not in font  
Largest ASCII code in font  
Character not in font  
Load font offset pointer  
Code times two  
Yields bit number in font  
Divided by 8 yields byte number  
Pointer to font data  
Yields pointer to data for this character  
Flag for character in font

0090A8 4E75	rts		Character not present in font
0090AA 7601	moveq.l #1,D3		
0090AC 4E75	rts		
*****			
0090AE 61CE	bsr \$907E		ascout, ignore control codes
0090B0 6702	beq \$90B4		Character present in font ?
0090B2 4E75	rts		Yes
*****			
0090B4 22790000255A	move.l \$255A,A1		Screen address of the character
0090BA 3E3900002556	move.w \$2556,D7		Background color
0090C0 4847	swap D7		In upper word
0090C2 3E3900002558	move.w \$2558,D7		Character color in lower word
0090C8 0839000400002576	btst #4,\$2576		Reverse turned on ?
0090D0 6702	beq \$90D4		No
0090D2 4847	swap D7		Exchange character and background colors
0090D4 08B9000200002576	bclr #2,\$2576		
0090DC 40E7	move.w SR, -(A7)		Save status
0090DE 61000160	bsr \$9240		Calculate new position
0090E2 22790000255A	move.l \$255A,A1		Screen address
0090E8 303900002560	move.w \$2560,D0		Cursor column
0090EE 323900002562	move.w \$2562,D1		Cursor line
0090F4 61000252	bsr \$9348		Calculate relative screen address
0090F8 6732	beq \$912C		
0090FA 303900002554	move.w \$2554,D0		Number of bytes per character line
009100 C0C1	mulu.w D1,D0		
009102 22790000044E	move.l \$44E,A1		Address of the screen RAM
009108 D3C0	add.l D0,A1		
00910A 4240	clr.w D0		
00910C B27900002552	cmp.w \$2552,D1		Compare with maximum cursor line
009112 640A	bcc \$911E		
009114 D2F900002554	add.w \$2554,A1		plus number of bytes per character line

00911A 5241	addq.w #1,D1	
00911C 600E	bra \$912C	
00911E 48E7C040	movem.l D0-D1/A1, -(A7)	Save registers
009122 7200	moveq.l #0,D1	
009124 6100025A	bsr \$9380	
009128 4CDF0203	movem.l (A7)+,D0-D1/A1	Restore registers
00912C 23C90000255A	move.l A1,\$255A	Screen address
009132 33C000002560	move.w D0,\$2560	Cursor column
009138 33C100002562	move.w D1,\$2562	Cursor line
00913E 44DF	move.w (A7)+,CCR	Restore status
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	bar \$91F8	Calculate cursor position
009162 E242	asr.w #1,D2	
009164 36390000257C	move.w \$257C,D3	Number of screen planes (1,2 or 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 per screen line
00917E 94C1	sub.w D1,A2	
009180 3202	move.w D2,D1	
009182 4842	swap D2	

009184	5242	addq.w	#1,D2	Times height of a character
009186	C4F90000254E	mulu.w	\$254E,D2	
00918C	5342	subq.w	#1,D2	
00918E	4280	clr.l	D0	
009190	3A3900002556	move.w	\$2556,D5	Background color
009196	0C7900020000257C	cmp.w	#2,\$257C	Number of screen levels
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	51CAFFFF2	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

*****
0091F8 363900002550  move.w $2550,D3
0091FE B640      cmp.w D0,D3
009200 6A02      bpl $9204
009202 3003      move.w D3,D0
009204 363900002552  move.w $2552,D3
00920A B641      cmp.w D1,D3
00920C 6A02      bpl $9210
00920E 3203      move.w D3,D1
009210 36390000257C  move.w $257C,D3
009216 3A00      move.w D0,D5
009218 08850000  bclr #0,D5
00921C C6C5      mulu.w D5,D3
00921E 08000000  btst #0,D0
009222 6702      beq $9226
009224 5283      addq.l #1,D3
*****
Calculate cursor position (D0/D1)
Maximum cursor column

Column value smaller ?
Else use max value
Maximum cursor line

Line value smaller ?
Else use maximum value
Number of screen planes
Column to D5

Times maximum value

```

009226 3A3900002554	move.w \$2554,D5	Number of bytes per character line
00922C CAC1	mulu.w D1,D5	Times line value
00922E 2279000044E	move.l \$44E,A1	Base address of the screen RAM
009234 D3C5	add.l D5,A1	Plus offset for line
009236 D3C3	add.l D3,A1	Plus offset for column
009238 D2F90000255E	add.w \$255E,A1	Plus number of bytes per raster line
00923E 4E75	rts	
009240 34790000256E	move.w \$256E,A2	formwidth
009246 36790000257E	move.w \$257E,A3	Bytes per screen line
00924C 38390000254E	move.w \$254E,D4	Height of a character
009252 5344	subq.w #1,D4	
009254 3C390000257C	move.w \$257C,D6	Number of screen planes
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 DACE      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 DACE      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
0092B2 6306          bls $92BA
0092B4 303900002550  move.w $2550, D0
0092BA B27900002552  cmp.w $2552, D1
0092C0 6306          bls $92C8
0092C2 323900002552  move.w $2552, D1
0092C8 33C000002560  move.w D0, $2560
0092CE 33C100002562  move.w D1, $2562
0092D4 41F900002576  lea $2576, A0
0092DA 08100002      btst #2, (A0)
0092DE 6732          beq $9312
0092E0 08100000      btst #0, (A0)
0092E4 670A          beq $92F0
*****
Set cursor
Compare column with maximum value
Smaller ?
Else use maximum value
Compare line with maximum value
Smaller ?
Else use maximum value
Save cursor column
Save cursor line
Address of the cursor flag

Character inverted at old position?
Yes

```

```

0092E6 08900002      bclr    #2,(A0)      Screen address of old cursor
0092EA 08100001      btst     #1,(A0)      Invert character at cursor position
0092EE 671E          beq      $930E      Calculate new cursor address
0092F0 22790000255A  move.l  $255A,A1      Screen address of the cursor
0092F6 6126          bsr      $931E      Invert character at cursor position
0092F8 6100FEFE      bsr      $91F8      Flag for character is inverted
0092FC 23C90000255A  move.l  A1,$255A      Calculate new cursor position
009302 611A          bsr      $931E      Screen address of the cursor position
009304 08F9000020002576 bset     #2,$2576      Invert character at cursor position
00930C 4E75          rts

00930E 08D00002      bset     #2,(A0)      Bytes per screen line
009312 6100FEE4      bsr      $91F8      Height of a screen line
009316 23C90000255A  move.l  A1,$255A      As dbra counter
00931C 4E75          rts      Number of screen planes
                                As dbra counter
                                Counter for raster lines
                                Screen address of the character
                                Invert a raster line of the character
                                Pointer to next raster line
                                Next raster line
                                Next screen level

*****
00931E 34790000257E  move.w  $257E,A2
009324 38390000254E  move.w  $254E,D4
00932A 5344          subq.w  #1,D4
00932C 3C390000257C  move.w  $257C,D6
009332 5346          subq.w  #1,D6
009334 3A04          move.w  D4,D5
009336 2849          move.l  A1,A4
009338 4614          not.b  (A4)
00933A D8CA          add.w  A2,A4
00933C 51CDFFFA      dbra    D5,$9338
009340 5449          addq.w  #2,A1
009342 51CEFFFO      dbra    D6,$9334
009346 4E75          rts
*****

```

009348 B07900002550	cmp.w	\$2550,D0	Maximum cursor column
00934E 6612	bne	\$9362	
009350 0839000300002576	btst	#3,\$2576	Flag for line overflow set ?
009358 6604	bne	\$935E	Yes
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 36390000257C	move.w	\$257C,D3	Number of screen levels
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	Scroll screen down at line D1
009386 363900002554	move.w	\$2554,D3	Address of the screen RAM
00938C C6C1	mulu.w	D1,D3	Number of bytes per character line
00938E 47F33000	lea	0(A3,D3.w),A3	Multiply by number of lines
009392 4441	neg.w	D1	Yields address of the line
009394 D27900002552	add.w	\$2552,D1	Current line
00939A 363900002554	move.w	\$2554,D3	Maximum cursor line
			Number of bytes per character line

0093A0 45F33000	lea	0(A3,D3.w),A2	Yields address of line 1
0093A4 C6C1	mulu.w	D1,D3	Number of bytes to move
0093A6 E443	asr.w	#2,D3	Number of long words
0093A8 6002	bra	\$93AC	
0093AA 26DA	move.l	(A2)+,(A3)+	Copy screen lines
0093AC 51CBFFFC	dbra	D3,\$93AA	
0093B0 323900002552	move.w	\$2552,D1	Maximum cursor line
0093B6 3401	move.w	D1,D2	
0093B8 4841	swap	D1	
0093BA 4842	swap	D2	
0093BC 4241	clr.w	D1	
0093BE 343900002550	move.w	\$2550,D2	Maximum cursor column
0093C4 6000FD92	bra	\$9158	
*****			
0093C8 26790000044E	move.l	\$44E,A3	Scroll screen at line D1 up
0093CE 3639000002552	move.w	\$2552,D3	Address of the screen RAM
0093D4 C6F9000002554	mulu.w	\$2554,D3	Maximum cursor line
0093DA 47F33000	lea	(A3,D3.w),A3	Mult by number of bytes per character line
0093DE 3639000002554	move.w	\$2554,D3	Yields address of last character line
0093E4 45F33000	lea	(A3,D3.w),A2	Number of bytes per line
0093E8 3001	move.w	D1,D0	Yields address of line 1
0093EA 4440	neg.w	D0	Current line
0093EC D079000002552	add.w	\$2552,D0	Add maximum cursor line
0093F2 C6C0	mulu.w	D0,D3	
0093F4 E443	asr.w	#2,D3	Divide by 4 for long word transfer
0093F6 6002	bra	\$93FA	
0093F8 2523	move.l	-(A3),-(A2)	Copy screen lines
0093FA 51CBFFFC	dbra	D3,\$93F8	
0093FE 60B6	bra	\$93B6	

```

*****
009400 207900002584      move.l  $2584,A0
009406 2050             move.l  (A0),A0
009408 30280052          move.w  82(A0),D0
00940C 33C00000254E      move.w  D0,$254E
009412 32390000257E      move.w  $257E,D1
009418 C2C0             mulu.w  D0,D1
00941A 33C100002554      move.w  D1,$2554
009420 7200             moveq.l  #0,D1
009422 323900002578      move.w  $2578,D1
009428 82C0             divu.w  D0,D1
00942A 5341             subq.w  #1,D1
00942C 33C100002552      move.w  D1,$2552
009432 7200             moveq.l  #0,D1
009434 323900002570      move.w  $2570,D1
00943A 82E80034          divu.w  52(A0),D1
00943E 5341             subq.w  #1,D1
009440 33C100002550      move.w  D1,$2550
009446 33E800500000256E  move.w  80(A0),$256E
00944E 33E800240000256C  move.w  36(A0),$256C
009456 33E800260000256A  move.w  38(A0),$256A
00945E 23E8004C00002566  move.l  76(A0),$2566
009466 23E8004800002572  move.l  72(A0),$2572
00946E 4E75             rts
*****
VDI ESC 102, initialize font parameters
Pointer to INTIN array
Address of the font header
formheight, height of a character
merken
Bytes per screen line
Times height of character
Yields bytes per character line

Number of raster lines on the screen
Divide by font height
Minus
Yields maximum cursor line

Screen width in bits
Divide by maximum character width
Minus 1
Yields maximum cursor column
formwidth, width of the font
Smallest ASCII code in font
Largest ASCII code in font
Pointer to font data
Pointer to offset table

```

```

*****
00F6C4 10390000044C      move.b $44C,D0
00F6CA C07C0003      and.w #3,D0
00F6CE B07C0003      cmp.w #3,D0
00F6D2 6604      bne $F6D8
00F6D4 303C0002      move.w #2,D0
00F6D8 3F00      move.w D0,-(A7)
00F6DA 6100007E      bsr $F75A
00F6DE 301F      move.w (A7)+,D0
00F6E0 41F900017EAA      lea $17EAA,A0
00F6E6 B07C0002      cmp.w #2,D0
00F6EA 6606      bne $F6F2
00F6EC 41F900018906      lea $18906,A0
00F6F2 61009D14      bsr $9408
00F6F6 33FCFF00002558      move.w $FFFF,$2558
00F6FE 7000      moveq.l #0,D0
00F700 33C000002556      move.w D0,$2556
00F706 33C000002560      move.w D0,$2560
00F70C 33C000002562      move.w D0,$2562
00F712 33C00000255E      move.w D0,$255E
00F718 20790000044E      move.l $44E,A0
00F71E 23C80000255A      move.l A0,$255A
00F724 13FC000100002576      move.b #1,$2576
00F72C 13FC001E00002565      move.b #$1E,$2565
00F734 13FC001E00002564      move.b #$1E,$2564
00F73C 33FC000100002422      move.w #1,$2422
00F744 323C1F3F      move.w $1F3F,D1
00F748 20C0      move.l D0,(A0)+
00F74A 51C9FFFC      dbra D1,$F748
00F74E 23FC00008AEE000004A8      move.l #$8AEE,$4A8
00F758 4E75      rts
*****
Initialize screen output
sshiftd, screen resolution
Isolate bits 0 and 1
3 ?
No
Replace with 2 (high resolution)
Save resolution
Set parameters for screen resolution
Restore resolution
Address of the 8x8 system font
High resolution ?
No
Address of the 8x16 system font
Initialize font data
Character color to black
Background color white
Cursor column zero
Cursor line zero
Address of the video RAM
As screen address of the cursor
Set cursor flag
Cursor flash counter to 30
Cursor flash rate to 30
Set flag for cursor on
8000 long words
Clear screen
conout vector to standard

```



```

*****
00F75A 7200          moveq.l #0,D1
00F75C 123B0030      move.b $F78E(PC,D0.w),D1
00F760 33C10000257C  move.w D1,$257C
00F766 123B0029      move.b $F791(PC,D0.w),D1
00F76A 33C10000257E  move.w D1,$257E
00F770 33C10000257A  move.w D1,$257A
00F776 E340          asl.w #1,D0
00F778 323B001A      move.w $F794(PC,D0.w),D1
00F77C 33C100002578  move.w D1,$2578
00F782 323B0016      move.w $F79A(PC,D0.w),D1
00F786 33C100002570  move.w D1,$2570
00F78C 4E75          rts

*****
00F78E 040201          dc.b 4,2,1
00F791 A0A050          dc.b 160,160,80
00F794 00C800C80190   dc.w 200,200,400
00F79A 014002800280   dc.w 320,640,640

*****
Initialize screen output
D0 contains screen resolution
Get number of screen planes
And save
Get bytes per screen line
And save

Get screen height
And save
Get screen height
And save

Screen parameters
Number of screen planes
Bytes per screen line
Screen heights
Screen widths

```

Note: This BIOS listing contains some of the most important sections of TOS Version 1. Later versions of TOS may have some minor differences, but this listing should still prove valuable.

# Chapter Four

## Appendix

- 4.1 The System Fonts**
- 4.2 Alphabetical listing of GEMDOS functions**

2000

2000

2000

## **4.1 The System Fonts**

The operating system contains three system fonts for character output.

The 6X6 font is used by the Icons, the 8X8 font is the standard font for output on the color monitor, and the 8X16 font is used for the monochrome monitor output. The chart on the next page includes the characters with the ASCII codes 1 to 255.

## 6X6 System Font

[illegible]

## 8X8 System Font

[illegible]

## 8X16 System Font

0123456789;=<>?@ABCDEFGHIJKLMNPO  
 QRSTUVWXYZ[\]^\_`abcdefgijklmnopqr  
 rstuvwxyz{|}~¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊË  
 ÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèé  
 êëìíîïðññòóôõö÷øùúûüýþÿ

## 4.2 Alphabetical listing of GEMDOS functions

Name	Opcode (hex)	Page Number
Cauxin	03	106
Cauxis	12	111
Cauxos	13	111
Cauxout	04	106
Cconin	01	105
Cconis	0B	110
Cconos	10	111
Cconout	02	106
Cconrs	0A	109
Cconws	09	109
Cnecin	08	108
Cprnos	11	111
Cprnout	05	107
Crawcin	07	108
Crawio	06	107
Dcreate	39	117
Ddelete	3A	118
Dfree	36	116
Dgetdrv	19	111
Dgetpath	47	128
Dsetdrv	0E	110
Dsetpath	3B	119
Fattrib	43	126
Fclose	3E	122
Fcreate	3C	120
Fdateime	57	137
Fdelete	41	124
Fdup	45	128
Fforce	46	128
Fgetdta	2F	115
Fopen	3D	121
Fread	3F	123
Frename	56	136
Fseek	42	125
Fsetdta	1A	112
Fsfirst	4E	134
Fsnext	4F	136
Fwrite	40	124

---

Malloc	48	129
Mfree	49	130
Mshrink	4A	131
Pexec	4B	132
Pterm	4C	133
Pterm0	00	105
Ptermres	31	115
Super	20	112
Sversion	30	115
Tgetdate	2A	113
Tgettime	2C	114
Tsetdate	2B	114
Tsettime	2D	115

## **Index**





---

ACIA 6850--see Asynchronous Communications Interface Adapter  
address bus, 7, 8  
asynchronous bus control, 8-9  
    ADDRESS STROBE(AS), 8  
    DTACK, 9, 10, 12  
    LOWER DATA STROBE(LDS), 8  
    READ/WRITE(R/W), 8  
    UPPER DATA STROBE(UDS), 8  
Asynchronous Communications Interface Adapter(ACIA), 41-47, 62-63  
    pins, 41-44  
    registers, 45-47  
        control register, 45-46  
        status registers, 46-47

BANK, 55  
Basic Input Output System(BIOS), 140-154, 242, 246, 247  
    listing, 268-442  
BCD--see Binary Coded Decimal  
BERR, 11, 12, 15  
BG--see Bus Grant  
BGACK--see Bus Grant Acknowledge  
BGO--see Bus Grant Out  
Binary Coded Decimal (BCD), 4  
BIOS--see Basic Input Output System  
BLANK, 15  
BR--see Bus Request  
Bus Grant(BG), 10, 13  
Bus Grant Acknowledge(BGACK), 10, 13  
Bus Grant Out(BGO), 13  
Bus Request(BR), 10, 13

cartridge slot, 96  
Centronics interface, 88-89  
CLK, 11

data bus, 7  
data registers, 4  
Data Request(DR), 22  
DE--see Display Enable  
Digital Research, 103  
Direct Memory Access(DMA), 8-9, 12-13, 18-19, 25, 58-59, 99-100  
Display Enable(DE), 15

DMA--see Direct Memory Access  
DR--see Data Request

exception vectors, 234-236

FDC--see Floppy Disk Controller  
Floppy Disk Controller(FDC), 20-27  
    Command Register(CR), 24  
    Data Register(DR), 24  
    Sector Register(SR), 24  
    Status Register(STR), 24  
    Track Register(TR), 24

floppy disk interface, 97-98

GEM graphics, 206-233  
    high-res, 209-212  
    line-A opcodes, 229-233  
    line-A variables, 226-228  
    lo-res, 208-209  
    medium-res, 209-210  
GEM graphics -- commands, 213-233  
    BITBLT, 218  
    COPY RASTER FORM, 225-226  
    DRAW SPRITE, 224-225  
    FILLED POLYGON, 217  
    FILLED RECTANGLE, 216  
    GET PIXEL, 213  
    HIDE CURSOR, 223  
    HORIZONTAL LINE, 215  
    Initialize, 213  
    LINE, 214  
    PUT PIXEL, 213  
    SHOW MOUSE, 223  
    TEXTBLT, 218-222  
    TRANSFORM MOUSE, 223-224  
    UNDRAW SPRITE, 224

GEMDOS, 103-139, 242  
    functions, 104-138  
    error messages, 139

GLUE, 13-15, 18, 69

HALT, 11, 12  
HSYNC, 15

- LACK, 13
- integrated circuits, 3-63
- INTEL, 3
- interrupts, 7, 10, 236-241
- I/O registers, 55-63
  - ACIAs, 62
  - DMA/Disk Controller, 58-59
  - keyboard, 62
  - MFP 68901, 60-61
  - MIDI, 62, 240
  - sound chip, 59-60
  - Video Display Register, 56-58
- keyboard control, 67-71, 74-84
- longword, 7
- Memory Management Unit(MMU), 11, 13, 15-16, 18, 55
- memory maps, 62-63
- MFP 68901--see Multi-Function Peripheral
- MFPINT, 13
- MIDI --see Musicial Instrument Digital Interface
- MMU--see Memory Management Unit
- Motorola 68000 microprocessor, 3-12, 255-267
  - registers, 4-6
  - exceptions, 7
  - connections, 7-12
  - instruction set, 255-267
- mouse, 71-74
- MS-DOS, 104, 180
- Multi-Function Peripheral(MFP 68901), 28-40, 60-61, 90, 171, 239, 241
  - Active Edge Register(AER), 32
  - connections, 28-32
  - Data Direction Register(DDR), 32
  - General Purpose I/O Interrupt Port(GPIP), 32
  - Interrupt Enable Register(IERA,IERB), 33
  - Interrupt In-Service Register(ISRA,ISRB), 34
  - Interrupt Mask Register(IMRA,IMRB), 34
  - Interrupt Pending Register(IPRA,IPRB), 33-34
  - Receiver Status Register(RSR), 38-39
  - registers, 32-40
  - Synchronous Character Register(SCR), 37

---

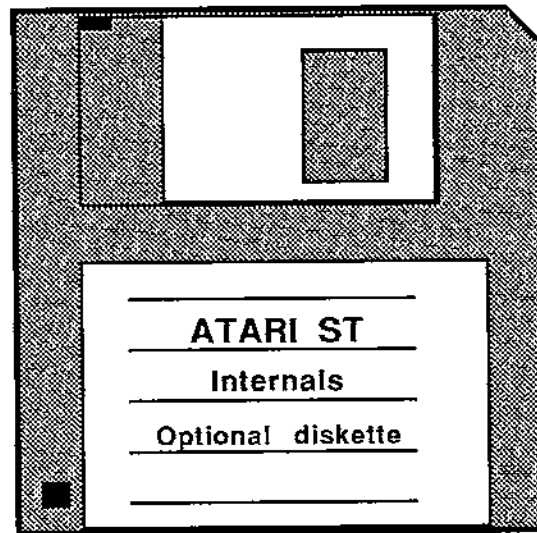
Timer A/B Control Register(TACR,TBCR), 35  
Timers C and D Control Register(TCDCR), 36  
Timer Data Registers (TADR,TBDR,TCDR,TDDR), 37  
Transmitter Status Register(TSR), 39-40  
UCR/USART, 37-38  
UDR/USART, 40  
Vector Register(VR), 34  
Musical Instruement Digital Interface(MIDI), 93-95, 170  
  
NMI--see non-maskable interrupt  
non-maskable interrupt (NMI), 6, 13, 237  
  
operating system, 103  
  
PSG (Programmable Sound Generator)--see YM-2149 Sound Generator  
  
RESET, 11, 12  
RS-232 interface, 90-92, 240-241  
  
SHIFTER, 13, 15, 17, 18  
status register, 6  
supervisor mode, 4, 6, 7, 234  
synchronous bus control, 9  
    E, 9  
    Valid Memory Address(VMA), 9  
    Valid Peripheral Address(VPA), 9, 10  
  
system fonts, 443-446  
system variables, 247-254  
  
Tramiel Operating System(TOS),103  
  
UNIX,104  
user byte ,4  
user mode ,4, 6, 7, 234  
  
video interface, 85-87  
VSYNC, 15  
VT52 emulator, 242-246  
  
WD 1772, 20-27  
word, 7  
word access, 8

XBIOS, 155-205

YM-2149 Sound Generator, 48-54

- attack/decay/sustain/release(ADSR), 48
- digital/analog(D/A) converter, 48
- registers, 52-54

## Optional Diskette



For your convenience, the program listings contained in this book are available on an SF354 formatted floppy disk. You should order the diskette if you want to use the programs, but don't want to type them in from the listings in the book.

All programs on the diskette have been fully tested. You can change the programs for your particular needs. The diskette is available for \$14.95 plus \$2.00 (\$5.00 foreign) for postage and handling.

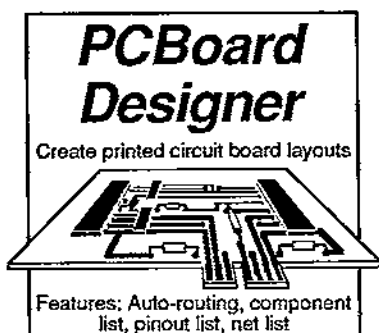
When ordering, please give your name and shipping address. Enclose a check, money order or credit card information. Mail your order to:

Abacus Software  
P.O. Box 7219  
Grand Rapids, MI 49510

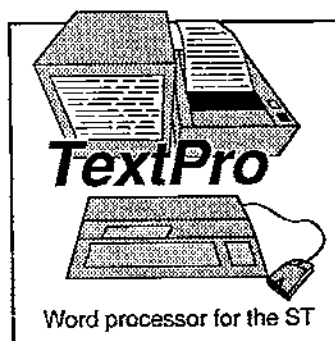
Or for fast service, call 616/241-5510.

# PROFESSIONAL DUCTIVITY

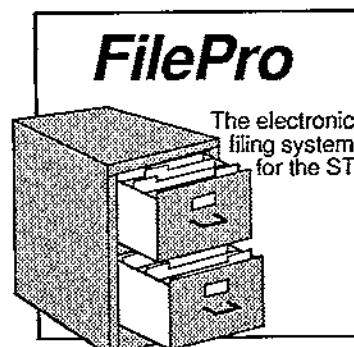
*New ST software from a name you can count on...*



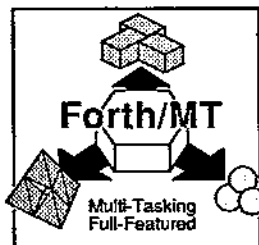
**PCBoard Designer**  
Interactive, computer-aided design package that automates layout of printed circuit boards. Auto-routing with 45° or 90° traces; two-sided boards; pin-to-pin, pin-to-BUS or BUS-to-BUS. Rubberbanding of components during placement. Prints board layout, pinout, component list, net list. Output to Epson printer at 2:1. Pays for itself after first designed board. **\$395.00**



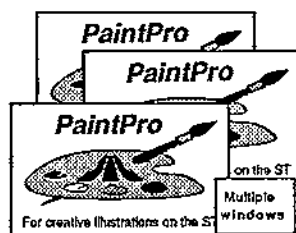
**ST TextPro**  
Wordprocessor with professional features and easy-to-use Full-screen editing with mouse or keyboard shortcuts. High speed input, scrolling and editing; sideways printing; multi-column output; flexible printer installation; automatic index and table of contents; up to 180 chars/line; 30 definable function keys; metafile output; much more. **\$49.95**



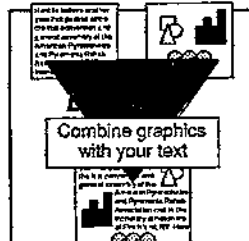
**ST FilePro**  
A simple-to-use and versatile database manager. Features help screens; lightning-fast operation; tailorable display using multiple fonts; user-definable edit masks; capacity up to 64,000 records. Supports multiple files. RAM-disk support for 1040ST. Complete search, sort and file subsetting. Interfaces to TextPro. Easy printer control. **\$49.95**



**ST Forth/MT**  
Powerful, multi-tasking Forth for the ST. A complete, 32-bit implementation based on Forth-83 standard. Development aids: full screen editor, monitor, macro assembler. 1500+ word library. TOS/LINEA commands. Floating point and complex arithmetic. Available Sept. '86. **\$49.95**



**ST PaintPro**  
A GEM™ among ST drawing programs. Very friendly, but very powerful. A must for everyone's artistic or graphics needs. Use up to three windows. Free-form sketching; lines, circles, ellipses, boxes, text, fill, copy, move, zoom, spray, paint, erase, undo, help. **\$49.95**



**ST Text Designer**  
An ideal package for page layout on the ST. Accepts prepared text files from TextPro or other ASCII wordprocessors. Performs block operations—copy, move, columns. Merges bit-mapped graphics. Tools to add borders & separator lines, more. Available September '86. **\$49.95**



**ST AssemPro**  
Professional developer's package includes editor, two-pass interactive assembler with error locator, online help including instruction address mode and GEM parameter information, monitor-debugger, disassembler and 68020 simulator, more. Available Sept. '86. **\$59.95**

ST and 1040ST are trademarks of Atari Corp.  
GEM is a trademark of Digital Research Inc.

## Abacus Software

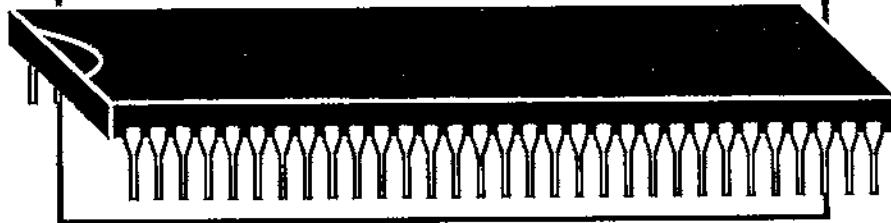
P.O. Box 7219 Dept. N9 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

Call now for the name of your nearest dealer. Or order directly from ABACUS with your MasterCard, VISA, or Amex card. Add \$4.00 per order for postage and handling. Foreign add \$10.00 per item. Other software and books coming soon. Call or write for your free catalog. Dealer inquiries welcome—over 1400 dealers nationwide.



# ***AssemPro***

The complete 68000  
assembler development  
package for the ST

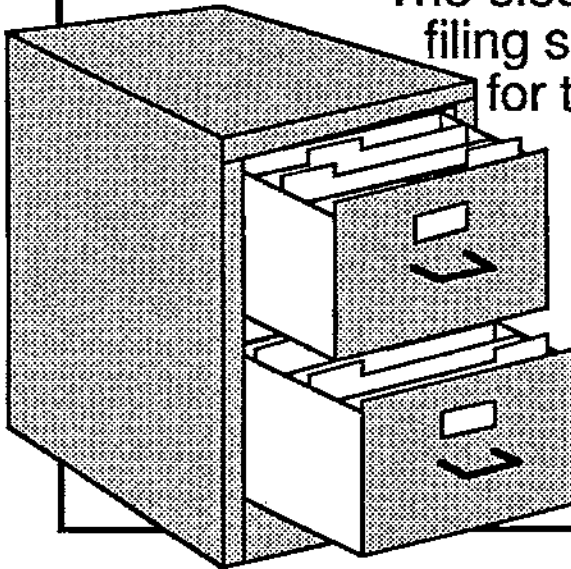


**AssemPro** is the professional developer's package for programming in 68000 assembly language on the ST. The package includes: editor, two-pass interactive assembler with error locator, online help including instruction address mode and GEM parameter information, monitor-debugger, disassembler and 68020 simulator.

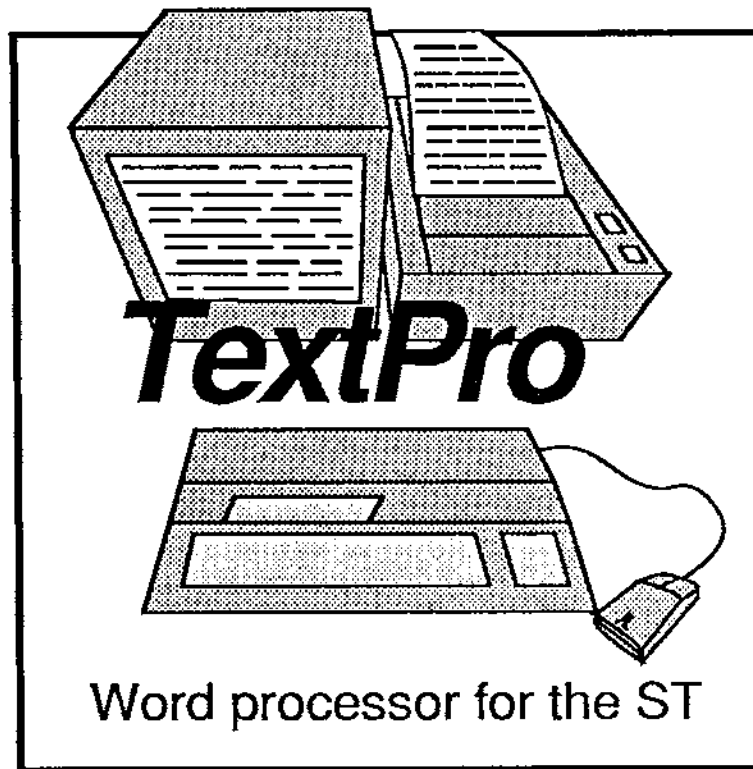
\$59.95

# **FilePro**

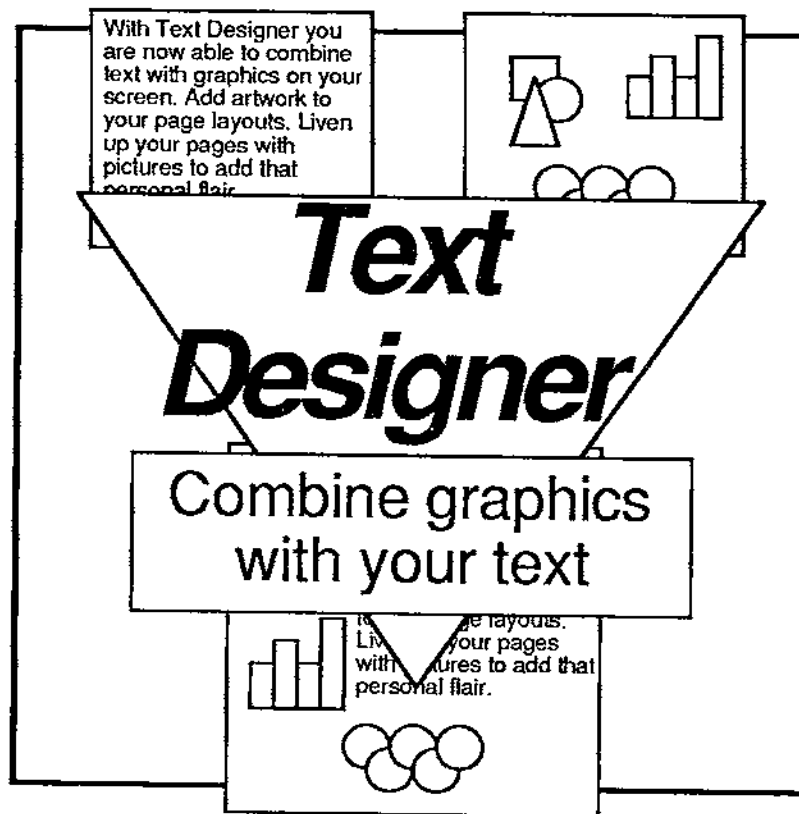
The electronic  
filing system  
for the ST



**FilePro** is a simple to use but flexible data manager. The drop-down menus allow you to quickly define your file and enter your information through screen templates. **FilePro** has many unique features: store data items in different type styles; create subsets of a file; change file definition and format; includes and supports a RAM disk for high speed operation. **FilePro** also has a fast search and sort capabilities, can handle records up to 64,000 characters in length, allows numerical values with up to 15 significant digits, accesses up to 4 files simultaneously, indexes up to 20 different fields per file and has complete, built-in reporting capabilities. \$49.95

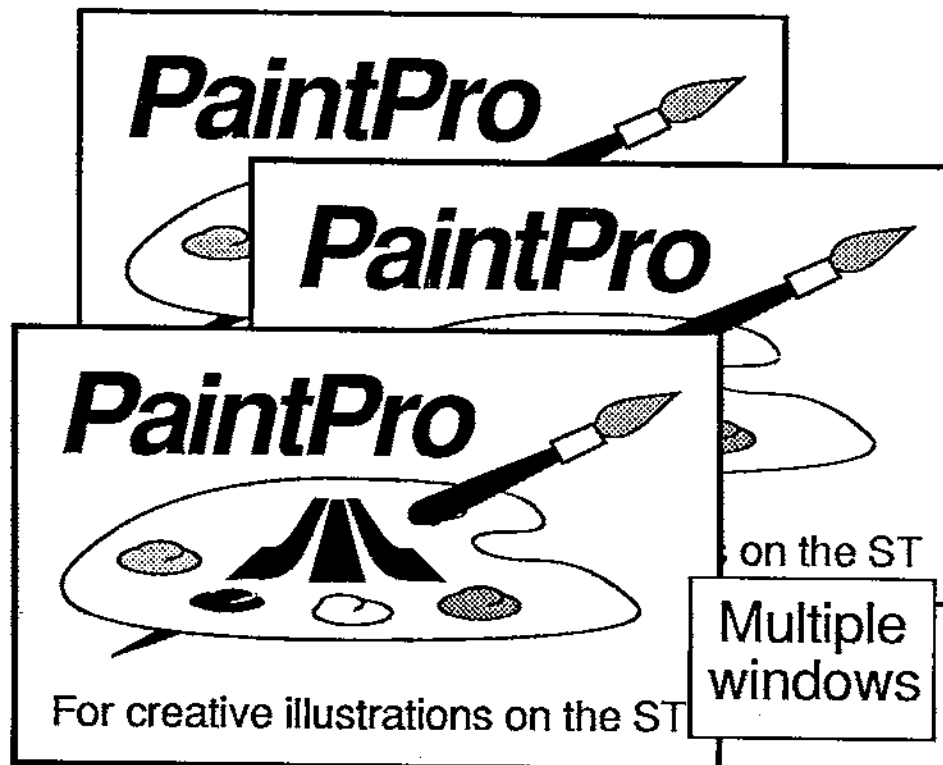


**TextPro** is a professional quality wordprocessor. In addition to the standard options found in most other word processors, **TextPro** features multi-column output automatic indexing and table of contents, sideways printing (to Epson printers), up to 30 user definable function keys, mode for editing C language source programs and flexible printer driver installation. It is designed with fast entry of text in mind. The advanced **TextPro** user can substitute shortcut keyboard commands for the drop-down menu commands. \$49.95



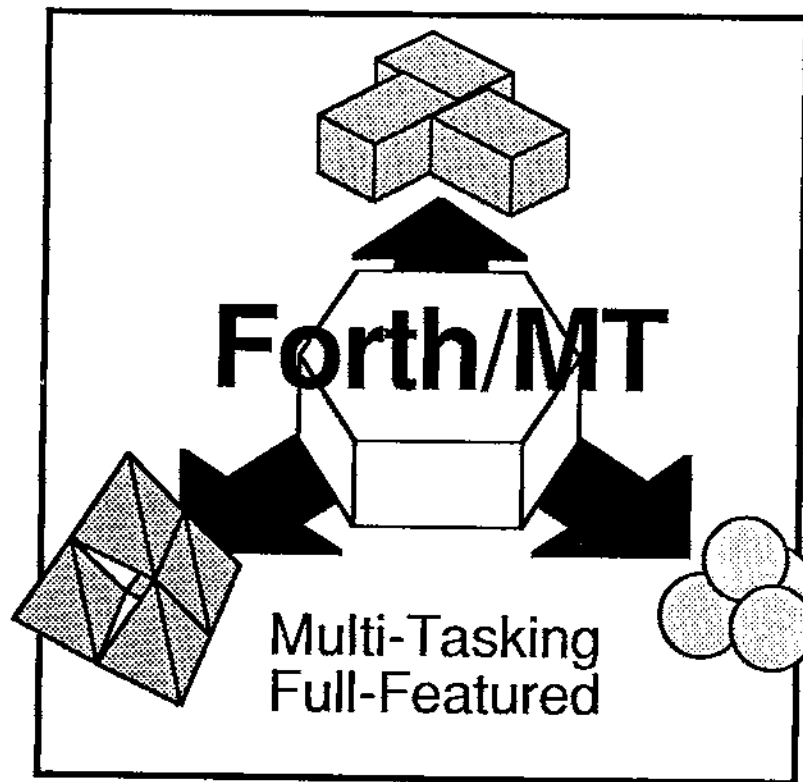
**TextDesigner** is an ideal package for page layout on the ST allowing you to interlace graphics with text. **TextDesigner** reduces the steps and time involved in preparing professional documents and newsletters. Use the tools provided for borders, lines, text, etc. or cut and paste from **PaintPro**, **Doodle**, **Degas**, **TextPro** and other ASCII word processors. **TextDesigner** is a simple to use and versatile package which will give you the professional results you deserve at a reasonable cost.

\$49.95



**PaintPro** is a friendly, yet powerful design and painting package for drawing graphic and artistic pictures. This GEM-based package supports up to three active windows and has a complete toolkit of functions: free-form sketching, lines, circles, ellipses, boxes, fill, copy, move, spray, zoom, undo, help and extensive text capabilities. You can also import "foreign" pictures for enhancement using **Paintpro's** double-sized picture format and send hardcopy to most popular dot-matrix printers. **PaintPro** works with either monochrome or color systems.

\$49.95



**Forth/MT**, the multi-tasking, full-featured Forth language for the ST, is for serious programmers. **Forth/MT** is a complete, 32-bit implementation based on Forth '83 standard. Includes many development aids: full screen editor, monitor, macro assembler, over 1500 word library. Includes TOS, LINEA, floating-point and complex arithmetic commands. \$49.95

# Top shelf books

## from Abacus



**PRESENTING THE ST**  
Gives you an in-depth look at this sensational new computer. Discusses the architecture of the ST, working with GEM, the mouse, operating system, all the various interfaces, the 68000 chip and its instructions, LOGO. 200pp \$16.95

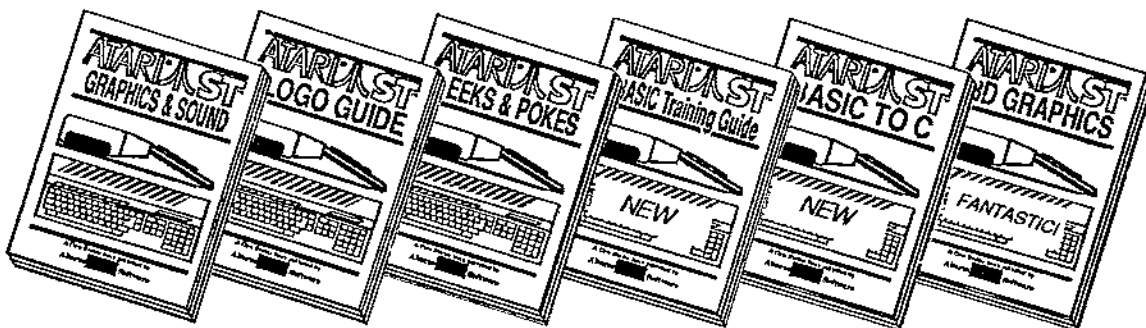
**ST Beginner's Guide**  
Written for the firsthand ST user. Get a basic understanding of your ST. Explore LOGO and BASIC from the ground up. Simple explanations of the hardware and internal workings of the ST. Illustrations, diagrams, Glossary, Index. 200pp \$14.95

**ST INTERNALS**  
Essential guide to the inside information of the ST. Detailed descriptions of sound and graphics chips, internal hardware, I/O ports, using GEM. Commented BIOS listing. An indispensable reference for your ST library. 450pp \$19.95

**GEM Programmer's Ref.**  
For serious programmers needing detailed information on GEM. Presented in an easy-to-understand format. All examples are in G and assembly language. Covers VDI and AES functions. No serious programmer should be without. 410pp \$19.95

**MACHINE LANGUAGE**  
Program in the fastest language for your ATARI ST. Learn 68000 assembly language, its numbering system, use of registers, structure & important details of instruction set, and use of internal system routines. Gained for the ST. 280pp \$19.95

**ST TRICKS & TIPS**  
Fantastic collection of programs and info for the ST. Complete programs include: super-fast RAM disk; time-saving printer spooler; color print hardcopy; plotter output hardcopy; creating accessories. Money saving tricks and tips. 260pp \$19.95



**ST GRAPHICS & SOUND**  
Detailed guide to graphics and sound on the ST. 2D & 3D function plotters, Molté patterns, graphic memory and various resolutions, recursion-Hilbert & Sierpinski fractals, recursion, waveform generation. Examples written in C, LOGO, BASIC and Module2. 250pp \$19.95

**ST LOGO GUIDE**  
Take control of your ST by learning ST LOGO—the easy to use, powerful language. Topics include: file handling, recursion-Hilbert & Sierpinski plots. Data structure, error in C, LOGO, BASIC and handling. Helpful guide for Module2. 250pp \$19.95

**ST TEEKS & POKES**  
Enhance your programs with the examples found within this book. Explores using different languages BASIC, C, LOGO and machine language, using various interfaces, memory usage, reading and saving from and to disk, more. 280pp \$18.95

**BASIC Training Guide**  
Thorough guide for learning ST BASIC programming. Detailed programming fundamentals, commands descriptions, ST graphics & sound, using GEM in BASIC, file management, disk operation. Tutorial programs give hands on experience. 300pp \$18.95

**BASIC to C**  
Move up from BASIC to C. If you're already a BASIC programmer, you can learn C all that much faster. Parallel examples demonstrate the programming techniques and constructs in both languages. Variables, pointers, arrays, data structure. 250pp \$19.95

**3D GRAPHICS**  
**FANTASTIC!** Rotate, zoom, and shade 3D objects. All programs written in machine language for high speed. Learn the mathematics behind 3D graphics. Hidden line removal, shading. With 3D pattern maker and animator. \$24.95

The ATARI logo and ATARI ST are trademarks of Atari Corp.

# Abacus Software

P.O. Box 7219 Dept. A9 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

Optional diskettes are available for all book titles at \$14.95

Call now for the name of your nearest dealer. Or order directly from ABACUS with your MasterCard, VISA, or Amex card. Add \$4.00 per order for postage and handling. Foreign add \$10.00 per book. Other software and books coming soon. Call or write for your free catalog. Dealer inquiries welcome—over 1400 dealers nationwide.



---

# ATARI® ST™

# INTERNALS

---

This INTERNALS volume is a welcome addition to any ST programmer's library. Inside you'll find important hardware and programming information for your ST. Contains valuable information for the professional programmer and ST novice. Here is a short list of some of the things you can expect to read about:

- 68000 processor
- WD 1772 disk controller
- ACIA's 6850
- Centronics interface
- MIDI-interface
- GEMDOS
- Interrupt instructions
- BIOS listing
- Custom chips
- MFP 68901
- YM-2149 sound generator
- RS-232
- DMA controller
- BIOS & XBIOS
- Error codes
- Blitter chip

#### About the authors:

The authors, Klaus Gerits, Lothar Englisch and Rolf Bruckmann, are all part of the experienced Data Becker Product Development team, based in Duesseldorf, W. Germany. They are all best selling computer book authors and very knowledgeable concerning the subjects presented in this book.

ISBN 0-916439-46-1

---

A Data Becker book published by

You Can Count On  **Abacus Software**