# Table of Contents

1  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  

2  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
List of Figures

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

\begin{itemize}
  \item BIT 0 = Carry flag
  \item BIT 1 = Overflow flag
  \item BIT 2 = Zero flag
  \item BIT 3 = Negative flag
  \item BIT 4 = eXtend flag
\end{itemize}
Figure 1.1-1  68000 Registers

31 16 15 8 7 0

D 0
D 1
D 2
D 3
D 4
D 5
D 6
D 7

DATA REGISTERS

31 0

A 0
A 1
A 2
A 3
A 4
A 5
A 6

ADDRESS REGISTERS

31 0

System Stack Pointer SSP
User Stack Pointer USP

STACK POINTER

31 24 23 0

PC

PROGRAM COUNTER

15 8 7 0

Sys Byte User Byte

SR

STATUS REGISTER
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 12 of the interrupt mask is set, while 10 and 11 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 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 IPLO to IPL2; these three lines give you eight possible combinations. The combination determines the priority of the interrupt. That is, if IPLO, 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 IPLO-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 FCO 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**

<table>
<thead>
<tr>
<th>FC2</th>
<th>FC1</th>
<th>FCO</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>unused</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>User-mode data access</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>User-mode program</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>unused</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>unused</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Supervisor data access</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Supervisor program</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Interrupt verification</td>
</tr>
</tbody>
</table>

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

![Diagram of GLUE chip with pin labels and connections.]

- **BGI*: 27
- **RDY*: 28
- **VPA*: 29
- **BEER*: 30
- **DTACK*: 31
- **IPL 1*: 32
- **IPL 2*: 33
- **8MHZ in*: 34
- **GND*: 35
- **BLANK*: 36
- **HSYNC*: 37
- **VSYNC*: 38
- **DE*: 39
- **BR*: 40
- **BGACK*: 41
- **6850CS*: 42
- **500HZ out*: 43
- **9*: A21
- **8*: A20
- **7*: A19
- **6*: A18
- **5*: A17
- **4*: A16
- **3*: A15
- **2*: A14
- **1*: Vcc
- **68*: A13
- **67*: A12
- **66*: A11
- **65*: A10
- **64*: A9
- **63*: A8
- **62*: A7
- **61*: A6
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

![Diagram of MMU]

- GND* 27
- CMPCS 28
- DCYC* 29
- RDAT* 30
- DEV* 31 A5 32
- RAM* 33
- R/W* 34
- A15 35
- A14 36
- A13 37
- A12 38
- A11 39
- A10 40 A9 41 A8 42 A7 43
- VSSCB VCCB A6 A4 A3 A2 A1 VSYNC DACK M0 M1 M2 M3 M4 M5 M6 M7 M8 M9
- 9 LATCH 8 RAS0 7 CAS0LOW 6 CAS0HIGH 5 16MHZ IN 4 D7 3 D6 2 D5 1 D4 68 D3 67 D2 66 D1 65 D0 64 MAD 9 63 MAD 8 62 MAD 7 61 GND
Figure 1.2-3 SHIFTER

XTL 0

XTL 1 32MHZ in

VCC

16MHZ out

CS*

DE

A 1

A 2

A 3

A 4

A 5

R/W*

LOAD*

MONO

R 0

R 1

R 2

G 0

G 1

G 2

B 0

B 1

B 2

GND
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

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS*</td>
<td>INTR</td>
</tr>
<tr>
<td>R/W*</td>
<td>DRQ</td>
</tr>
<tr>
<td>A0</td>
<td>DD*</td>
</tr>
<tr>
<td>A1</td>
<td>WP*</td>
</tr>
<tr>
<td>DAL 0</td>
<td>INDEX</td>
</tr>
<tr>
<td>DAL 1</td>
<td>TRK0</td>
</tr>
<tr>
<td>DAL 2</td>
<td>WD</td>
</tr>
<tr>
<td>DAL 3</td>
<td>WG</td>
</tr>
<tr>
<td>DAL 4</td>
<td>MO</td>
</tr>
<tr>
<td>DAL 5</td>
<td>RD*</td>
</tr>
<tr>
<td>DAL 6</td>
<td>CLK</td>
</tr>
<tr>
<td>DAL 7</td>
<td>DIRC</td>
</tr>
<tr>
<td>MR*</td>
<td>STEP</td>
</tr>
<tr>
<td>GND</td>
<td>Vcc</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>R/W=1</th>
<th>R/W=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Status Reg.</td>
<td>Command Reg.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Track Reg.</td>
<td>Track Reg.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Sector Reg.</td>
<td>Sector Reg.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Data Reg.</td>
<td>Data Reg.</td>
</tr>
</tbody>
</table>

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.

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

**Type 1 Commands**

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restore</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>H</td>
<td>V</td>
<td>R1 R0</td>
</tr>
<tr>
<td>Seek</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>H</td>
<td>V</td>
<td>R1 R0</td>
</tr>
<tr>
<td>Step</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>U</td>
<td>H</td>
<td>V</td>
<td>R1 R0</td>
<td></td>
</tr>
<tr>
<td>Step In</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>U</td>
<td>H</td>
<td>V</td>
<td>R1 R0</td>
<td></td>
</tr>
<tr>
<td>Step Out</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>U</td>
<td>H</td>
<td>V</td>
<td>R1 R0</td>
<td></td>
</tr>
</tbody>
</table>
All five commands have several variable bits; bits R0 and R1 give the time between two step impulses. The possible combinations are:

<table>
<thead>
<tr>
<th>R1</th>
<th>R0</th>
<th>STEP RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2 milliseconds</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3 milliseconds</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>5 milliseconds</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>6 milliseconds</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>BIT</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Sector</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>M</td>
<td>H</td>
<td>E</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Write Sector</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>M</td>
<td>H</td>
<td>E</td>
<td>P</td>
<td>A0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>BIT</th>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Address</td>
<td>1 1 0 0 H E 0 0</td>
</tr>
<tr>
<td>Read Track</td>
<td>1 1 1 0 H E 0 0</td>
</tr>
<tr>
<td>Write Track</td>
<td>1 1 1 1 H E P 0</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>BIT</th>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Force Interrupt</td>
<td>1 1 0 1 1 3 1 2</td>
</tr>
</tbody>
</table>

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.

**TAI,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:

<table>
<thead>
<tr>
<th>IERA</th>
<th>I/O port bit 7 (highest priority)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 7: I/O port bit 7 (highest priority)</td>
<td></td>
</tr>
<tr>
<td>Bit 6: I/O port bit 6</td>
<td></td>
</tr>
<tr>
<td>Bit 5: Timer A</td>
<td></td>
</tr>
<tr>
<td>Bit 4: Receive buffer full</td>
<td></td>
</tr>
<tr>
<td>Bit 3: Receive error</td>
<td></td>
</tr>
<tr>
<td>Bit 2: Sender buffer empty</td>
<td></td>
</tr>
<tr>
<td>Bit 1: Sender error</td>
<td></td>
</tr>
<tr>
<td>Bit 0: Timer B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IERB</th>
<th>I/O port bit 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 7: I/O port bit 5</td>
<td></td>
</tr>
<tr>
<td>Bit 6: I/O port bit 4</td>
<td></td>
</tr>
<tr>
<td>Bit 5: Timer C</td>
<td></td>
</tr>
<tr>
<td>Bit 4: Timer D</td>
<td></td>
</tr>
<tr>
<td>Bit 3: I/O port bit 3</td>
<td></td>
</tr>
<tr>
<td>Bit 2: I/O port bit 2</td>
<td></td>
</tr>
<tr>
<td>Bit 1: I/O port bit 1</td>
<td></td>
</tr>
<tr>
<td>Bit 0: I/O port bit 0, lowest priority</td>
<td></td>
</tr>
</tbody>
</table>

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.
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 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
\[ 0 = \text{Frequency from TC and RC} \]
\[ \text{directly used as transfer frequency (used only for synchronous transfer)} \]
\[ 1 = \text{Frequency in TC and RC internally divided by 16.} \]

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.
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

VSS 1
RX DATA
RX CLK
TX CLK
RTS*
TX DATA
IRQ*
CS 0
CS 2*
CS 1
RS
Vcc

CTS* 2
DCD*1
D 0
D 1
D 2
D 3
D 4
D 5
D 6
D 7

E

R/W*
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:

<table>
<thead>
<tr>
<th>R/W</th>
<th>RS</th>
<th>Register</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Control Register</td>
<td>write</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Sender Register</td>
<td>write</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Status Register</td>
<td>read</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Receive Register</td>
<td>read</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>CR1</th>
<th>CR0</th>
<th>RXCLK/TXCLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>RXCLK/TXCLK</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>RXCLK/TXCLK by 16 (for MIDI)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>RXCLK/TXCLK by 64 (for keyboard)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Master RESET</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>CR4</th>
<th>CR3</th>
<th>CR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>7 databits, 2 stopbits, even parity</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>7 databits, 2 stopbits, odd parity</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>7 databits, 1 stopbit, even parity</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>7 databits, 1 stopbit, odd parity</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8 databits, 2 stopbit, no parity</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8 databits, 1 stopbit, no parity</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8 databits, 1 stopbit, even parity</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8 databits, 1 stopbit, odd parity</td>
</tr>
</tbody>
</table>

**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.

45
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.

SR7  
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 $FFFFC00 and $FFFFC02. 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

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

YM-2149

49
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.

<table>
<thead>
<tr>
<th>BDIR</th>
<th>BC2</th>
<th>BC1</th>
<th>PSG Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Inactive</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Latch address</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Inactive</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Read from PSG</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Latch address</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Inactive</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Write to PSG</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Latch address</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>BDIR</th>
<th>BC1</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Inactive, PSG data bus high</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Read PSG registers</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Write PSG registers</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Latch, write register number(s)</td>
</tr>
</tbody>
</table>

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 register 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

<table>
<thead>
<tr>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
<th>CONT</th>
<th>ATAC</th>
<th>ALT</th>
<th>HOLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

53
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:
Abacus Software
Atari ST Internals

<table>
<thead>
<tr>
<th>$FF8001</th>
<th>Bit</th>
<th>Memory configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-0</td>
<td></td>
<td>0000 128K 128K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001 128K 512K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0010 128K 2M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0011 reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100 512K 128K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0101 512K 512K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100 512K 2M, normally reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100 reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000 2M 128K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1001 2M 512K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1010 2M 2M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1011 reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11XX reserved</td>
</tr>
</tbody>
</table>

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.
Abacus Software

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
Abacus Software

Atari ST Internals

$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**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFFC00</td>
<td>2 ACIA's 6580</td>
</tr>
<tr>
<td>$FFFA00</td>
<td>MFP 68901</td>
</tr>
<tr>
<td>$FF8800</td>
<td>SOUND AY-3-8910</td>
</tr>
<tr>
<td>$FF8600</td>
<td>DMA / WD 1770</td>
</tr>
<tr>
<td>$FF8400</td>
<td>RESERVED</td>
</tr>
<tr>
<td>$FF8200</td>
<td>VIDEO CONTROLLER</td>
</tr>
<tr>
<td>$FF8000</td>
<td>DATA CONFIGURATION</td>
</tr>
</tbody>
</table>
**Figure 1.7-2 Memory Map of the ATARI ST**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FF$ FCO0</td>
<td>I/O - Area</td>
<td>16776192</td>
</tr>
<tr>
<td>$FF$ FA00</td>
<td></td>
<td>16775680</td>
</tr>
<tr>
<td>$FF$ 8800</td>
<td></td>
<td>16746496</td>
</tr>
<tr>
<td>$FF$ 8600</td>
<td>I/O - Area</td>
<td>16745984</td>
</tr>
<tr>
<td>$FF$ 8400</td>
<td></td>
<td>16745472</td>
</tr>
<tr>
<td>$FF$ 8200</td>
<td></td>
<td>16744960</td>
</tr>
<tr>
<td>$FF$ 8000</td>
<td></td>
<td>16744448</td>
</tr>
<tr>
<td>$FE$ FFFF</td>
<td>192 K System ROM</td>
<td>16711679</td>
</tr>
<tr>
<td>$FC$ 0000</td>
<td>128 K ROM Expansion Cartridge</td>
<td>16515072</td>
</tr>
<tr>
<td>$FA$ 0000</td>
<td></td>
<td>16384000</td>
</tr>
<tr>
<td>$07$ FFFF</td>
<td>512 K RAM</td>
<td>524287</td>
</tr>
<tr>
<td>$00$ 0000</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
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 processors 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.

<table>
<thead>
<tr>
<th>Port 0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 XB/UP</td>
<td>(K12)</td>
</tr>
<tr>
<td>2 XA/DOWN</td>
<td>(K10)</td>
</tr>
<tr>
<td>3 YA/LEFT</td>
<td>(K9)</td>
</tr>
<tr>
<td>4 YB/RIGHT</td>
<td>(K8)</td>
</tr>
<tr>
<td>6 LEFT BUTTON/FIRE</td>
<td>(K11)</td>
</tr>
<tr>
<td>7 +5V</td>
<td>(K13)</td>
</tr>
<tr>
<td>8 GND</td>
<td>(K1)</td>
</tr>
<tr>
<td>9 RIGHT BUTTON</td>
<td>(K6)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 UP</td>
<td>(K7)</td>
</tr>
<tr>
<td>2 DOWN</td>
<td>(K5)</td>
</tr>
<tr>
<td>3 LEFT</td>
<td>(K4)</td>
</tr>
<tr>
<td>4 RIGHT</td>
<td>(K3)</td>
</tr>
<tr>
<td>5 Port 0 enable</td>
<td>(K17)</td>
</tr>
<tr>
<td>6 FIRE</td>
<td>(K6)</td>
</tr>
<tr>
<td>7 +5V</td>
<td>(K13)</td>
</tr>
<tr>
<td>8 GND</td>
<td>(K1)</td>
</tr>
</tbody>
</table>
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:
Abacus Software

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:
Abacus Software

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.
$17$
Joystick duration message. One parameter is required.

1 byte Time between two messages in 1/100 sec.

From this point on, packages of the following form are sent continuously (as long as no other mode is selected):

1 byte Bit 0 for the button on joystick 1, bit 1 for that of joystick 0
1 byte Bits 0-3 for the position of joystick 1, bits 4-7 for the position of joystick 0

NOTE: The read interval should not be shorter than the transfer channel needs to send the two bytes of the package.

$18$
Fire button duration message. The condition of the button in joystick 1 (!) is continually tested and the result packed into a byte. This means that a message byte contains 8 such tests, whereby bit 7 is the most recent. The keyboard controller determines the time between byte fetches by the main processor. This time is divided into eight equal intervals in which the button is polled. The polling then takes place as regularly as possible. This mode remains active until another command is received.

$19$
Cursor key simulation mode for joystick 0 (!). The current position of the joystick is sent to the main processor as if the corresponding cursor keys had been pressed (as often as necessary). To avoid having to explain the same things for the following parameters, here are the most important: All times are assumed to be in tenths of seconds. R indicates the time, when reached, cursor clicks will be sent in intervals of T. After this the interval is V. If R=0, only V is responsible for the interval. Naturally, this mechanism comes into play only when the joystick is held in the same position for longer than T or R.

1 byte RX
1 byte RY
1 byte TX
1 byte TY
1 byte VX
1 byte VY

78
$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.

```assembly
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
1 00000000 3F3C0022
2 00000004 4B9E
3 00000006 549F
4 00000008 41F900000000
5 0000000E 43F9000000D6
6 00000014 23C0000000104
7 0000001A 23F00000C00000100
8 00000022 2189000C

start:
mov.w #kbvec,-(a7)
trap #xbios
addq.l #2,a7
lea 0,a0
lea keyin,al
move.l d0,savea
move.l stvec(a0,d0),save
move.l al,stvec(a0,d0)
```
move . w #$f000, d4
loop:
move . w d4, tbuf+1
bsr keyout
wait:
cmpl.b #0, rbuf
beq wait
move . w #6, d6
bsr bytout
addq.w #6, d4
bsr hexout
cmpi.w #$ffff, d4
bit loop
bca exit
bufout:
lea rbuf+1, a4
bytout:
move . b (a4)+, d0
bsr hexout
subq.b #1, d6
move . w #6, d4
bufout
move . w #6, d4
loop
exit:
movea . w d0, al
lsr.b #4, d0
andl.l #15, d0
lea table, a3
move . b 0(a3, d0), d2
move . w a1, d0
move . w a1, d0
loop
move . w #8, d0
bsr chrout
move . w (a7)+, d0
move . w #" ", d0
chrout:
move . w d0, -(a7)
move . w #prt, -(a7)
move . w #chout, -(a7)
trap #bios
addq.l #6, a7
rts
exit:
movea savea, a0
Atari ST Internals

67 000000A6 203900000100  move.l save,d0  
68 000000AC 2140000C  move.l d0,stvec(a0)  
69 000000B0 3F3C0000  move.w #torm,-(a7)  
70 000000B1 4E41  trap #gmdos  
71 
72 000000B6 13FC0000000000F8  move.b #0,rbuf  
73 000000BE 487900000109  psa 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  
80 000000D6 103C0008  move.b #8,d0  
81 000000DA 43F9000000FB  lea rbuf,al  
82  
83 000000E0 12D8  move.b (a0)+,(al)+  
84 000000E2 5300  subq.b #1,d0  
85 000000E4 66FA  bne repin  
86 000000E6 4E75  rts  
87  
88 000000EB 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 ds.b rdm  
94 0000010A  ds.b 2  
95 0000010C  .end  

82
$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 $OC 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 17 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

R/-W
-SNDCS
A1
2MHz
-RESET
D8-15

SOUND

10/TA1
GPO
DRIVE0
DRIVE1
SIDE0
RTS
DTR

AUDIO IN

AUDIO OUT
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

SO
I/OA4
I/OA3
SI
I6
I2
I1

2
20
4
3
22
5
8

1
13
14
25
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 though 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

+5V

1

4

5

14

95

ACIA 6850

6850CS

A2

A1

R/-W

D8-15

500KHz

5

4

3

2

1

4

5

1
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

| +5V | 1 | 21 | A8 |
| 5V | 2 | 22 | A14 |
| D14 | 3 | 23 | A7 |
| D15 | 4 | 24 | A9 |
| D12 | 5 | 25 | A6 |
| D13 | 6 | 26 | A10 |
| D10 | 7 | 27 | A5 |
| D11 | 8 | 28 | A12 |
| D8 | 9 | 29 | A11 |
| D9 | 10 | 30 | A4 |
| D6 | 11 | 31 | -ROM3 |
| D7 | 12 | 32 | A3 |
| D4 | 13 | 33 | -ROM4 |
| D5 | 14 | 34 | A2 |
| D2 | 15 | 35 | -DOS |
| D3 | 16 | 36 | A1 |
| D0 | 17 | 37 | -LDS |
| D1 | 18 | 38 | GND |
| A13 | 19 | 39 | GND |
| A16 | 20 | 40 | GND |

Position:

```
1
21
```

20

40

96
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
2. SIDE 0 SELECT
3. GND
4. INDEX
5. DRIVE 0 SELECT
6. DRIVE 1 SELECT
7. GND
8. MOTOR ON
9. DIRECTION IN
10. STEP
11. WRITE DATA
12. WRITE GATE
13. TRACK 00
14. WRITE PROTECT
Figure 2.7-1 Disk Connection

CD0-7 14
CA2 13
CA1 12
CR/-W 11
-FDSCS 10
-RESET 9
8MHz 8
I/OA2 7
I/OA1 6
I/OA0 5

FDC

FDRQ
INTR
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**

![DMA Port Diagram]

**Figure 2.8-2 DMA Connections**

![DMA Connections Diagram]
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
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 ($FFFFFFFE) 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.
$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',$OD,$OA,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:
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, DO 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, DO contains the value zero or $FFFF. A zero in DO 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

ADD.L $6,SP        * The supervisor mode is active
MOVE.L DO,_SAVE_SSP * after the TRAP

MOVE.L _SAVE_SSP,-(SP) * Old supervisor stack pointer
MOVE.W #$20,-(SP)   * Call SUPER
TRAP #1            * Old supervisor stack pointer
ADD.L #6,SP        * Now we are back in the user mode

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 #%1011011011001,-(SP)  * Set date to 10/25/1985
MOVE.W i$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.

\[
\begin{align*}
\text{MOVE.W} & \ #0,-(\text{SP}) \quad \text{* Error code no error, else 1} \\
\text{MOVE.L} & \ #1000,-(\text{SP}) \quad \text{* Protect $1000$ bytes at program start} \\
\text{MOVE.W} & \ #31,-(\text{SP}) \quad \text{* Function number, end program} \\
\text{TRAP} & \ #1 \quad \text{* now}
\end{align*}
\]

$\text{36 GET DISK FREE SPACE}$

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 $\text{36}$ 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/csorsts.s/cqsort.s

Translated this would mean: load the file cqsort.s from the subdirectory csorsts.s. This subdirectory csorsts.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 DO           * 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 $28 = 40$ 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 DO         * Error occurred?
BMI   error      * Apparently
MOVE DO,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 $400$. 

123
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, DO 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 DO * 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:
.buffer 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 #$-l,-(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               * Error or address of memory?
TST.W D0                   * Negative long word = error!
BMI    error                * Else start addr of the reserved area
MOVE.L D0,mstart

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.
$4A$ 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 $C(A5),D0  * $C(A5) contains length of the prg area
ADD.L $14(A5),D0  * $14(A5) contains 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.
Abacus Software

Atari ST Internals

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, DO contains the value zero if a corresponding file has been found. In this case the 44-byte DTA buffer is constructed as follows:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>0-20</th>
<th>Reserved for GEMDOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>21</td>
<td>File attribute</td>
</tr>
<tr>
<td>Bytes</td>
<td>22-23</td>
<td>Clock time of file creation</td>
</tr>
<tr>
<td>Bytes</td>
<td>24-25</td>
<td>Date of file creation</td>
</tr>
<tr>
<td>Bytes</td>
<td>26-29</td>
<td>File size in bytes (long)</td>
</tr>
<tr>
<td>Bytes</td>
<td>30-43</td>
<td>Name and extension of the file</td>
</tr>
</tbody>
</table>

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

```assembly
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 check 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 pointer 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)  
MOVE.L #oldname,-(SP)  
MOVE.W #0,-(SP)  
MOVE.W #$56,-(SP)  
TRAP #1  
ADD.L #12,SP  
TST.L DO  

oldnam:  
   .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

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

Assembler:
```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:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No characters ready</td>
</tr>
<tr>
<td>-1</td>
<td>(at least) one character ready</td>
</tr>
</tbody>
</table>

The parameter `dev` specifies the input device:

<table>
<thead>
<tr>
<th>dev</th>
<th>Input device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PRT:, Centronics interface</td>
</tr>
<tr>
<td>1</td>
<td>AUX:, RS-232 interface</td>
</tr>
<tr>
<td>2</td>
<td>CON:, Keyboard and screen</td>
</tr>
<tr>
<td>3</td>
<td>MIDI, MIDI interface</td>
</tr>
<tr>
<td>4</td>
<td>IKBD, Keyboard port</td>
</tr>
</tbody>
</table>

The following table lists the allowed accesses to these devices:

<table>
<thead>
<tr>
<th>Operation</th>
<th>PRT:</th>
<th>AUX:</th>
<th>CON:</th>
<th>MIDI</th>
<th>IKBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input status</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Input</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Output status</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Output</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

```assembly
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
```

142
2 conin  

read character from device

C:

```c
long conin(dev)
int dev;
```

Assembler:

```assembly
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:

```assembly
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

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


6 tickcal

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:

```c
long getbpb(dev)
    int dev;
```

Assembler:

```assembly
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:

```assembly
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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>80 track SS</th>
<th>80 track DS</th>
</tr>
</thead>
<tbody>
<tr>
<td>recsiz</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>clsiz</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>clsizb</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>rdlen</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>fsiz</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>fatrec</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>datrec</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>numcl</td>
<td>351</td>
<td>711</td>
</tr>
</tbody>
</table>
8 bcostat

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**  \hspace{1cm} *inquire drive status*

C: `long drvmap()`

Assembler:

```
move.w #10, -(sp)
trap #13
addq.1 #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)   \hspace{1cm} drvmap
trap #13
addq.1 #2, sp
```
11 kbshift

inquire/change keyboard status

C:

```c
long kbshift(mode)
int mode;
```

Assembler:

```asm
move.w mode,-(sp)
move.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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Right shift key</td>
</tr>
<tr>
<td>1</td>
<td>Left shift key</td>
</tr>
<tr>
<td>2</td>
<td>Control key</td>
</tr>
<tr>
<td>3</td>
<td>ALT key</td>
</tr>
<tr>
<td>4</td>
<td>Caps Lock on</td>
</tr>
<tr>
<td>5</td>
<td>Right mouse button (CLR/HOME)</td>
</tr>
<tr>
<td>6</td>
<td>Left mouse button (INSERT)</td>
</tr>
<tr>
<td>7</td>
<td>Unused</td>
</tr>
</tbody>
</table>

Example:

```asm
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:

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable mouse</td>
</tr>
<tr>
<td>1</td>
<td>Enable mouse, relative mode</td>
</tr>
<tr>
<td>2</td>
<td>Enable mouse, absolute mode</td>
</tr>
<tr>
<td>3</td>
<td>unused</td>
</tr>
<tr>
<td>4</td>
<td>Enable mouse, keyboard mode</td>
</tr>
</tbody>
</table>

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
```
**ssbrk**

save memory space

C:

```c
long ssbrk(number)
int number;
```

Assembler:

```assembly
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:

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

C: long physbase()

Assembler:

    move  #2,-(sp)
    trap  #14
    addq.1 #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

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

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

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

163
7 setcolor

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

Assembler:

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

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:**

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

**Assembler:**

```assembly
move.w count,-(sp)
mov.e side,-(sp)
mov.e track,-(sp)
mov.e sector,-(sp)
mov.e dev,-(sp)
clr.l -(sp)
mov.e buffer,-(sp)
mov.e #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)     ; floprd
trap #14
add.l #20,sp
tst d0
bmi error
...
buffer ds.b 512     ; Buffer for a sector
```
9 flopwr

write diskette sector

C:

```c
int flopwr(buffer, filler, dev, sector, track, side, count)
    long buffer, filler;
    int dev, sector, track, side, count;
```

Assembler:

```asm
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:

```asm
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
bmi error
```

```
...  Did an error occur?
```

```
buffer ds.b 3*512        Buffer for three sectors
```

167
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) interleaved
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) flopfmt
move.w #10,-(sp)
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-l, -(sp) Length
    move.w #12, -(sp) midiws
    trap #14
    addq.l #8, sp

....

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

**initialize MFP format**

C: void mfpint(number, vector)

```c
int number;
long vector;
```

Assembler:

```assembly
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:

```assembly
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:
```

---

171
14 iorec  

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:

<table>
<thead>
<tr>
<th>dev</th>
<th>Input device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RS-232</td>
</tr>
<tr>
<td>1</td>
<td>Keyboard</td>
</tr>
<tr>
<td>2</td>
<td>MIDI</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>ibuf  Pointer to an input buffer</td>
</tr>
<tr>
<td>int</td>
<td>ibufsize Size of the input buffer</td>
</tr>
<tr>
<td>int</td>
<td>ibufhd  Head index</td>
</tr>
<tr>
<td>int</td>
<td>ibuf1   Tail index</td>
</tr>
<tr>
<td>int</td>
<td>ibuflow Low water mark</td>
</tr>
<tr>
<td>int</td>
<td>ibufhi  High water mark</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>RS-232 input</th>
<th>RS-232 output</th>
<th>Keyboard</th>
<th>MIDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>$9D0</td>
<td>($9DE)</td>
<td>$942</td>
</tr>
<tr>
<td>Buffer address</td>
<td>$6D0</td>
<td>$7D0</td>
<td>$8D0</td>
</tr>
<tr>
<td>Buffer length</td>
<td>$100</td>
<td>$100</td>
<td>$80</td>
</tr>
<tr>
<td>Head index</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tail index</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Low water mark</td>
<td>$40</td>
<td>$40</td>
<td>$20</td>
</tr>
<tr>
<td>High water mark</td>
<td>$C0</td>
<td>$C0</td>
<td>$20</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>baud</th>
<th>Baud rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>1</td>
<td>9600</td>
</tr>
<tr>
<td>2</td>
<td>4800</td>
</tr>
<tr>
<td>3</td>
<td>3600</td>
</tr>
<tr>
<td>4</td>
<td>2400</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
</tr>
<tr>
<td>6</td>
<td>1800</td>
</tr>
<tr>
<td>7</td>
<td>1200</td>
</tr>
<tr>
<td>8</td>
<td>600</td>
</tr>
<tr>
<td>9</td>
<td>300</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
</tr>
<tr>
<td>11</td>
<td>150</td>
</tr>
<tr>
<td>12</td>
<td>134</td>
</tr>
<tr>
<td>13</td>
<td>110</td>
</tr>
<tr>
<td>14</td>
<td>75</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
</tr>
</tbody>
</table>

Example:

```
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: ...

176
17 random

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 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:

<table>
<thead>
<tr>
<th>Address</th>
<th>40 track SS</th>
<th>40 track DS</th>
<th>80 track SS</th>
<th>80 track DS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-7</td>
<td>&quot;Loader&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-10</td>
<td>24-bit serial number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11-12</td>
<td>BPS 512</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>13</td>
<td>SPC 1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>14-15</td>
<td>RES 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>FAT 2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>17-18</td>
<td>DIR 64</td>
<td>112</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>19-20</td>
<td>SEC 360</td>
<td>720</td>
<td>720</td>
<td>1440</td>
</tr>
<tr>
<td>21</td>
<td>MEDIA 252</td>
<td>253</td>
<td>248</td>
<td>249</td>
</tr>
<tr>
<td>22-23</td>
<td>SPF 2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>24-25</td>
<td>SPT 9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>26-27</td>
<td>SIDE 1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>28-29</td>
<td>HID 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>510-511</td>
<td>CHECKSUM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The abbreviations have the following meanings:

- **BPS:** 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:

```assembly
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)       ; protobt
move.w #18,-(sp)
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
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 `verify ($444)` is set, the sectors will automatically be verified after they are written.

Example:

```assembly
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.w -(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.1 #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:

<table>
<thead>
<tr>
<th>function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable cursor</td>
</tr>
<tr>
<td>1</td>
<td>Enable cursor</td>
</tr>
<tr>
<td>2</td>
<td>Flash cursor</td>
</tr>
<tr>
<td>3</td>
<td>Steady cursor</td>
</tr>
<tr>
<td>4</td>
<td>Set cursor flash rate</td>
</tr>
<tr>
<td>5</td>
<td>Get cursor flash rate</td>
</tr>
</tbody>
</table>

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.1 #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 %#101100110000010000000000000000,-(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 gettime

C: long gettime()

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)  gettime
trap  #14
addq.l #2,sp
move.l d0,time  Save time and date
24 bioskeys  \textit{restore keyboard table}

C: \texttt{void bioskeys()}

Assembler:

\begin{verbatim}
move.w \#24,-(sp)
trap \#14
addq.l \#2,sp
\end{verbatim}

If you have selected a new keyboard layout with the XBIOS function 16, \textit{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:

\begin{verbatim}
move.w \#24,-(sp) bioskeys
trap \#14
addq.l \#2,sp
\end{verbatim}
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)

```c
int number;
```

Assembler:

```assembly
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:

```assembly
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.1 #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.1 #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.1 #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:

```assembly
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:

<table>
<thead>
<tr>
<th>Bit number</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>matrix printer</td>
<td>daisy-wheel</td>
</tr>
<tr>
<td>1</td>
<td>monochrome printer</td>
<td>color printer</td>
</tr>
<tr>
<td>2</td>
<td>Atari printer</td>
<td>Epson printer</td>
</tr>
<tr>
<td>3</td>
<td>Test mode</td>
<td>Quality mode</td>
</tr>
<tr>
<td>4</td>
<td>Centronics port</td>
<td>RS-232 port</td>
</tr>
<tr>
<td>5</td>
<td>Continuous paper</td>
<td>Single-sheet</td>
</tr>
<tr>
<td>6-14</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>always 0</td>
<td></td>
</tr>
</tbody>
</table>

Example:

    move.w #%0000100,-(sp)    Epson printer
    move.w #33,-(sp)    setprt
    trap #14
    addq.l #4,sp

198
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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0E</td>
<td>midivec</td>
<td>$79C6$</td>
</tr>
<tr>
<td>A12</td>
<td>vkbderr</td>
<td>$759C$</td>
</tr>
<tr>
<td>A16</td>
<td>vmiderr</td>
<td>$759C$</td>
</tr>
<tr>
<td>A1A</td>
<td>statvec</td>
<td>$7034$</td>
</tr>
<tr>
<td>A1E</td>
<td>mousevec</td>
<td>$15296$</td>
</tr>
<tr>
<td>A22</td>
<td>clockvec</td>
<td>$6A46$</td>
</tr>
<tr>
<td>A26</td>
<td>joyvec</td>
<td>$7034$</td>
</tr>
<tr>
<td>A2A</td>
<td>MIDI</td>
<td>$7556$</td>
</tr>
<tr>
<td>A2E</td>
<td>keyboard</td>
<td>$7568$</td>
</tr>
</tbody>
</table>
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.1 #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.1 #6, sp
    ...
    parameter dc.1 ...

202
37 wvbl  

wait for video

C: void wvbl();

Assembler:

\begin{verbatim}
move.w #36,-(sp)
trap #14
addq.l #2,sp
\end{verbatim}

This function waits for the next picture return. It can be used to synchronize graphic outputs with the beam return, for example.

Example:

\begin{verbatim}
move.w #36,-(sp)       wait for wvbl
trap #14
addq.l #2,sp
\end{verbatim}
38 supexec  

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
    ...

set supervisor execution
39 puntaes

disable AES

C: void puntaes()

Assembler:

    movw    #39,-(sp)
    trap    #14
    addq.l  #2,sp

The AES can be disabled with this function, provided it is not in ROM.

Example:

    movw    #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)

VIDEO SCREEN

COLOR NUMBER

VIDEO-RAM
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 $x_1,y_1$ and $x_2,y_2$. 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:

- $X_1 =$ $x_1$ coordinate
- $Y_1 =$ $y_1$ coordinate
- $X_2 =$ $x_2$ coordinate
- $Y_2 =$ $y_2$ 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 $x_1,y_1$ to $x_2,y_2$.

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 line 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
_pathptr = Pointer to the line pattern to use
_pathmsk = "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 _pathptr 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 _pathmsk plays an important role in this. The number of 16-bit values at the address in _pathptr is dependent on its value. If, for example, _pathmsk contains the value 5, six 16-bit values should be located at the address in _pathptr 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.

- $X1 = xl$ coordinate, left upper
- $Y1 = yl$ 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)
- $WRT\_MOD = Determines the write mode$
- $jpatptr = Pointer to the fill pattern used$
- $jpatmsk = "Mask" for the fill pattern$
- $CLIP = Clipping flag$
- $XMN\_CLIP = X$ 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_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

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 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.

- \_WRT\_MODE = Write mode
- \_TEXT\_FG = Text foreground color
- \_TEXT\_BG = Text background color
- \_FBASE = Pointer to the start of the font data
- \_FWIDTH = Width of the font
- \_SOURCE\_X = X-coordinate of the char in the font
- \_SOURCE\_Y = Y-coordinate of the char in the font
- \_DEST\_X = X-coordinate of the char on the screen
- \_DEST\_Y = Y-coordinate of the char on the screen
- \_DEL\_X = Width of the character in pixels
- \_DEL\_Y = Height of the character in pixels
- \_STYLE = Bit-wise coded flag for special effects
- \_LITEMASK = Bit pattern used for "lightening"
- \_SKEWMASK = Bit pattern used for skewing
- \_WEIGHT = Factor for character enlargement
- \_R\_OFF = Right offset of the char for skewing
- \_L\_OFF = Left offset of the char for skewing
- \_SCALE = Flag for scaling
- \_XACC\_DDA = Accumulator for scaling
- \_DDA\_INC = Scaling factor
- \_T\_SCL\_STS = Scaling direction flag
- \_CHUP = Character rotation vector
- \_MONO\_STATUS = Flag for monospaced type
- \_scr\_chp = Pointer to buffer for effects
- \_scr\_t2 = Offset scaling buffer in \_scr\_chp

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. Usable 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 _scrtchp 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 function 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.

$\text{AOOC UNDRAW SPRITE}$

This opcode is related to $\text{AOOD, DRAW SPRITE}$. 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 4x64+10=266 bytes large.

$\text{AOOD DRAW SPRITE}$

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**

<table>
<thead>
<tr>
<th>fg</th>
<th>bg</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The background appears</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>The color in word 4 appears</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The color in word 5 appears</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The color in word 5 appears</td>
</tr>
</tbody>
</table>

**XOR Format**

<table>
<thead>
<tr>
<th>fg</th>
<th>bg</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The background appears</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>The color in word 4 appears</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The pixel on the screen is XORed with the fb bit</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The color in word 5 appears</td>
</tr>
</tbody>
</table>

### $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 CONTRL, 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.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Size</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>v_planes</td>
<td>word</td>
<td>Number of planes</td>
</tr>
<tr>
<td>2</td>
<td>v_lin_wr</td>
<td>word</td>
<td>Bytes per scan line</td>
</tr>
<tr>
<td>4</td>
<td>CONTRL</td>
<td>long</td>
<td>Pointer to the CONTRL array</td>
</tr>
<tr>
<td>8</td>
<td>INTIN</td>
<td>long</td>
<td>Pointer to the INTIN array</td>
</tr>
<tr>
<td>12</td>
<td>PTSIN</td>
<td>long</td>
<td>Pointer to the PTSIN array</td>
</tr>
<tr>
<td>16</td>
<td>INTOUT</td>
<td>long</td>
<td>Pointer to the INTOUT array</td>
</tr>
<tr>
<td>20</td>
<td>PTSOUT</td>
<td>long</td>
<td>Pointer to the PTSOUT array</td>
</tr>
<tr>
<td>24</td>
<td>_FG_BP_1</td>
<td>word</td>
<td>Plane 0 color value</td>
</tr>
<tr>
<td>26</td>
<td>_FG_BP_2</td>
<td>word</td>
<td>Plane 1 color value</td>
</tr>
<tr>
<td>28</td>
<td>_FG_BP_2</td>
<td>word</td>
<td>Plane 2 color value</td>
</tr>
<tr>
<td>32</td>
<td>_LSTLIN</td>
<td>word</td>
<td>Should be -1 ($FFFF) (?)</td>
</tr>
<tr>
<td>34</td>
<td>_LN_MASK</td>
<td>word</td>
<td>Line pattern for $A003</td>
</tr>
</tbody>
</table>
| 36     | _WRT_MODE    | word | Write mode (0=write mode  
1=transparent  
2=XOR mode  
3=Inverse trans.)                           |
<p>| 38     | X1           | word | X1-coordinate                                 |
| 40     | Y1           | word | Y1-coordinate                                 |
| 42     | X2           | word | X2-coordinate                                 |
| 44     | Y2           | word | Y2-coordinate                                 |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>Abacus Software</th>
<th></th>
<th>Atari ST Internals</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>_patptr</td>
<td>long</td>
<td>Pointer to the fill pattern (see $A004)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>_patmsk</td>
<td>word</td>
<td>Fill pattern &quot;mask&quot; (see $A004)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 52 | _multifill | word | 0=fill pattern is only for one plane  
1=fill pattern is for multi-plane |   |   |
| 54 | _CLIP | word | 0=no clipping (see $A005) not 0=clipping |   |   |
| 56 | _XMN_CLIP | word | 0=fill pattern is only for one plane  
1=fill pattern is for multi-plane |   |   |
| 58 | _YMN_CLIP | word | 0=fill pattern is only for one plane  
1=fill pattern is for multi-plane |   |   |
| 60 | _XMX_CLIP | word | Define upper left corner of the visible area for clipping |   |   |
| 62 | _YMX_CLIP | word | Define lower right corner of the visible area for clipping |   |   |
| 64 | _XACC_DDA | word | Should be set to $8000 before each call to TXTBLT (?) |   |   |
| 66 | _DDA_INC | word | Enlargement/reduction factor $FFFF for enlargement, reduction doesn't work (?) |   |   |
| 68 | _T_SCLSTS | word | 0=reduction (?)  
1=enlargement |   |   |
| 70 | _MONO_STATUS | word | 1=not proportional font  
0=proportional type or width of character changed by bold or italics |   |   |
| 72 | _SOURCEX | word | X-coordinate of char in font |   |   |
| 74 | _SOURCEY | 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-element (ASCII value minus FIRST ADE)

The variable FIRST ADE is contained in bytes 36,37 of the font header (see example)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>76</td>
<td>_DESTX</td>
<td>word</td>
<td>X-position of char on screen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>_DESTY</td>
<td>word</td>
<td>Y-position of char on screen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>_DELX</td>
<td>word</td>
<td>Width of the character</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>_DELY</td>
<td>word</td>
<td>Height of the character</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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  _scrtpt2 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

*                      do 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.

Abacus Software
Atari ST Internals

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.

Abacus Software
Atari ST Internals

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.

Abacus Software
Atari ST Internals

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.

Abacus Software
Atari ST Internals

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.

Abacus Software
Atari ST Internals

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.

Abacus Software
Atari ST Internals

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.

Abacus Software
Atari ST Internals

The next example shows how a triangle can be drawn on the screen with the function FILLED POLYGON.
move.l #fill, patptr(a0)        * pointer of the fill pattern
move.w #4, patsk(a0)          * four fill patterns
clr.w multfill(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                 * receive 8 coordinates
loop
move.w (a5)+, (a6)+
dbra d3, loop

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 #300, 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 $0110110111011011
dc.w $0110011001100110
dc.w $1100110011001100

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.1 intin(a0),a5
    move   #0,6(a5)  * INTIN (3) = mask color value
    move   #1,8(a5)  * INTIN (4) = data color value
    add.1  #10,a5    * a5 > INTIN (5)
    lea    maus,a4   * data for new cursor
    move   #16,d0    * 32 words = 16 longs

loop:
    move.1 (a4)+(a5)+ * transfer INTIN array
    dbra   d0,loop
    .dc.w transmouse * and set form
    .dc.w init_a
    move.1 intin(a0),a0
    clr.w  (a0)       * Number Hide Cursor - ignore call
    .dc.w show_mouse * now the new cursor
```
Abacus Software

move.w #1, -(sp)   * Code CONIN wait for keypress
trap #1            * Call GEMDOS
addq.l #2, sp      * stack correction

* rts
move.w #0, -(sp)   * subroutine all done
trap #1             * terminate to desktop
                    * Call GEMDOS

maus:
makse:
      .dc.w %0000000000000000
      .dc.w %0000000000000000
      .dc.w %0000000000000000
      .dc.w %0111111111111111
      .dc.w %1111111111111111
      .dc.w %1011001110111111
      .dc.w %0000000000000000
      .dc.w %0000000000000000
      .dc.w %0000000000000000
      .dc.w %0000000000000000
      .dc.w %0000000000000000
daten:
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000
      .dc.w $0000000000000000

233
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.
<table>
<thead>
<tr>
<th>Vector number</th>
<th>Address</th>
<th>Exception vector meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$000</td>
<td>Stack pointer after reset</td>
</tr>
<tr>
<td>1</td>
<td>$004</td>
<td>Program counter after reset</td>
</tr>
<tr>
<td>2</td>
<td>$008</td>
<td>Bus error</td>
</tr>
<tr>
<td>3</td>
<td>$00C</td>
<td>Address error</td>
</tr>
<tr>
<td>4</td>
<td>$010</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>5</td>
<td>$014</td>
<td>Division by zero</td>
</tr>
<tr>
<td>6</td>
<td>$018</td>
<td>CHK instruction</td>
</tr>
<tr>
<td>7</td>
<td>$01C</td>
<td>TRAPV instruction</td>
</tr>
<tr>
<td>8</td>
<td>$020</td>
<td>Privileged violation</td>
</tr>
<tr>
<td>9</td>
<td>$024</td>
<td>Trace</td>
</tr>
<tr>
<td>10</td>
<td>$028</td>
<td>Line A emulator</td>
</tr>
<tr>
<td>11</td>
<td>$02C</td>
<td>Line F emulator</td>
</tr>
<tr>
<td>12-14</td>
<td>$030-$038</td>
<td>reserved</td>
</tr>
<tr>
<td>15</td>
<td>$03C</td>
<td>Uninitialized interrupt</td>
</tr>
<tr>
<td>16-23</td>
<td>$040-$05C</td>
<td>reserved</td>
</tr>
<tr>
<td>24</td>
<td>$060</td>
<td>Spurious interrupt</td>
</tr>
<tr>
<td>25</td>
<td>$064</td>
<td>Level 1 interrupt</td>
</tr>
<tr>
<td>26</td>
<td>$068</td>
<td>Level 2 interrupt</td>
</tr>
<tr>
<td>27</td>
<td>$06C</td>
<td>Level 3 interrupt</td>
</tr>
<tr>
<td>28</td>
<td>$070</td>
<td>Level 4 interrupt</td>
</tr>
<tr>
<td>29</td>
<td>$074</td>
<td>Level 5 interrupt</td>
</tr>
<tr>
<td>30</td>
<td>$078</td>
<td>Level 6 interrupt</td>
</tr>
<tr>
<td>31</td>
<td>$07C</td>
<td>Level 7 interrupt</td>
</tr>
<tr>
<td>32</td>
<td>$080</td>
<td>TRAP #0 instruction</td>
</tr>
<tr>
<td>33</td>
<td>$084</td>
<td>TRAP #1 instruction</td>
</tr>
<tr>
<td>34</td>
<td>$088</td>
<td>TRAP #2 instruction</td>
</tr>
<tr>
<td>35</td>
<td>$08C</td>
<td>TRAP #3 instruction</td>
</tr>
<tr>
<td>36</td>
<td>$090</td>
<td>TRAP #4 instruction</td>
</tr>
<tr>
<td>37</td>
<td>$094</td>
<td>TRAP #5 instruction</td>
</tr>
<tr>
<td>38</td>
<td>$098</td>
<td>TRAP #6 instruction</td>
</tr>
<tr>
<td>39</td>
<td>$09C</td>
<td>TRAP #7 instruction</td>
</tr>
<tr>
<td>40</td>
<td>$0A0</td>
<td>TRAP #8 instruction</td>
</tr>
<tr>
<td>41</td>
<td>$0A4</td>
<td>TRAP #9 instruction</td>
</tr>
<tr>
<td>42</td>
<td>$0A8</td>
<td>TRAP #10 instruction</td>
</tr>
<tr>
<td>43</td>
<td>$0AC</td>
<td>TRAP #11 instruction</td>
</tr>
<tr>
<td>44</td>
<td>$0B0</td>
<td>TRAP #12 instruction</td>
</tr>
<tr>
<td>45</td>
<td>$0B4</td>
<td>TRAP #13 instruction</td>
</tr>
<tr>
<td>46</td>
<td>$0B8</td>
<td>TRAP #14 instruction</td>
</tr>
<tr>
<td>47</td>
<td>$0BC</td>
<td>TRAP #15 instruction</td>
</tr>
<tr>
<td>48-63</td>
<td>$0C0-$0FC</td>
<td>reserved</td>
</tr>
<tr>
<td>64-255</td>
<td>$100-$3FC</td>
<td>User interrupt vectors</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Level</th>
<th>IPL 2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (NMI)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

236
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:

<table>
<thead>
<tr>
<th>Level</th>
<th>Vector number</th>
<th>Vector address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPL 1</td>
<td>25</td>
<td>$64</td>
</tr>
<tr>
<td>IPL 2</td>
<td>26</td>
<td>$68</td>
</tr>
<tr>
<td>IPL 3</td>
<td>27</td>
<td>$6C</td>
</tr>
<tr>
<td>IPL 4</td>
<td>28</td>
<td>$70</td>
</tr>
<tr>
<td>IPL 5</td>
<td>29</td>
<td>$74</td>
</tr>
<tr>
<td>IPL 6</td>
<td>30</td>
<td>$78</td>
</tr>
<tr>
<td>IPL 7</td>
<td>31</td>
<td>$7C</td>
</tr>
</tbody>
</table>

Only lines IPL 1 and IPL 2 are used on the Atari ST; Line IPL 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 μs 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 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 semaphor) 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 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 dmpf1g ($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, \texttt{vb1sem} 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.

<table>
<thead>
<tr>
<th>Level</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Monochrome monitor detect</td>
</tr>
<tr>
<td>14</td>
<td>RS-232 ring indicator</td>
</tr>
<tr>
<td>13</td>
<td>System clock timer A</td>
</tr>
<tr>
<td>12</td>
<td>RS-232 receive buffer full</td>
</tr>
<tr>
<td>11</td>
<td>RS-232 receive error</td>
</tr>
<tr>
<td>10</td>
<td>RS-232 transmit buffer empty</td>
</tr>
<tr>
<td>9</td>
<td>RS-232 transmit error</td>
</tr>
<tr>
<td>8</td>
<td>Line return counter, timer B</td>
</tr>
<tr>
<td>7</td>
<td>Floppy controller and DMA</td>
</tr>
<tr>
<td>6</td>
<td>Keyboard and MIDI ACIAs</td>
</tr>
<tr>
<td>5</td>
<td>Timer C</td>
</tr>
<tr>
<td>4</td>
<td>RS-232 baud rate generator, timer D</td>
</tr>
<tr>
<td>3</td>
<td>unused</td>
</tr>
<tr>
<td>2</td>
<td>RS-232 CTS</td>
</tr>
<tr>
<td>1</td>
<td>RS-232 DCD</td>
</tr>
<tr>
<td>0</td>
<td>Centronics busy</td>
</tr>
</tbody>
</table>

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
Cleans 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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Length</th>
<th>Name</th>
<th>Sample contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$400 L</td>
<td>etv_timer</td>
<td>$F526</td>
<td>This is the event timer vector of the GEM. It takes care of the periodic tasks of GEM.</td>
</tr>
<tr>
<td>$404 L</td>
<td>etv_critic</td>
<td>$5562</td>
<td>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.</td>
</tr>
<tr>
<td>$408 L</td>
<td>etv_term</td>
<td>$5328</td>
<td>This is the GEM vector for ending a program.</td>
</tr>
<tr>
<td>$40C 5L</td>
<td>etv_xtra</td>
<td></td>
<td>Here is space for 5 additional GEM vectors, which at the time are not yet used.</td>
</tr>
</tbody>
</table>
If the memory location contains the given value, the configuration of the memory controller is valid.

This is a copy of the configuration value in the memory controller. The value given applies for a 512K machine.

If the given value is located here, a jump is made at a reset via the reset vector in address $42A.

See above.

This is the physical end of the RAM memory; $80000 for a 512K machine.

The user memory begins here (TPA, transient program area).

This is the upper end of the user memory.

This "magic" value together with "memvalid" declares the memory configuration valid.

If this variable contains a value other than zero, a disk access is in progress and the VBL disk routine is disabled.
The seek rate (the time it takes to move the read/write head to the next track) is determined according to the following table:

<table>
<thead>
<tr>
<th>Seek rate</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6 ms</td>
</tr>
<tr>
<td>1</td>
<td>12 ms</td>
</tr>
<tr>
<td>2</td>
<td>2 ms</td>
</tr>
<tr>
<td>3</td>
<td>3 ms</td>
</tr>
</tbody>
</table>

The time span between two timer calls, 20 ms corresponds to 50 Hz.

If this memory location contains a value other than zero, a verify is performed after every disk write access.

Contains the device number of the drive from the operating system was loaded.

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.

If the Atari is switched from monochrome to color, it gets the new resolution from here (0=low, 1 medium resolution).

Here is a copy of the register contents for the screen resolution.

| 0 | 320x200, low resolution |
| 1 | 640x200, medium resolution |
| 2 | 640x400, high resolution |
This variable contains a pointer to the video RAM (logical screen base). The screen address must always begin on a 256 byte boundary.

If this variable is zero, the vertical blank routine is not executed.

Number of vertical blank routines.

Pointer to a list of nvbls routines which will be executed during the VBL.

If this value is not zero, it is interpreted as a pointer to a color palette which will be loaded at the next VBL.

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).

Counter for the number of VBL interrupts.

Number of VBL routines executed (not disabled by vblsem).

Vector for hard disk initialization.
Vector for changing the screen resolution. A branch is made via this vector with the screen resolution is changed (default is reset).

Vector to fetch the BIOS parameter block for a hard disk.

Read/write routine for a hard disk.

Vector to a routine to reboot the hard disk.

Media change routine for hard disk.

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.

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 retured in bits 24-31 for the BIOS function "conin"

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\ _bufl\ $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 _dskbufp $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 SL _vbl_list $15398,0,0...
List of the standard VBL routines.

$4EE W _dumpflg $FFFE
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 \text{ W } _{pztabt} \quad 0

Printer abort flag due to time-out.

$4F2 \text{ L } _{sysbase} \quad $5000

Pointer to start of the operating system.

$4F6 \text{ L } _{shell\_p} \quad 0

Global shell information.

$4FA \text{ L } _{end\_os} \quad $3B900

Pointer to the end of the operating system in RAM, start of the TPA.

$4FE \text{ L } _{exec\_os} \quad $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 decrement 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 ±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 ADDQL #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 \texttt{.W} and \texttt{.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 \texttt{.W} and \texttt{.L} are allowed, whereby the first is sign-extended to a long word.

Some listings contain instructions of the form \texttt{MOVE.L #27,D0}. The programmer then assumes that the assembler will produce \texttt{#$0000001B} from \texttt{#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 \texttt{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 \texttt{ADDA}, \texttt{ADDQ}, \texttt{ADDI}, and \texttt{ADDX}.

**AND Logical AND**

Two operand are logically combined with each other according the AND function.

Variation: \texttt{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 \texttt{C} and \texttt{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 \texttt{C} and \texttt{X}. The sign bit is shifted over from the left. See \texttt{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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA</td>
<td>Branch Always</td>
<td>no condition</td>
</tr>
<tr>
<td>BCC</td>
<td>Branch Carry Clear</td>
<td>-C</td>
</tr>
<tr>
<td>BCS</td>
<td>Branch Carry Set</td>
<td>C</td>
</tr>
<tr>
<td>BEQ</td>
<td>Branch Equal</td>
<td>Z</td>
</tr>
<tr>
<td>BGE</td>
<td>Branch Greater or Equal</td>
<td>N<em>V/-N</em>-V</td>
</tr>
<tr>
<td>BGT</td>
<td>Branch Greater Than</td>
<td>N<em>V</em>-Z/-N*-V*-Z</td>
</tr>
<tr>
<td>BHI</td>
<td>Branch Higher</td>
<td>-C*-Z</td>
</tr>
<tr>
<td>BLE</td>
<td>Branch Less or Equal</td>
<td>2/N*-V/-N*V</td>
</tr>
<tr>
<td>BLS</td>
<td>Branch Lower or Same</td>
<td>C/Z</td>
</tr>
<tr>
<td>BLT</td>
<td>Branch Less Than</td>
<td>N*-V/-N*V</td>
</tr>
<tr>
<td>BMI</td>
<td>Branch Minus</td>
<td>N</td>
</tr>
<tr>
<td>BNE</td>
<td>Branch Not Equal</td>
<td>-Z</td>
</tr>
<tr>
<td>BPL</td>
<td>Branch Plus</td>
<td>-N</td>
</tr>
<tr>
<td>BVC</td>
<td>Branch Overflow Clear</td>
<td>-V</td>
</tr>
<tr>
<td>BVS</td>
<td>Branch Overflow Set</td>
<td>V</td>
</tr>
</tbody>
</table>

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.

DBce 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.LI. A0-A6/D0-D7,-(A7)
**MOVEP Move Peripheral Data**

This specialty 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!*

265
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.

Sec 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

Abacus Software

**-----------------------------**
005000 601C  bra  $501E
005002 0100  dc.b  1,0
005004 000501E  dc.l  $501E
005008 0005000  dc.l  $5000
00500C 00019C00  dc.l  $19C00
005010 000501E  dc.l  $501E
005014 00019BF4  dc.l  $19BF4
005018 6201985  dc.w  0
00501C 0000  move.w  #$2700,SR
005022 23FS0000501E0000042A  move.l  $501E,$42A
00502C 23FC3141592690000426  move.l  #$31415926,$426
005036 41F9FFFF8800  lea  $FFFF8800,A0
00503C 10BC0007  move.b  #$7,(A0)
005040 117C00000002  move.b  #$C0,2,(A0)
005046 10BC0008  move.b  #$E,(A0)
00504A 117C00070002  move.b  #$7,2,(A0)
005050 083AD000FFC0  btest  #0,501A(PC)
005056 6706  beq  $5060
005058 3FC0002FFFF820A  move.b  #2,FFFF820A
005060 43F9FFFF8240  lea  $FFFF8240,A1
005066 303C000F  move.w  #$F,00
00506A 41FA03B0  lea  $541C(PC),A0
00506E 32D8  move.w  (A0)+,(A1)+
005070 51C9FFFC  dbra  D0,$5062
005074 9BCD  sub.l  A5,A5
005076 307C05FC  move.w  #$5FC,yA0
00507A 327C5000  move.w  #$5000,A1
00507E 7000  moveq.l  #0,0,D0
005080 30C0  move.w  D0,(A0)+
005082 3B38  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
Reset magic to resvalid
Address of the sound chip
Port A and B
To output
Select port A
Deselect floppy

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 variables
Clear D0
Clear memory range
End reached ?
005084 66FA 005086 26ED042E 00508A 91FC00080000 005090 2B40044E 005094 13ED0444FFFF8020 00509C 13ED0450FFFF8023 0050A0 323C07FF 0050A8 20C0 0050AA 20C0 0050AC 20C0 0050AE 20C0 0050B0 51C9FFFFE 0050B4 207AFFFE 0050BB 0C8B754321 0050B2 6704 0050C0 41FFFA4E 0050C4 23E80000000004FA 0050CC 23E80000000004FE 0050D4 2B7C0005AE8046A 0050DC 2B7C0005D880476 0050D8 2B7C0005B5E0472 0050EC 2B7C0005D1E047E 0050F4 2B7C00056C2047A 0050FC 2B6D044E043E 005102 286D04FA0432 005108 47F900003E2A 00510E 3B7C00060454 005114 55ED0444 005118 3B7C00030440 00511E 2B7C000512BC04C6 005126 3B7C0005FF04E2 00512C 2B7C000560004F2

bne $508C
move.l $42E(A5),A0
sub.l #$8000,A0
move.l A0,$41E(A5)
move.b $44F(A5),$FFFF8021
move.b $450(A5),$FFFF8023
move.w #$7FF,D1
move.l DC,(A0)+
move.l DO,(A0)+
move.l DC,(A0)+
move.l DC,(A0)+
dbra D1,$50A0
move.l $5C14(PC),A0
cmp.l #$87654321,(A0)
beq $50C1
lea $50D9(PC),A0
move.l 4(A0),$4FA
move.l 8(A0),$4FE
move.l #$5AE8,$46A(A5)
move.l #$5D89,$476(A5)
move.l #$5B6E,$472(A5)
move.l #$5D1E,$47E(A5)
move.l #$60B2,$47A(A5)
move.l $45E(A5),$436(A5)
move.l $4FA(A5),$432(A5)
lea $3E2A,A7
move.w #$8,9454(A5)
st $44(A5)
move.w $3,$440(A5)
move.l #$12BC,$4C6(A5)
move.w #$FFFF,$4EE(A5)
move.l #$5C00,$4F2(A5)

No
Phystop, end of RAM
Minus 32 K
As video address v be ad
Dbaseh, video address for hardware
Dbase1
($7FF+1)*16 = 32 K video RAM

Clear screen
Next 16 K bytes

End os
Exec os
Hdv init
Hdv rw
Hdv bpp
Hdv mediach
Hdv boot
V bs ad as memory top
End os as memory bottom
Initialize stack pointer
Nvbs, number of VBL routines
Pverify, Floppy Verify after write
Seekrate for floppy to 3 ms
Dskbuf to $12BC, disk buffer
Clear dumpfyl for hardcopy
Sysbase, start of the operating system
move.l $5FC, $4A2 (A5)
move.l $5326, $46E (A5)
lea $5542 (SC), A3
lea $5328 (SC), A4
cmp.l $FA52235F, $FA0000
beq $517E
lea $5838 (PC), A1
add.l $2000000, A1
lea $8, A0
move.w $3D, D0
move.l A1, (A0)+
add.l $1000000, A1
dbza D0, $516C
move.l A3, $14
move.l $5452, $70 (A5)
move.l $543C, $68 (A5)
move.l A3, $68 (A5)
move.l $556C, $68 (A5)
move.l $5566, $68 (A5)
move.l $556A, $28 (A5)
move.l A4, $400 (A5)
move.l $5562, $404 (A5)
move.l A4, $408 (A5)
lea $4CC (A5), A0
move.l A0, $556 (A5)
move.w $7, D0
cir.l (A0)+
dbza D0, $516C
bsr $6EE2
moveq.l $2, D0
bsr $52FE
sub.l A5, A5

Savptr for TRAPs to $5FC
Adv dsb auf rts
Pointer to rte
Pointer to rts
Cartbase, Diagnostic cartridge inserted?
Yes, don't initialize vectors
Terminate process
Vector number is in bits 24-31
Start with bus error
Number of vectors
Set vector
Increment number in bits 24-31
Next vector
'Division by zero' to rts
Vertical Blank Interrupt, IPL4
Horizontal Blank Interrupt, IPL2
TRAP #2 to rts
TRAP #13 vector
TRAP #14 vector
LINE A vector
Btv timer to rts
Btv critic vector
Btv term to rts
Vbl list
As pointer to vblqueue
8 vectors
Clear list
Next vector
Initialize MFP
Bit number
Cart scan, test cartridge
Clear A5
0051DB 6100036A 0051DC 61000366
0051E0 103C0002 0051E4 08390007FFFFFA01
0051EC 670C 0051E8 1020044A
0051F2 B03C0002 0051F6 6D02
0051FA 420C 0051FC 1B40044C
005204 1300FFF8260 005208 660A
00520A 33F9FFFF825EFFFF8246
005214 4EB90000F6C4 00521A 2B7C000050100046
005222 33FC00100000452 005226 A240
00522C 610000D0 005230 46FC2300
005234 7001 005236 610000C6
00523A 61004234 00523E 610000B0
005242 4A7900000492 005248 6718
00524A 61003C64 00524E 610006FE
005252 487A0099 005256 487A0095
00525A 487A007E 00525E 4267

bar $5544
bar $5546
move.b #2,DO
bst #7,$FFFFFA01
beq $51FA
move.b $44A(A5),D0
cmp.b #2,D0
bic $51FA
clr.b D0
move.b D0,$44C(A5)
move.b D0,$FFFF8260
cmp.b $1,D0
bne $5214
move.w $FFFF825E,$FFFF8246
jsr $F6C4
move.l $501E,$60E(A5)
move.w $1,6452
clr.w D0
bsr $52FE
move.w $2330,SR
move.q.1 #1,DO
bar $52FE
bar $9470
bar $52FO
tst.w $482
beq $5262
bar $8EBC
bar $594E
pea $52ED(PC)
pea $52ED(PC)
pea $52DA(PC)
clr.w -(A7)

NVbl, wait for next picture return
NVbl, wait for next picture return
Default resolution to monochrome
Test MFP qip monochrome detect
No monochrome monitor?
Get defshift

Color mode?
Yes
Otherwise select low resolution
Save ashift

Shifted, program shifter
Medium resolution?
No

Copy color 15 (black) to color 3

Initialize screen output

Hdv dsb

V Olsen, free VEL

Bit number 0

Cartscan, test cartridge

IPL to 3

Bit number 1

Cartscan, test cartridge

Initialize DOS

Dskboot

Caddload, load shell from disk?

No

Enable cursor

Auto exec

Null name

Null name

"COMMAND.PRG"
005260  605C
005262  6100EBA
005266  41FAC066
00526A  327C0502
00526E  0C100023
005272  6602
005274  2449
005276  12DB
005278  6AF4
00527A  10390000646
005280  D0300041
005284  1480
005286  48900000502
00528C  48700000528D
005292  487A0059
005296  3F3000C5
00529A  3F30004B
00529E  4E41
0052A0  DEFC000E
0052A4  2040
0052A6  2179000004F00008
0052AE  48700000502
0052B4  2F08
0052B6  487A0035
0052BA  3F300044
0052BE  3F30004B
0052C2  4E41
0052C4  DEFC000E
0052C8  42F90000501E
0052CE  304154683D00
0052D4  2335C00FF
0052DA  43F9D5D414E4425052
dc.b  'COMAND.FRG' 0

'&' drive indicator?
No
Address of the drive indicator
Copy drive number
Copy entire string
Bootdev
'A', calculate drive number
Insert in filename
Environrnent string
Null name
Null name

Exec, load program
GEMDOS call
Correct stack pointer
Save base page address
Exec os

Exec, load program
GEMDOS call
Correct stack
If return, then Reset
Dskboot, boot from disk
Bit number 3
Test cartridge
Hdv boot, load boot vector
And start attempt

Test cartridge
Cartbase
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
Clear test pattern
End address of the test
Test memory contents
Unequal, terminate
Create next test pattern
End address reached?
No, continue test
Back to the calling address

Clear test pattern
End address of the test
Test memory contents
Unequal, terminate
Create next test pattern
End address reached?
No, continue test
Back to the calling address

Clear test pattern
End address of the test
Test memory contents
Unequal, terminate
Create next test pattern
End address reached?
No, continue test
Back to the calling address
005386 1039FFFF8201 move.b $FFFF8201,D0 Dbasew
00538E E148 lsl.w #8,D0 Dbasew
005390 1039FFFF8203 move.b $FFFF8203,D0 Dbasew
005396 E188 lsl.l #8,D0
005398 D1CO add.l D0,A0 Dbaseh
00539A 7E0F moveq.l #15,D7
00539C 3C3B4078 move.w $5416(PC,D4.w),D6
0053A0 3605 move.w D5,D3
0053A2 2148 move.l A0,A2
0053A4 D4FB466A add.w $5410(PC,D4.w),A2
0053A8 1011 move.b (A1),D0
0053AA 4FBA0004 lea $53BC(PC),A5
0053AE 6642 bra $53F2
0053B0 3202 move.w D2,D1
0053B2 16290001 move.b 1(A1),D0
0053B6 4FBA0004 lea $53BC(PC),A5
0053BA 6036 bra $53F2
0053BC 3C11 move.w (A1),D0
0053BE 4FBA0002 jmp $53C2(PC,D4.w)
0053C2 6004 bra $53C8 Low resolution
0053C4 600C bra $53D2 Medium resolution
0053C6 6014 bra $53DC High 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 $53EC

*****************************************************
0053D2 30C1 move.w D1,(A0)+ Medium resolution
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, $53A6
0053E4 206A
move.l A2, A0
0053E6 51CEFFB0
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
xorl.b #1, D0
0053FA 40E7
move.w SR, -(A7)
0053FC E352
xorl.w #1, D2
0053FE 46DF
move.w (A7)+, SR
005400 E352
xorl.w #1, D2
005402 51CBFFT4
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 0355       dc.w $355  
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 07C0700    and.w #$700,D0  
005446 6606       bne $544E  
005448 06F03000002 or.w #$300,2(A7)  
00544E 301F       move.w (A7)+,D0  
005450 4E73       rte  

Color palatte
  White
  Red
  Green
  Yellow
  Blue
  Magenta
  Blue-green
  Light grey
  Grey
  Light red
  Light green
  Light yellow
  Light blue
  Light magenta
  Light blue-green
  black

HBL interrupt
  Save D0
  Get status from stack
  Isolate IPL mask
  Not equal to zero ?
  Otherwise IPL tp 3
  Restore D0
005452 328900000466
005454 327900000452
005454 6B00000DC
005452 48E7FFFE
005452 52B900000462
005454 9ECD
00545C 1039FF8260
00545E C03C0003
00545F B03C0002
005460 6C10
00546E 08390007FF8FA01
00546E 662C
00546F 103C0002
005470 6016
00547A 08390007FF8FA01
00547E 611C
00547F 102D044A
005480 B03C0002
005485 6D02
005485 4200
005486 1B00044C
005487 13C0FF8260
005488 206D0462
005489 4E90
00548A 61003B42
00548B 9ECD
00548C 4AADD45A
00548E 6718
0054C0 206D045A
0054C4 43FFFFFFFF240
0054CA 303C000F

**---------------------------**
addq.l #1, $466
subq.w #1, $452
bmi $553C
movem.l D0-D7/A0-A6,-(A7)
addq.l #1, $462
subl.l A5,A5
move.b $FF8260,D0
and.b #3,D0
cmp.b #2,D0
bge $546E
btst #7, $FF8FA01
bne $54B4
move.b $2, D0
bra $54A4
btst #7, $FF8FA01
beq $54B4
move.b $4A(A5), D0
cmp.b #2, D0
bit $54A4
clr.b D0
move.b D0, $44C(A5)
move.b D0, $FF8260
move.l $46E(A5), A0
jex (A0)
bst $8FF0
sub.l A5, A5
tst.l $43A(A5)
beq $5408
move.l $45A(A5), A0
lea $FF8260, A1
move.w $8F, D0

VBL interrupt
Increment frclock
Vblsem
VBL interrupt disabled ?
Save registers
Increment vbclock erh\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 shiftmd
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
execute routine

VBL routine for floppies

NVbls, number of VBL routines

No vectors?

Convert to dbase counter

Address of the vblqueue

Get vector

Zero?

Don't execute routine

Save registers

Execute routine

Restore registers

Test next vector

Clear A5

Dumpflag, hardcopy desired?

No

Hardcopy

Clear dumpflag again

Restore registers

Vblsem, restore VBL again
Wml, wait for next V-sync
Save status
IPL 0, allow interrupts
Load frclock
Frclock
Still equal?
Restore status

Critical error handler
Etv critic

TRAP #14
Address of the TRAP #14 routines

TRAP #13
Address of the TRAP #13 routines
Savptr, pointer to save area
Status register to DO
Save in save area
Save PC
And save C register
Update savptr
Call from supervisor mode?
Yes
Else use USP
Get function number from stack
Compare with maximal number
Too large, stop
Convert number at 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, bconstat
3, bconstat
4, (indirect) rwabs
5, sstexec
6, tickcal
7, (indirect) getbpb
8, bcostat
9, (indirect) mediach
10, drvmap
11, shift
Addresses of the TRAP #14 calls

<table>
<thead>
<tr>
<th>Routine</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0028</td>
</tr>
<tr>
<td>1</td>
<td>0028</td>
</tr>
<tr>
<td>2</td>
<td>0028</td>
</tr>
<tr>
<td>3</td>
<td>0028</td>
</tr>
<tr>
<td>4</td>
<td>0028</td>
</tr>
<tr>
<td>5</td>
<td>0028</td>
</tr>
<tr>
<td>6</td>
<td>0028</td>
</tr>
<tr>
<td>7</td>
<td>0028</td>
</tr>
<tr>
<td>8</td>
<td>0028</td>
</tr>
<tr>
<td>9</td>
<td>0028</td>
</tr>
<tr>
<td>10</td>
<td>0028</td>
</tr>
<tr>
<td>11</td>
<td>0028</td>
</tr>
<tr>
<td>12</td>
<td>0028</td>
</tr>
<tr>
<td>13</td>
<td>0028</td>
</tr>
<tr>
<td>14</td>
<td>0028</td>
</tr>
<tr>
<td>15</td>
<td>0028</td>
</tr>
<tr>
<td>16</td>
<td>0028</td>
</tr>
<tr>
<td>17</td>
<td>0028</td>
</tr>
<tr>
<td>18</td>
<td>0028</td>
</tr>
<tr>
<td>19</td>
<td>0028</td>
</tr>
<tr>
<td>20</td>
<td>0028</td>
</tr>
<tr>
<td>21</td>
<td>0028</td>
</tr>
<tr>
<td>22</td>
<td>0028</td>
</tr>
<tr>
<td>23</td>
<td>0028</td>
</tr>
<tr>
<td>24</td>
<td>0028</td>
</tr>
<tr>
<td>25</td>
<td>0028</td>
</tr>
<tr>
<td>26</td>
<td>0028</td>
</tr>
<tr>
<td>27</td>
<td>0028</td>
</tr>
<tr>
<td>28</td>
<td>0028</td>
</tr>
<tr>
<td>29</td>
<td>0028</td>
</tr>
</tbody>
</table>

Number of routines

0, initmouse
1, cts
2, physbase
3, logbase
4, getrez
5, setscreen
6, setpalette
7, setcolor
8, floprd
9, flopwr
10, flopfmt
11, gdbab
12, mids
13, mfpint
14, iorec
15, rsconf
16, keytrans
17, rand
18, probt
19, flopver
20, dumpit
21, cursconf
22, settle
23, gettime
24, bioskeys
25, ikbdws
26, jdisint
27, javabint
28, giacces
29, offgibit
005666 00007A74       dc.l $7A74
00566A 00007B8A       dc.l $7B8A
00566E 00007C0C       dc.l $7C0C
005672 00007C20       dc.l $7C20
005676 00007C54       dc.l $7C54
00567A 00007C32       dc.l $7C32
00567E 00007D92       dc.l $7D92
005682 00005544       dc.l $5544
005686 0000568E       dc.l $568E
00568A 0000581C       dc.l $581C

move . 1
jmp 4(A7),AO

*****************************

lea $56B6(PC),AO
bra $56AA

*****************************

lea $56CE(PC),AO
bra $56AA

*****************************

lea $56E6(PC),AO
bra $56AA

*****************************

lea $56FE(PC),AO

30, ongibit
31, xbtimer
32, dosound
33, setprt
34, ikbdvecs
35, kbrate
36, prtblk
37, wvbl
38, supexec
39, puntaes

supexec, routine in supervisor mode
Get address from stack
Execute routine in supervisor mode

bconst, get input status
Status table

bconst, get output status
Status table

bconstout, output
Conout
Get device number
Times 4
Get address of the routine
Execute routine

Input status
Rts
RS 232 status
Console status
MIDI status
RTS

Output status
Centronics status
RS 232 status
Console status
Keyboard status
MIDI status
rts
**Output**
- Centronics output
- RS 232 output
- Console output
- MIDI output
- Keyboard output
- ASCII output

**Drvmap, get active floppies**
- Drvbits, get bit vector

**Shift, keyboard status**
- Kshift, get shift status
- Get parameters from stack
- Negative, then set
- Accept as kbshift

**Getmpb, Memory Parameter Block**
- New MPB
- Themd, Memory descriptor
- Mp mfl = address of the MD
- Mp mal = 0
- Mp rover = address of the MD
- Clear m link
- Membot as mstart
- Memtop
- Minus membot
- Length as m length
- M own = 0
Abacus Software

Page 1 of 2

Abacus Software

Page 2 of 2
005796 102D8260 move.b $FFFF8260(A5),D0
00579A C03C0003 and.b #$3,D0
00579E 4E75 rts

Load shiftmd
Isolate bits 0 and 1

******************************************************************************
00579A0 4AAF0004 tst.l 4(A7)
00579A4 6B06 bmi $57AC
00579A6 2B6F0004044E move.l 4(A7),$44E(A5)
00579AC 4AAFC008 tst.l 8(A7)
00579B0 6B10 bmi $57C2
00579B2 135FC009FFFF8201 move.b 9(A7),$FF8201
00579BA 13EF000AFFFF8203 move.b 10(A7),$FF8203
00579C2 4A6F000C tst.w 12(A7)
00579C6 6B24 bmi $57EC
00579CE 1B6F000D044C move.b 13(A7),$44C(A5)
00579D2 6100PD74 bsr $5544
00579D2 3E6900FFFF820C move.b $4C(A5),$FF820C
00579DA 426D0452 clr.w $52(A5)
00579DE 4EB900046C4 jr $56C4
00579E4 33FC00100000452 move.w #1,$452
00579E8 4E75 rts

Setscreen, set screen address
Logical address
Negatie, don't set
Set v bs ad
Physical address
Negative, don't set
Dbaseh
Dbasel
Video resolution
Negative, don't set
Sshiftmd
Vwbl, wait for VBL
Sshiftmd to shiftmd
Vbsem, VBL disabled
Initialize screen output
Vbsem, permit VBL again

******************************************************************************
0057EE 2B6F0004045A move.l 4(A7),$45A(A5)
0057F4 4E75 rts

Setpalette, load new color palette
Set colorptr (execution in VBL)

******************************************************************************
0057F4 322FD004 move.w 4(A7),D1
0057FA D241 add.w D1,D1
0057FD C27C001T and.w #61F,D1
005800 41F9FFFF8240 lea $FFFF8240,A0
005806 30301000 move.w 0(A0,D1.w),D0

Setcolor, set single color
Color number
Times 2
Limit to valid number
Address colorO
Get old color
**Alien Software**

```
00580A C07C0777
00580C 4A6F0006
005812 6B06
005814 31AF00061000
00581A 4E75

and.w  #8777, D0
tst.w  @A7)
bmi  $581A
move.w  @A7),0(A0,D1.w)

 pneumonia, clear AES and restart
os magic
Already there ?
No, done
In ROM ?
Yes, do nothing
Clear magic
To reset
```

```
00581C 207AF7F6
005820 0C9087654321
005826 6C06
005828 B1F90000042E
00582E 4E71

move.l  $5014(PC),AO
cmp.l  #$87654321,(AO)
bne  $5836
cmp.l  $42E,A0
bge  $5836
clr.l  (A0)
bra  $501E

Puntaes, clear AES and restart
os magic
Already there ?
No, done
In ROM ?
Yes, do nothing
Clear magic
To reset
```

```
005838 0162
00583A 4E71
00583C 23DF00003C4
005842 48F9FFFE00003B4
00584A 4E68
00584C 23C8000003C8
005852 303C000F
005856 41F9000003CC
00585C 224F
005862 30D9
005864 51C8FFFC
005864 23FC12A56780000380
00586E 7200
005870 1239000003C4
005876 5341

bar  $583C
nop
move.l  (A7)+, 3C4
movem.l  D0-D7/A0-A7,$3B4
move.l  USP, AO
move.l  AO, 3C8
move.w  #$F, D0
lea  $3C3, AO
move.l  A7, A1
move.w  (A1)+, (AO)+
dbra  DC, 5852
move.l  #$12345678,$3B0
moveq.l  #0,D1
move.b  3C4, D1
subq.w  #1, D1
```

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 A1
Convert in dbra counter
Abacus Software

**Atari ST Internals**

### $5890 Output appropriate # of 'mushrooms'

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>005878 6116</td>
<td>bsr</td>
<td>$5890</td>
</tr>
<tr>
<td>00587A 23FC00005FC00004A2</td>
<td>move.l</td>
<td>#$5FC,$4A2 Reset savptr for BIOS</td>
</tr>
<tr>
<td>005884 3F3C0001</td>
<td>move.w</td>
<td>#1,-(A7) Return code for error</td>
</tr>
<tr>
<td>005888 42A7</td>
<td>clrl</td>
<td>-/(A7) Terminate process</td>
</tr>
<tr>
<td>00588A 4241</td>
<td>trap</td>
<td>#1</td>
</tr>
<tr>
<td>00588C 6000F790</td>
<td>bra</td>
<td>$501E</td>
</tr>
</tbody>
</table>

### $5890

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>005890 1E3FFPPF8260</td>
<td>move.b</td>
<td>$FFFF8260,D7 Shiftmd, get resolution</td>
</tr>
<tr>
<td>005896 CE7C0003</td>
<td>and.w</td>
<td>#3,D7 Isolate significant bits</td>
</tr>
<tr>
<td>00589A DE47</td>
<td>add.w</td>
<td>D7,D7 Times 2 for word access</td>
</tr>
<tr>
<td>00589C 4280</td>
<td>clrl</td>
<td>D0</td>
</tr>
<tr>
<td>00589E 1039FFFF8201</td>
<td>move.b</td>
<td>$FFFF8201,D0</td>
</tr>
<tr>
<td>0058A4 E148</td>
<td>lsl.w</td>
<td>#8,D0</td>
</tr>
<tr>
<td>0058A6 1039FFFF8203</td>
<td>move.b</td>
<td>$FFFF8203,D0</td>
</tr>
<tr>
<td>0058AC E188</td>
<td>lsl.l</td>
<td>#8,D0</td>
</tr>
<tr>
<td>0058A2 2040</td>
<td>move.l</td>
<td>D9,A0</td>
</tr>
<tr>
<td>0058B0 D0FB7C02A</td>
<td>add.w</td>
<td>$580C(PC,D7.w),A0 Plus offset for middle of screen</td>
</tr>
<tr>
<td>0058B4 43FA0038</td>
<td>lea</td>
<td>$58EE(PC),A1 Address of bit pattern for &quot;mushroom&quot;</td>
</tr>
<tr>
<td>0058B9 3C3C000F</td>
<td>move.w</td>
<td>#5F,D6</td>
</tr>
<tr>
<td>0058BC 3401</td>
<td>move.w</td>
<td>D1,D2</td>
</tr>
<tr>
<td>0058BE 2448</td>
<td>move.l</td>
<td>A0,A2</td>
</tr>
<tr>
<td>0058C0 3A3B7C20</td>
<td>move.w</td>
<td>$58E2(PC,D7.w),D5 Number of words (screen planes)</td>
</tr>
<tr>
<td>0058C4</td>
<td>move.w</td>
<td>(A1), (AC)+ Write a raster line</td>
</tr>
<tr>
<td>0058C6 51C0FFFC</td>
<td>dbra</td>
<td>D5,$58C4 A complete mushroom</td>
</tr>
<tr>
<td>0058CA 51C0FFFC</td>
<td>dbra</td>
<td>D2,$58C0 The next on the same line</td>
</tr>
<tr>
<td>0058CE 5449</td>
<td>add.w</td>
<td>#2,A1</td>
</tr>
<tr>
<td>0058D0 D4FB7C16</td>
<td>add.w</td>
<td>$58E8(PC,D7.w),A2 Next word of the bit pattern</td>
</tr>
<tr>
<td>0058D4 204A</td>
<td>move.l</td>
<td>A2,A0</td>
</tr>
<tr>
<td>0058D6 51CEFFE4</td>
<td>dbra</td>
<td>D6,$582C Next destination address</td>
</tr>
<tr>
<td>0058DA 4E75</td>
<td>rts</td>
<td></td>
</tr>
</tbody>
</table>

---

*Write 'mushrooms' on screen*

*Shiftmd, get resolution*

*Isolate significant bits*

*Times 2 for word access*

*Dbaseh*

*Dbasal*

*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*
<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00580C 3E80</td>
<td>dc.w</td>
<td>100*160</td>
</tr>
<tr>
<td>00580E 3E80</td>
<td>dc.w</td>
<td>100*160</td>
</tr>
<tr>
<td>0058B0 3E80</td>
<td>dc.w</td>
<td>200*80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0058E2 0003</td>
<td>dc.w</td>
<td>3</td>
</tr>
<tr>
<td>0058E4 0001</td>
<td>dc.w</td>
<td>1</td>
</tr>
<tr>
<td>0058E6 0000</td>
<td>dc.w</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0058E8 00A0</td>
<td>dc.w</td>
<td>160</td>
</tr>
<tr>
<td>0058EA 00A0</td>
<td>dc.w</td>
<td>160</td>
</tr>
<tr>
<td>0058EC 0050</td>
<td>dc.w</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0058EE 07C0</td>
<td>dc.w</td>
<td>$000001111100000</td>
</tr>
<tr>
<td>0058F0 1FF0</td>
<td>dc.w</td>
<td>$000011111111000</td>
</tr>
<tr>
<td>0058F2 3BF8</td>
<td>dc.w</td>
<td>$001110111111000</td>
</tr>
<tr>
<td>0058F4 77F4</td>
<td>dc.w</td>
<td>$011101111110100</td>
</tr>
<tr>
<td>0058F6 B7FA</td>
<td>dc.w</td>
<td>$101101111111010</td>
</tr>
<tr>
<td>0058F8 BBFA</td>
<td>dc.w</td>
<td>$101110111111100</td>
</tr>
<tr>
<td>0058FA DFF6</td>
<td>dc.w</td>
<td>$110111111110110</td>
</tr>
<tr>
<td>0058FC 66FC</td>
<td>dc.w</td>
<td>$011001101111100</td>
</tr>
<tr>
<td>0058FE 3289</td>
<td>dc.w</td>
<td>$011001011000100</td>
</tr>
<tr>
<td>005900 0280</td>
<td>dc.w</td>
<td>$000000101000000</td>
</tr>
<tr>
<td>005902 0440</td>
<td>dc.w</td>
<td>$000010001000000</td>
</tr>
<tr>
<td>005904 0440</td>
<td>dc.w</td>
<td>$000001000100000</td>
</tr>
<tr>
<td>005906 0540</td>
<td>dc.w</td>
<td>$000010010010000</td>
</tr>
<tr>
<td>005908 0520</td>
<td>dc.w</td>
<td>$000001000100000</td>
</tr>
<tr>
<td>00590A 0920</td>
<td>dc.w</td>
<td>$000010001000000</td>
</tr>
<tr>
<td>00590C 0920</td>
<td>dc.w</td>
<td>$000010001000000</td>
</tr>
</tbody>
</table>

**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  d(A7),A0
005914 226F0008  move.l  b(A7),A1
005918 303C003P  move.w  #$3F,D0
00591C 12D8  move.b  (AC)++, (AI)+
005920 12D8  move.b  (AC)++, (AI)+
005922 12D8  move.b  (AC)++, (AI)+
005924 12D8  move.b  (AC)++, (AI)+
005926 12D8  move.b  (AC)++, (AI)+
005928 12D8  move.b  (AC)++, (AI)+
00592A 12D8  move.b  (AC)++, (AI)+
00592C 12D8  move.b  (AC)++, (AI)+
00592E 12D8  move.b  (AC)++, (AI)+
005930 51C8FFEE  dbra  D0,5591C
005932 4E75  rts

******************************************************
00593A 5C4155544F5C  dc.b  '1AUTO1'
005940 2A2E555247C0  dc.b  '*PRG',0
005946 12345678ABCDEF0  dc.l  $12345678, $5ABCDEF0

******************************************************
00594E 41FAFFEA  lea  $543 (PC),A0
005952 43FAFFEC  lea  $594C(PC),A1
005956 23DF00005FC  move.l  (A7)+,5FC
00595A 9BCD  sub.l  A5,A5
00595E 2B48C600  move.l  A0,$600(A5)
005962 2B49C604  move.l  A1,$604(A5)

Fastcopy, copy disk sector
Source address
Destination address
(63+1)*8 = 512 bytes

Copy 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
move.l $4C2(A5),D0
move.w $446,D1
btst D1,D0
beq $59A
lea $52ED(PC),A0
move.l A0,-(A7)
move.l A0,-(A7)
move.w #$5,-(A7)
move.w #$4B,-(A7)
trap #1
add.w #$10,A7
move.l D0,A0
move.l #$59AC,8(A0)
move.l A3,-(A7)
move.l D0,-(A7)
move.l A3,-(A7)
move.w #$4,-(A7)
move.w #$4B,-(A7)
trap #1
add.w #$10,A7
rts

************
clr.l -(A7)
move.w #$20,-(A7)
trap #1
addq.w #$6,A7
move.l D0,A4
move.l 4(A7),A5
lea $100(A5),A7
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
Abacus Software
Atari ST Internals

293

0059C6 2FC0
0059C8 4267
0059CA 3F3C004A
0059CE 4241
0059D0 5C4F
0059D2 4A40
0059D4 665A
0059D6 3F3C0007
0059DA 2F3900000600
0059E0 3F3C000E
0059E4 7E08
0059E6 487900000608
0059EC 3F3C001A
0059F0 4E41
0059F2 5C4F
0059F4 4241
0059F6 DEC7
0059FA 4A40
0059FE 6644
0059FC 207900000600
005A02 247900000604
005A08 43F900000634
005A0E 12D8
005A10 B5C8
005A12 66FA
005A14 41F900000626
005A1A 12D8
005A1C 66FC
005A1E 487AF8CD
005A22 487AF8C9
005A26 487900000634
005A2C 4267

move.l A5,-(A7)
clr.w -(A7)
move.w #$4A,-(A7)
trap #1
addq.w #$6,A7
tst.w D0
bne $5A40
move.w #$7,-(A7)
move.l $600,-(A7)
move.w #$6E,-(A7)
moveq.l #$D7
pea $608
move.w #$1A,-(A7)
trap #1
addq.w #$6,A7
trap #1
add.w D7,A7
tst.w D0
bne $5A40
move.l $600,A0
move.l $604,A2
lea $634,A1
move.b (A0)+,(A1)+
cmp.l A0,A2
bne $5A9E
lea $626,A0
move.b (A0)+,(A1)+
bne $5A1A
lea $52ED(PC)
lea $52ED(PC)
lea $634
clr.w -(A7)

Start of the memory area

Setblock
GENDOS call
Correct stack pointer
Error ?
Yes
R/O, hidden and system files
Filename
Search first
Bytes for stack correction
DNA address for DOS
Setdata
GENDOS call
Correct stack pointer
GENDOS call
Correct stack pointer
File found ?
No
Pathname
Filename
Auto name
Copy path part

Name from DMA buffer
Append to path

Null name
Null name
Filename
Load and start program
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>005A2E</td>
<td>3F3C004B</td>
<td>move.w #$4B,-(A7)</td>
<td>Exec, load program</td>
</tr>
<tr>
<td>005A32</td>
<td>4E41</td>
<td>trap #1</td>
<td>GMDOs call</td>
</tr>
<tr>
<td>005A34</td>
<td>DEF0C010</td>
<td>add.w #$10,A7</td>
<td>Correct stack pointer</td>
</tr>
<tr>
<td>005A38</td>
<td>7B02</td>
<td>move.l #$2,D7</td>
<td>Bytes for stack pointer</td>
</tr>
<tr>
<td>005A3A</td>
<td>3F3C006F</td>
<td>move.w #$4F,-(A7)</td>
<td>Search next</td>
</tr>
<tr>
<td>005A3E</td>
<td>60A6</td>
<td>bra #$59E6</td>
<td>Next file</td>
</tr>
<tr>
<td>005A40</td>
<td>4FF900C3E2A</td>
<td>lea #$3E2A,A7</td>
<td>Stack pointer back to start value</td>
</tr>
<tr>
<td>005A46</td>
<td>2F39000005FC</td>
<td>move.l #$5FC,-(A7)</td>
<td>Load return address</td>
</tr>
<tr>
<td>005A4C</td>
<td>4E75</td>
<td>rts</td>
<td></td>
</tr>
<tr>
<td>005A4E</td>
<td>42790000D042E</td>
<td>clr.w #$4EE</td>
<td>Dumpit, screen hardcopy</td>
</tr>
<tr>
<td>005A54</td>
<td>610A</td>
<td>bsr #$5A60</td>
<td>Set dumpit</td>
</tr>
<tr>
<td>005A56</td>
<td>33FCFFFF000004EE</td>
<td>move.w #-1,$4EE</td>
<td>Hardcopy</td>
</tr>
<tr>
<td>005A5E</td>
<td>4E75</td>
<td>rts</td>
<td>Clear dumpit</td>
</tr>
<tr>
<td>005A60</td>
<td>9BCD</td>
<td>sub.l A5,A5</td>
<td>Scmdmp, hardcopy</td>
</tr>
<tr>
<td>005A62</td>
<td>2B6D0440654</td>
<td>move.l #$4E(A5),$654(A5)</td>
<td>Clear A5</td>
</tr>
<tr>
<td>005A68</td>
<td>426D0658</td>
<td>clr.w #$658(A5)</td>
<td>v bs ad, screen output</td>
</tr>
<tr>
<td>005A6C</td>
<td>4240</td>
<td>clr.w D0</td>
<td>Offset to null</td>
</tr>
<tr>
<td>005A6E</td>
<td>102D044C</td>
<td>move.b #$4C(A5),D0</td>
<td>Sshiftmd, screen resolution</td>
</tr>
<tr>
<td>005A72</td>
<td>3B600062</td>
<td>move.w D0,$662(A5)</td>
<td>Save</td>
</tr>
<tr>
<td>005A76</td>
<td>D040</td>
<td>add.w D0,D0</td>
<td>Times 2</td>
</tr>
<tr>
<td>005A78</td>
<td>41FA005A</td>
<td>lea #$5AD4(PC),A0</td>
<td>Table for screen resolution</td>
</tr>
<tr>
<td>005A7C</td>
<td>3870000065A</td>
<td>move.w 0(A0,D0.w),$65A(A5)</td>
<td>Get screen resolution</td>
</tr>
<tr>
<td>005A82</td>
<td>3870006065C</td>
<td>move.w 0(A0,D0.w),$65C(A5)</td>
<td>Get screen width</td>
</tr>
<tr>
<td>005A88</td>
<td>426D065E</td>
<td>clr.w #$65E(A5)</td>
<td>Get screen height</td>
</tr>
<tr>
<td>005A8C</td>
<td>426D0660</td>
<td>clr.w #$660(A5)</td>
<td>Left</td>
</tr>
<tr>
<td>005A90</td>
<td>287C00F082400666</td>
<td>move.l #$FF0240,8666(A5)</td>
<td>And right to zero</td>
</tr>
<tr>
<td>005A98</td>
<td>426D066E</td>
<td>clr.w #$66E(A5)</td>
<td>Address to color palette</td>
</tr>
<tr>
<td>005A9C</td>
<td>322D00ABC</td>
<td>move.w #$ABC(A5),D1</td>
<td>Clear mark pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Get printer configuration</td>
</tr>
<tr>
<td>Memory Address</td>
<td>Instruction</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>005AA0 E649</td>
<td>lsr.w #3,D1</td>
<td>Test / quality mode</td>
<td></td>
</tr>
<tr>
<td>005AA2 C27C0001</td>
<td>and.w #1,D1</td>
<td>Isolate bit</td>
<td></td>
</tr>
<tr>
<td>005AA6 3B410664</td>
<td>move.w D1,$664(A5)</td>
<td>And save</td>
<td></td>
</tr>
<tr>
<td>005AAA 322DC98C</td>
<td>move.w $A8C(A5),D1</td>
<td>Get printer configuration</td>
<td></td>
</tr>
<tr>
<td>005AAB 3001</td>
<td>move.w D1,D0</td>
<td>Parallel / Serial</td>
<td></td>
</tr>
<tr>
<td>005AB0 E848</td>
<td>lsr.w #4,D0</td>
<td>Isolate</td>
<td></td>
</tr>
<tr>
<td>005AB2 C37C0001</td>
<td>and.w #1,D0</td>
<td>And save for hardcopy</td>
<td></td>
</tr>
<tr>
<td>005AB6 3B40066C</td>
<td>move.w D0,$66C(A5)</td>
<td>Isolate printer type</td>
<td></td>
</tr>
<tr>
<td>005ABA C27C0007</td>
<td>and.w #7,D1</td>
<td>Get assignment from table</td>
<td></td>
</tr>
<tr>
<td>005ABE 103B102D</td>
<td>move.b $3AE0(PC,D1.w),D0</td>
<td>And save for hardcopy</td>
<td></td>
</tr>
<tr>
<td>005AC2 33C0000066A</td>
<td>move.w D0,$66A</td>
<td>Address of the parameter block</td>
<td></td>
</tr>
<tr>
<td>005AC8 486D0654</td>
<td>pea $554(A5)</td>
<td>Perform hardcopy</td>
<td></td>
</tr>
<tr>
<td>005ACC 61022C9</td>
<td>bsr $7D92</td>
<td>Correct stack pointer</td>
<td></td>
</tr>
<tr>
<td>005AD0 596F</td>
<td>addq.w #4,A7</td>
<td>Parameter table for hardcopy</td>
<td></td>
</tr>
<tr>
<td>005AD2 4E75</td>
<td>rts</td>
<td>Screen widths</td>
<td></td>
</tr>
<tr>
<td>005AD4 014002600280</td>
<td>dc.w 320,640,640</td>
<td>Screen heights</td>
<td></td>
</tr>
<tr>
<td>005AD8 00C800C80190</td>
<td>dc.w 200,200,400</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parameter table for hardcopy**

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>005AE0 00</td>
<td>dc.b</td>
<td>Printer types (-1 = not implemented)</td>
</tr>
<tr>
<td>005AE1 02</td>
<td>dc.b</td>
<td>ATARI B/W matrix</td>
</tr>
<tr>
<td>005AE2 01</td>
<td>dc.b</td>
<td>ATARI B/W daisy wheel</td>
</tr>
<tr>
<td>005AE3 FF</td>
<td>dc.b</td>
<td>ATARI color matrix</td>
</tr>
<tr>
<td>005AE4 03</td>
<td>dc.b</td>
<td>(ATARI color daisy wheel ?)</td>
</tr>
<tr>
<td>005AE5 FF</td>
<td>dc.b</td>
<td>Epson B/W matrix</td>
</tr>
<tr>
<td>005AE6 FF</td>
<td>dc.b</td>
<td>(Epson B/W daisy wheel)</td>
</tr>
<tr>
<td>005AE7 FF</td>
<td>dc.b</td>
<td>(Epson color matrix)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Epson color daisy wheel)</td>
</tr>
</tbody>
</table>
Abacus Software

**************
link A6, #16
move.l #300, $29F6
clr.w D0
move.w D0, $4A6
move.w D0, $4692
move.w D0, -2(A6)
bra $5B58
move.l $3E2A, A0
move.w -2(A6), A1
add.i A1, A0
clr.b (A0)
clr.w (A7)
clr.w -(A7)
clr.w -(A7)
move.w -2(A6), -(A7)
clr.l -(A7)
clr.l -(A7)
jsr $628C
add.l $8E, A7
move.w D0, -(A7)
move.w -2(A6), A0
add.l A0, A0
move.w (A7)+, (A0)
bne $5B54
addq.w $1, $4A6
or.l $3, $4C2
addq.w $1, -2(A6)
cmp.w $2, -(A6)
blt $5B0A
unlk A6

hdv init
maxacctime to 300*20 ms
nflops
Start with drive A
Address of the dsb
Drive number

Drive number

flopinit
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
getbpb, get BIOS parameter block

Zero

getdsb

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
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>005BC2</td>
<td>6C16</td>
<td>bge</td>
<td>$5BDA</td>
</tr>
<tr>
<td>005BC4</td>
<td>3EA0008</td>
<td>move.w</td>
<td>8(A6), (A7)</td>
</tr>
<tr>
<td>005BC6</td>
<td>202EFF4</td>
<td>move.l</td>
<td>-12(A6), D0</td>
</tr>
<tr>
<td>005BC8</td>
<td>3F00</td>
<td>move.w</td>
<td>D0, -12(A7)</td>
</tr>
<tr>
<td>005BCE</td>
<td>4EB9000055C</td>
<td>jsr</td>
<td>$555C</td>
</tr>
<tr>
<td>005BD0</td>
<td>549F</td>
<td>addq.l</td>
<td>$2, A7</td>
</tr>
<tr>
<td>005BD6</td>
<td>2D40FF4</td>
<td>move.l</td>
<td>D0, -12(A6)</td>
</tr>
<tr>
<td>005BDA</td>
<td>202EFF4</td>
<td>move.l</td>
<td>-12(A6), D0</td>
</tr>
<tr>
<td>005BDE</td>
<td>B0BC00010000</td>
<td>cmp.l</td>
<td>$10000, D0</td>
</tr>
<tr>
<td>005BD4</td>
<td>67B0</td>
<td>beq</td>
<td>$5B96</td>
</tr>
<tr>
<td>005BDE</td>
<td>4AAEFFF4</td>
<td>tst.l</td>
<td>-12(A6)</td>
</tr>
<tr>
<td>005BFA</td>
<td>6C06</td>
<td>bge</td>
<td>$5BF2</td>
</tr>
<tr>
<td>005BFC</td>
<td>4280</td>
<td>cir.l</td>
<td>D0</td>
</tr>
<tr>
<td>005BEE</td>
<td>60000124</td>
<td>bra</td>
<td>$5D14</td>
</tr>
<tr>
<td>005BF2</td>
<td>2EBC00012C7</td>
<td>move.l</td>
<td>$512C7, (A7)</td>
</tr>
<tr>
<td>005BF4</td>
<td>6100CD6C</td>
<td>bsr</td>
<td>$6266</td>
</tr>
<tr>
<td>005BFC</td>
<td>3E00</td>
<td>move.w</td>
<td>D0, D7</td>
</tr>
<tr>
<td>005BFE</td>
<td>670E</td>
<td>beq</td>
<td>$5C0E</td>
</tr>
<tr>
<td>005C00</td>
<td>1C39000012C9</td>
<td>move.b</td>
<td>$12C9, D6</td>
</tr>
<tr>
<td>005C06</td>
<td>4886</td>
<td>ext.w</td>
<td>D6</td>
</tr>
<tr>
<td>005C09</td>
<td>CC7C00FF</td>
<td>and.w</td>
<td>$8FF, D6</td>
</tr>
<tr>
<td>005C0C</td>
<td>6606</td>
<td>bne</td>
<td>$5C14</td>
</tr>
<tr>
<td>005C0E</td>
<td>4280</td>
<td>cir.l</td>
<td>D0</td>
</tr>
<tr>
<td>005C10</td>
<td>60000102</td>
<td>bra</td>
<td>$5D14</td>
</tr>
<tr>
<td>005C14</td>
<td>3887</td>
<td>move.w</td>
<td>D7, (A4)</td>
</tr>
<tr>
<td>005C16</td>
<td>39460002</td>
<td>move.w</td>
<td>D6, 2(A4)</td>
</tr>
<tr>
<td>005C1A</td>
<td>2EBC00012D2</td>
<td>move.l</td>
<td>$512D2, (A7)</td>
</tr>
<tr>
<td>005C20</td>
<td>61000664</td>
<td>bsr</td>
<td>$6266</td>
</tr>
<tr>
<td>005C24</td>
<td>39400008</td>
<td>move.w</td>
<td>D0, 8(A4)</td>
</tr>
<tr>
<td>005C28</td>
<td>302C0008</td>
<td>move.w</td>
<td>8(A4), D0</td>
</tr>
<tr>
<td>005C2C</td>
<td>5240</td>
<td>addq.w</td>
<td>#1, D0</td>
</tr>
</tbody>
</table>

**Notes:**
- Drive number
- Error code
- Critical error handler
- Correct stack pointer
- Save error code
- Read boot sector again
- ok ?
- Buffer+11, bytes per sector
- u2i, convert 8086 integer to 68000 int
- Save bytes per sector
- Buffer+13, sectors per cluster
- resize
- clsiz
- Buffer+22, sectors per FAT
- u2i, convert 8086 integer to 68000 int
- faiz
- plus 1
005C2E 3940000A
005C32 3014
005C34 C1EC0002
005C38 39400004
005C3C 2EBC000012CD
005C42 61000622
005C46 6B40
005C48 48CO
005C4A 81D4
005C4C 39400006
005C50 302C006A
005C54 00C0006
005C58 00C0008
005C5C 394000CC
005C60 2EBC000012CF
005C66 610005FE
005C6A 90C0000C
005C6E 48CO
005C70 81EC0002
005C74 3940000E
005C78 2EBC000012D6
005C7E 610005E6
005C82 3B400014
005C86 2EBC000012D4
005C8C 610005DB
005C90 3B400018
005C94 302D0014
005C98 C1ED0018
005C9C 3B400016
005CA0 2EBC000012DB
005CA6 610005E8
005CAA 3B40001A
move.w D0,16(A4)
move.w (A4),D0
mul.s.w 2(A4),D0
move.w D0,4(A4)
move.l #$12CD,(A7)
bsr $6266
asl.w #$5,D0
ext.l D0
divs.w (A4),D0
move.w D0,6(A4)
move.w 10(A4),D0
add.w 6(A4),D0
add.w 8(A4),D0
move.w D0,12(A4)
move.l #$12CF,(A7)
bsr $6266
sub.w 12(A4),D0
ext.l D0
div.s.w 2(A4),D0
move.w D0,14(A4)
move.l #$12D6,(A7)
bsr $6266
move.w D0,20(A5)
move.l #$12D4,(A7)
bsr $6266
move.w D0,24(A5)
move.w 20(A5),D0
mul.s.w 24(A5),D0
move.w D0,22(A5)
move.l #$12D8,(A7)
bsr $6266
move.w D0,26(A5)

fatrec
resize
Times claiz
Yields claizb
Buffer+17, number of directory entries
u21, convert 8086 integer to 68000 int
Times 32
Divided by resiz2
Yields rlen
fatrec
Plus rlen
Plus fslz
Yields datrec
Buffer+19, number of sectors
u21, convert 8086 integer to 68000 int
Minus datrec
Divided by claiz
Yields numcl
Buffer+26, number of sides
u21, convert 8086 integer to 68000 int
dnsides
Buffer+24, sector per track
dnsides
Buffer+28, number of hidden sectors
dnsides
Times dspt
Yields depc
Buffer+28, number of hidden sectors
dnsides
Times dspt
Yields depc
move.l #$12CF,(A7)
bsr $6266
ext.l D0
divs.w 22,(A5),D0
move.w D0,18,(A5)
cir.w D7
bra $5CDC
move.l A5,A0
move.w D7,A1
addl A1,A0
move.w D7,A1
addl #$123C,A1
move.b 8,(A1),28,(A0)
addq.w #1,D7
cmp.w #3,D7
bit $5CC6
move.l #$676,A0
move.w 8,(A6),A1
addl A1,A0
move.l #$674,A1
move.w 8,(A6),A2
addl A2,A1
move.b (A1),(A0)
beq $5D02
moveq.l #1,D0
bra $5D04
cir.w D0
move.l #$322A,A1
move.w 8,(A6),A2
addl A2,A1
move.b D0,(A1)
move.l A5,D0
Buffer+19, number of sectors on disk
u2i, convert 8086 integer to 68000 int
Divided by dspc
Yields dtracks
Counter to zero
Jump to end of loop
Buffer pointer
Loop counter
Plus BPB address
Loop counter
Plus buffer address
Copy byte of the serial number
Next byte
Three bytes already ?
No
cdev

wpstatus

Disk status uncertain

Status certain

Address of BPB as result
005D14 4A9F
tst.l (A7)+
movem.l (A7)+,D6-D7/A4-A5
005D16 4CDF3CC0
unlk A6
005D1A 4E5E
rts
005D1C 4E75

*******************************************************************************
005D1E 4E56000
link A6,10
005D22 48E70304
movem.l D6-D7/A5,-(A7)
005D26 0CE600000000
cmp.w $2,8,(A6)
005D2C 6D04
blt $5D32
005D2E 75F1
moveq.l #15,D0
005D30 604C
bra $5D7E
005D32 3E2E0008
move.w B,(A6),D7
005D36 3A47
move.w D7,A5
005D38 DBFC00000E2A
add.l $3E2A,A5
005D3E UC150002
cmp.b #2,(A5)
005D42 6604
bne $5D46
005D44 7002
moveq.l #2,D0
005D46 6036
bra $5D7E
005D48 207CC0000076
move.l #$676,A0
005D4E 4A307000
tst.l (A0,D7.w)
005D52 6704
beq $5D58
005D54 1ABC001
move.b $1,(A5)
005D58 2039000004BA
move.l $4BA,D0
005D5E 3247
move.w D7,A1
005D60 3C9
add.l A1,A1
005D62 3C9
add.l A1,A1
005D64 D3FC00000678
add.l #$678,A1
005D6A 2211
move.l (A1),D1
005D6C 9061
sub.l D1,D0
005D6E B0B9000025F6
cmp.l $25F6,D0
005D74 6C04
bge $5D7A
005D76 4240       clr.w DO
005D78 6004       bra $5D7E
005D7A 1015       move.b (A5),D0
005D7C 4880       ext.w DO
005D7E 4A9F       tst.l (A7)+
005D80 4CDF2080   movem.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)
005DAB 6C0000CE   bge $5E7B
005DAC 3006       move.w D6,D0
005DAE EB40       asl.w #5,D0
005DB0 48C0       ext.l D0
005DB2 2A40       move.l D0,A5
005DB4 DBFC0000E3E add.l #$3E3E,A5
005DBA 3E86       move.w D6,(A7)
005DBC 6100FF60   bar $5D1E
005DC0 320C       move.w D0,D7
005DC2 BE7C0002   cmp.w #2,D7
005DC6 660A       bne $5DD2
005DC8 70F2       moveq.l #-14,D0
005DCA 600000DE   bra $5EAA

***************
005DDB 4E56FFFC  rwabs, read/write sector(s)
005DD7 48E70F04  Save registers
005DD9 0C6E00020012 Drive number
005DDA 6D06      < 2 ?
005DDC 3006      No, unknown device
005DDD 3DE20008  Error branch
005DE0 6C0000CE  Save drive number
005DE2 48C0      rwflag
005DE4 DBFC0000E3E Test media change
005DEA 3E86      Save status
005DEC 6100FF60  Disk changed ?
005DF0 320C      No
005DF2 BE7C0002  Media change error
005DF6 660A      Error branch
005DF8 70F2
bra $5E78
cmp.w $1,D7
bne $5E78
move.w $1,(A7)
clr.w -(A7)
clr.w -(A7)
move.w $1,-(A7)
move.w D6,-(A7)
clr.l -(A7)
move.l $5126C,-(A7)
jsr $62D2
add.l $510,A7
move.l D0,-4(A6)
tst.l -4(A6)
beq $5E1A
move.w D6,(A7)
move.w -4(A6),D0
move.w D0,-(A7)
jsr $559C
addl $2,A7
move.l D0,-4(A6)
move.l -4(A6),D0
cmp.l $510000,DO
beq $5DDA
tst.l -4(A6)
beq $5E34
move.l -4(A6),D0
bra $5EAA
clr.w D7
bra $5E54

Disk possibly changed?
No
Read a sector (Boot sector)
Side 0
Track 0
Sector 1
Drive number
Filler
Sector buffer
flopdrd
Correct stack pointer
Save error number
And test
OK?

Error
Pass an critical error handler
Correct stack pointer

Read again
Error number
OK?
Error number
Error branch
Clear media change status
move.l #$12BC,A0
move.b $4(A0),D7,w,D0
ext.w D0
move.b $2(A5,D7.w),D1
ext.w D1
cmp.w D1,D0
beq $5E52
moveq.l #-14,D0
bra $5EAA

addq.w #1,D7
cmp.w #3,D7
blt $5E38
move.w D6,A0
add.l #$5676,A0
move.w D6,A1
add.l #$5674,A1
move.b (A1), (A0)
bra $5E76
move.w D6,A0
add.l #$32E2A,A0
cir.b (A0)
tst.w $4A6
bra $5E84
moveq.l #2,-D0
bra $5EAA

cmp.w #1,0 (A6)
ble $5E90
subq.w #2,0 (A6)
move.w 1c(A6), (A7)
move.w D6,-(A7)

Address of the sector buffer
Serial number

Compare
With previous value
Ok ?
Media change
Error branch

Next byte of the serial number
All 3 bytes tested ?
No
Drive number
wpstatus

nflops

Drive not ready
Error branch

Drive number

Drive number
count

Drive number
005E96 3F2E0010  move.w 16(A6),-(A7)  recno
005E99 2F2E000A  move.l 10(A6),-(A7)  buffer
005E9E 3F2E0008  move.w 9(A6),-(A7)  rwflag
005EA2 6110  bsr $5EB4  floprw
005EA4 DFFC000000A  add.l #$5A,A7  Correct stack pointer
005EAA 4A9F  tst.l (A7)+  Restore registers
005EAC 4CDE20E0  movem.l (A7)+,D5-D7/A5
005EB0 4E5E  unlink A6
005EB2 4E75  rts

***************************************************************************
005EB4 4E56FFFA  link  A6,#-6  floprw, read/write sectors
005EB8 4E73E004  movem.l D2-D7/A5,-(A7)  Save registers
005EC0 30250010  move.w 16(A6),D0  Drive number
005EC2 4B40  asl.w #5,D0  Times 32
005EC4 2A40  ext.l D0
005EC6 DBFC00003E3E  add.l #$3E3E,A5  Plus base address BPB
005EC8 0820000000D  bstat #0,13(A6)  Buffer address not even ?
005EC2 6604  bne $5ED8  Yes
005ED4 4240  clr.w D0  Clear oddflag
005ED6 6002  bra $5E0A  Set oddflag
005ED8 7001  moveq.l #1,D0  And save
005EDA 3D40FFFE  move.w D0,-2(A6)  dspc set ?
005EDE 4A600016  tst.w 22(A5)  Yes
005EE2 660A  bne $5EEE  Else take 9
005EE4 7009  moveq.l #9,D0  As cysz
005EE6 3B400016  move.w D0,22(A5)  And dspc
<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>005F02 4A6EFFFE</td>
<td>tst.w -2(A6)</td>
<td>oddflag set?</td>
</tr>
<tr>
<td></td>
<td>beq $5F00</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>move.l $12BC,D0</td>
<td>Sector buffer</td>
</tr>
<tr>
<td></td>
<td>bra $5F04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move.l 10(A6),D0</td>
<td>Get buffer address</td>
</tr>
<tr>
<td></td>
<td>move.l D0,-6(A6)</td>
<td>And save</td>
</tr>
<tr>
<td></td>
<td>move.w 14(A6),D6</td>
<td>reco, logical sector number</td>
</tr>
<tr>
<td></td>
<td>ext.l D6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>divs.w 22(A5),D6</td>
<td>Divided by dpc yields track number</td>
</tr>
<tr>
<td></td>
<td>move.w 14(A6),D4</td>
<td>reco, logical sector number</td>
</tr>
<tr>
<td></td>
<td>ext.l D4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>divs.w 22(A5),D4</td>
<td>Divided by dpc, sector per track</td>
</tr>
<tr>
<td></td>
<td>swap D4</td>
<td>Remainder of division as sector number</td>
</tr>
<tr>
<td></td>
<td>cmp.w 24(A5),D4</td>
<td>Compare with dpt</td>
</tr>
<tr>
<td></td>
<td>bge $5F28</td>
<td>Greater than or equal?</td>
</tr>
<tr>
<td></td>
<td>clr.w D5</td>
<td>Side 0</td>
</tr>
<tr>
<td></td>
<td>bra $5F2E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>moveq.l #1,D5</td>
<td>Side 1</td>
</tr>
<tr>
<td></td>
<td>sub.w 24(A5),D4</td>
<td>Subtract dpt</td>
</tr>
<tr>
<td></td>
<td>tst.w -2(A6)</td>
<td>oddflag set?</td>
</tr>
<tr>
<td></td>
<td>beq $5F38</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>moveq.l #1,D3</td>
<td>Set counter to 1</td>
</tr>
<tr>
<td></td>
<td>bra $5F50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move.w 24(A5),D0</td>
<td>dpt</td>
</tr>
<tr>
<td></td>
<td>sub.w D4,D0</td>
<td>Minus Sector number</td>
</tr>
<tr>
<td></td>
<td>cmp.w 18(A6),D0</td>
<td>Compare with number of sectors</td>
</tr>
<tr>
<td></td>
<td>bge $5F4C</td>
<td>Greater or equal?</td>
</tr>
<tr>
<td></td>
<td>move.w 24(A5),D3</td>
<td>dpt</td>
</tr>
<tr>
<td></td>
<td>sub.w D4,D3</td>
<td>Minus sector number equals counter</td>
</tr>
<tr>
<td></td>
<td>bra $5F50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move.w 18(A6),D3</td>
<td>Number of sectors as counter</td>
</tr>
</tbody>
</table>
Increment sector # (1st number is 1)  
Read
Buffer pointer
Equals buffer address
Yes
Source address
Destination address
fastcopy, copy a sector
Correct stack pointer
Number of sectors
Side
Track
Sector
Drive
Filler
Sector buffer
flop wr, read sector
Correct stack pointer
Error code
Ok?
No
verify, verify required?
No
Number of sectors
Side
Track
Sector
Drive
Filler
Sector buffer
flop wr, verify sector
add.l #$10, A7
move.l D0, D7
tst.l D7
bne $5FD6
move.l #$12BC,(A7)
ber $6266
tst.w D0
beq $5FD6
moveq.l #16, D7
bra $6012
move.w D3, {A7}
move.w D5, -(A7)
move.w D6, -(A7)
move.w D4, -(A7)
move.w 16(A6), -(A7)
clr.l -(A7)
move.l -6(A6), -(A7)
jsr $6202
add.l #$10, A7
move.l D0, D7
move.l -6(A6), D0
com.l 10(A6), D0
beq $6012
move.l 10(A6), (A7)
move.l -6(A6), -(A7)
jsr $5910
addq.l #4, A7
tst.l D7
bge $6028
move.w 16(A6), (A7)
move.l D7, D0

Correct stack pointer
Error code
Ok?
No
Sector buffer
u21, convert 8056 integer to 68000 int
Bad sector list
Sectors OK?
Bad sectors

Number of sectors
Side
Track
Sector
Drive
Filler
Sector buffer
flopdr, read sector
Correct stack pointer
Save error code
User buffer
Equals desired buffer?
Yes
Source address
Destination
fastcopy, copy sector
Correct stack pointer
Test error code
Ok?
Drive number
Error code
On stack
Critical error handler
Correct stack pointer
Get error code
Attempt again?
Yes
Error code
OK
Error code as result

Sector counter
Times 512
Increment buffer address
Logical sector number plus counter
Decrement number of sectors to process
Sectors yet to read/write?
Yes
OK
Restore registers

random, generate random number

Last random number
Not zero?
200 Hz counter
Abacus Software

-OrmCQ.4-1

QO CM

006076 E3A0
006078 8089000004BA
00607E 23C0000025FA
006084 2F33B4AE62D
006089 2F339000025FA
006090 4E29000094FA
006096 508F
00609A 23C0000025FA
0060A0 2039000025FA
0060A6 E080
0060AB C0C000FFFF
0060AE 4252
0060B0 4275

asl.l D1, D0
or.l $4BA, D0
move1 D0, $25FA
move1 #BB40E62D, -(A7)
jsr $94FA
addq.l #8, A7
addq.l #1, D0
move1 D0, $25FA
move1 $25FA, D0
asr.l #0, D0
and.l #FFFFF, D0
unlink A6
rts

*************************************************************************
0060B2 45560000
0060B6 48F70300
0060BA 4E9900005932
0060C0 4A7900000A6
0060C6 6704
0060CB 7001
0060CA 6002
0060CE 7002
0060CE 3200
0060D0 4A79000034A6
0060D6 6744
0060DB 6C7900002000446
0060E0 6C3A
0060E2 3EBCCD01
0060E6 4267
0060E8 4267

link A6, #0
movem1 D6-D7, -(A7)
jsr $5932
tst.w $4A6
beq $60CC
moveq.l #1, D0
bra $60CE
moveq.l #2, D0
move.w D0, D7
tst.w $4A6
beq $611C
cmp.w #2, 0446
bge $611C
move.w #1, -(A7)
cir.w -(A7)
cir.w -(A7)

Bits 0-15 to 16-31
Plus 200 Hz counter
Use as start value
3114S92621
Last random value
32 * 32 bit multiplication
Correct stack pointer
Result plus 1
As new start value

Bits 31-8 to 23-0
24-bit random number as result

bootload, load boot sector
Save registers
hsv init
nflps
No drive connected ?
'couldn't load'

'no drive'
Save errors
nflps

bootdev
No diskette ?
One sector
Side 0
Track 0
move.w $1,-(A7)  
move.w $465,-(A7)  
cir.l -(A7)  
move.l $582BC,-(A7)  
jer $62D2  
add.l $510,A7  
tst.l D0  
bne $6110  
cir.w D7  
bra $611C  
tst.b $674  
bne $611C  
moveq.l $3,D0  
bra $6140

Sector 1  
bootdev as drive number  
Filler  
Sector buffer  
floprd, read sector  
Correct stack pointer  
Error ?  
No  

getstatus  
'unreadable'

Get old error code

move.w $4100,(A7)  
move.l $412BC,-(A7)  
bsr $6236  
addr.l #4,A7  
cmp.w $1234,D0  
bne $613E  
cir.w D0  
bra $6140  
moveq.l $4,D0  
tst.l (A7)+  
movem.l (A7)+,D7

$100 words  
Sector buffer  
Calculate checksum  
Correct stack pointer  
Compare with checksum for boot sector  
Not equal ?  
OK  

'not valid boot sector'  
Restore registers
proto bt, generate boot sector

Save registers
Test execflag
Maintain executability
$100 words
Address of the sector buffer
Calculate checksum
Correct stack pointer
Equals checksum for boot sector?
Yes
Not executable

Executable
execflag 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
Abacus Software

Atari ST Internals

0061A0 D3E00008
0061A4 13400008
0061A8 20E00000
0061AC 0080
0061AE 2D40000C
0061B2 5247
0061B4 BE7C00C3
0061B8 6D3A
0061BA 4E6E0010
0061BE 6D28
0061C0 3C2E0010
0061C4 CDFC0013
0061C8 4247
0061CA 6016
0061CC 3047
0061CE D1E00008
0061D2 3246
0061D4 D3FCC0016D0C
0061DA 1151000B
0061DE 5246
0061E0 5247
0061E2 BE7C0013
0061E6 6DE4
0061E8 62E0FA
0061EC 2D6E0008FFC
0061F2 600E
0061F4 2D6EFFFC
0061F8 3010
0061FA D1E0FFFA
0061FE 54AFFFA

add.l B(A6), A1
move.b D0, 8(A1)
move.l 12(A6), D0
asr.l #8, D0
move.l D6, 12(A6)
addq.w #1, D7
cmp.w #$3, D7
bit $6194
tst.w 16(A6)
bit $61E8
move.w 16(A6), D6
mul.s.w #$13, D6
clr.w D7
bra $61E2
move.w D7, A0
add.l 8(A6), A0
move.w D6, A1
add.l #$16D00, A1
move.b (A1), 11(A0)
addq.w #1, D6
addq.w #1, D7
cmp.w #$13, D7
bit $61CC
clr.w -6(A6)
move.l 8(A6), -4(A6)
bra $6202
move.l -4(A6), A0
move.w (A0), D0
add.w D0, -5(A6)
addq.w #2, -4(A6)

Plus base address
Write byte of serial number in buffer
Random number
In order to shift 8 bits right
And save new value
Increment counter
Three bytes copied already?
No
Diskette size
Negative, don't change
Diskette size
Times 19 equals pointer to prototype BPB
Clear counter

Counter
Plus address of the buffer
Address of the prototype BPB
Copy BPB
Increment counter
Already 19?
No
Buffer address
Get word from buffer
Sum for checksum generation
Next word
move.l 8(A6), D0
add.l #$1FE, D0
cmp.l -4(A6), D0
bhi 561F4
move.w #$1234, D0
sub.w -6(A6), D0
move.l -4(A6), A1
move.w D0, (A1)
tst.w 18(A6)
bne 5622C
move.l -4(A6), A0
addq.w #1, (A0)
tst.l (A7)+
movem.l (A7)+, D6-D7/A5
unlock A6
rts

***************
link A6, #0
movem.l D6-D7,-(A7)
clr.w D7
bra 56245
move.l 8(A6), A0
move.w (A0), D0
add.w D0, D7
addq.l #2, 8(A6)
move.w 12(A6), D0
subq.w #1, 12(A6)
tst.w D0
bne 56242
move.w D7, D0
tst.l (A7)+
Buffer address
Plus $1FE
Last word already
Checksum for boot sector
Subtract from previous value
Checksum in buffer
execflg
Boot sector executable?
Increment checksum, not executable
Restore registers
Calculate checksum
Save registers
Clear sum
Address of the buffer
Get word
And sum
Manager to next word
Number of words
Minus 1
All words added already?
No
Result to D0
00625E 4C9FO080  movem.l (A7)+,D7
006262 4E5E  unlk A6
006264 4E75  rts

*****************************************************************************
006266 4E5FFFC  link A6,$-4
00626A 206E008  move.l 0(A6),AO
00626E 10280001  move.b 1(A0),DO
006272 4880  ext.w DO
006274 07C00FF  and.w #7F,DO
006278 E140  asl.w #8,DO
00627A 226E0008  move.l 8(A6),A1
00627E 1211  move.b (A1),D1
006280 4881  ext.w D1
006282 C27C00FF  and.w #3F,D1
006286 6041  or.w D1,DO
006288 4E5E  unlk A6
00628A 4E75  rts

*****************************************************************************
00628C 43F900006C8  lea $8C8,Al
006292 4A8000C  test.w 12(A7)
006296 6706  beq $629E
006298 43F900006CC  lea $6CC,Al
00629E 3379000044000062  move.w $500,2(AL)
0062A6 70FF  moveq.l #+1,DO
0062A8 42690000  clr.w 0(AL)
0062AC 610004BA  beq $6768
0062B0 61000696  bsr $6948
0062B4 337CF000000  move.w #8F000,0(AL)
0062BA 61000618  bsr $88D4
0062BE 7E06  moveq.l #6,D7

Restore registers

u21, convert 8086 number to 68000 number

Address of the number
Get high byte
Extend to word
Isolate bits 0-7
Shift in position 8-15
Address of the number
Get low byte
Extend to word
Isolate bits 0-7
Combine with bits 8-15

flopini, 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
0062C4 6608
0062C6 6100060C
0062CA 67000542
0062CE 60000530

-------------------
bsr  $6862
bne  $62C2
bsr  $68D4
beq  $689E
bra  $68D0

0062D2 61000712
0062D6 70F5
0062DA 610006BE
0062DC 6100066A
0062E0 610005CC
0062E4 66000090
0062E8 33FC0000006A2
0062F0 3CBC0090
0062F4 3CBC0190
0062F8 3CBC0090
0062FC 33ED068C0000004
006304 3CB00820
006308 3E300090
00630C 610006B6
006310 2E3000000000
006316 246D0092
00631A 08390005000FA01
006322 6734
006324 5387
006326 6724
006328 1879008609069D
006330 187900850B069E
006338 187900860D069F
006340 B5ED069C
006344 6ED4

Restorable
flopo, 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
accout to dscktl, 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
dmanid
dmalow
Current DMA address equal edma?
No, continue to wait
006346 61005E5E
00634A 600C
00634C 3B7C000E06A2
006352 6100050A
006356 601E
006358 3C8C0000
00635C 3016
00635E 08000000
006362 6712
006366 3CBC0000
006368 6100666E
00636C 0300018
006370 6700019C
006374 6118
006376 0C6D00010672
00637C 6604
00637E 610004FA
006382 536D0672
006386 6A00FF54
00638A 60000474

bsr $692E
bra $6338
move.w $-2,$E2A2(A5)
bsr $692E
bra $6376
move.w $890,(A6)
move.w (A6),D0
btst #0,D0
beq $6376
move.w $80,(A6)
bsr $69D8
and.b $818,D0
beq $680E
bsr $638E
cmp.w $1,$872(A5)
bne $6382
bsr $687A
subq.w $1,$872(A5)
bl$ $62DC
bra $6890

Reset, end transfer
currerr to timeout
Reset, end transfer
Start next attempt
Select DMA status register
Read status
DMA error ?
Yes, try again
Select 1772 status register
diskctl, read register
RNF, isolate checksum und lost data
flopok, no error
erbits, determine error number
retrycnt to second attempt ?
No
resee, home and resee
retrycnt, decrement attempt counter
Another attempt ?
No, flopfail

erbits, 1772 status in error number
Write protect ?

Yes
Record not found ?

Yes
CRC error ?

Yes
deferror, take default error
0063AA 3B4106A2
0063AE 4E75
move, w D1, $6A2 (A5) 

*****************************************************
0063B0 61000640
0063B4 70F6
0063B8 610003B0
0063BA 302D0688
0063BE 5340
0063C0 806D0686
0063C4 806D068A
0063C8 6606
0063CA 70G2
0063CC 6100056C
0063D0 61000576
0063D4 610004D8
0063D8 660007E
0063DC 3B7CFFFD06A2
0063E2 3CBC0190
0063E6 3CBC0090
0063EA 3CBC0190
0063EE 3E3C0001
0063F2 610005D0
0063F6 3C3C0180
0063FA 3E3C00A0
0063F2 610005C4
006402 2E3C00040000
006408 0819000500FFFA01
006410 670A
006412 5387
006414 66F2
00641E 61000516

As currerr
flopwr, write sector(s) on disk
Change, test for disk change
Default error to write error
floplock, set parameters
sect, 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 DNA status

Data direction to WRITE
Sector count register
wdiskcl, D7 to 1772
Selects 1772
Write Sector
wdiskcl, D7 to 1772
Timeout counter
npf, qplp, 1772 done?
Yes
Decrement timeout counter
Not timed-out yet?
reset, end transfer
00641A 6034  bra  $6450
00641C 3C10  move.w #$180,(A6)
006420 610055B6  bsr  $69D8
006424 6100FF68  bsr  $638E
006428 08000006  btest #6,0
00642C 660003D2  bne  $6800
006430 C03C005C  and.b $55C,0
                             Select 1772 status register
                             rdiskctl, read 1772 registers
                             errbits, calculate error number
                             Write protect?
                             flopfail, no further attempt
                             Write protect, RNF, checksum and lost data

006434 661A  bne  $6450
006436 526D0688  addq.w $1, 6608(A5)
006438 06ADCC0020068E  addl $512, $668E(A5)
006442 53ED0C68C  subq.w $1, 6608C(A5)
006446 670003C6  beq  $680E
00644A 61000524  bsr  $6970
00644E 608C  bra  $63DC
006450 0C6D0010672  cmp.w $1, 6672(A5)
006454 6604  bne  $645C
006458 61000420  bsr  $687A
00645C 53ED0672  subq.w $1, 6672(A5)
006460 6AD0FF6E  bpl  $63D0
006464 6000039A  bra  $6800
                             Error, try again
                             csect, increment sector number
                             cdma, DMA address to next sector
                             count, decrement number of sectors
                             flopok, all sectors, then done
                             select1, sector number and DMA pointer
                             Write next sector without seek
                             retrycnt, second attempt?
                             No
                             recovery, home and seek
                             retrycnt, decrement attempt counter
                             Another attempt?
                             flopfail, error
                             flopfmt, format track
                             Magic number?
                             No, flopfail
                             Change, test for disk change
                             Default error number
                             floplock, set parameters
                             select, select drive and side
                             spt, sectors per track
                             intervl, interleave factor
                             virgin, sector data for formatting
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 count 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
deferror, 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
$FE
wmult, write in buffer
$FE, address mark
00650E 14F9000000687
006514 14F900000068B
00651A 14C4
00651C 14FC0002
006520 14FC00F7
006524 323C0015
006528 103C004E
00652C 610000CC
006530 323C000B
006534 4200
006536 610000C2
00653A 323C0002
00653E 103C00F5
006542 610000B6
006546 14FC00F7
00654A 323C00FF
00654E 245D00F9
006552 14ED006B
006556 51C9FFFE
00655A 14FC00F7
00655E 323C0027
006562 103C004E
006566 61000092
00656A D86D0698
00656E B86D0696
006572 6F80
006574 5243
006576 566D0698
00657A 6F00FFFF
00657E 323C00578
006582 103C004E
006586 6172

move.b $687,(A2)+
move.b $68B,(A2)+
move.b $44,(A2)+
move.b #2,(A2)+
move.b #$F7,(A2)+
move.w #$15,D1
move.b #$4E,D0
bsr $65FA
move.w #$3B,D1
clr.b D0
bsr $65FA
move.w #2,D1
move.b #$F5,D0
bsr $65FA
move.b #$FB,(A2)+
move.w #$FF,D1
move.b #$9B,(A5),(A2)+
move.b #$9B,(A5),(A2)+
dbra D1,$654E
move.b #$F7,(A2)+
move.w #$27,D1
move.b #$4E,D0
bsr $65FA
add.w #$9B,(A5),D4
cmp.w #$9B,(A5),D4
ble $64F4
addq.w #1,D3
cmp.w #$9B,(A5),D3
ble $64F2
move.w #$578,D1
move.b #$4E,D0
bsr $65FA

Track
Side
Sector
Sector size (512 bytes)
Write checksum
22 times
$4E
wmult, write in buffer 12 times
0
wmult, write in buffer 3 times
$75
wmult, write in buffer
$FB, data block mark
256 times
virgin, initial data in buffer

Next word
Write checksum
60 times
$4E .
wmult, write in buffer
Add interly, next sector
apt, largest sector number ?
No, next sector
Start sector plus one
interly
Next sector
1401 times (until track end)
$4E
wmult, write in buffer

Atari ST Internals
Abacus Software

Atari ST Internals

move.b $691(A5),$FFFF860D
move.b $690(A5),$FFFF860B
dma low

move.w #$190,(A6)
move.w #$90,(A6)
move.w #$180,(A6)
move.w #$1F,D7

Sector counter to 31

Select 1772

Format track command

Data direction to WRITE

Sector counter to 31

Select 1772

Timeout counter

mfp gpip, 1772 done ?

Yes

Decrement timeout counter

Not yet timed-out ?

reset, terminate

Clear Z-bit, error

Write data in buffer

Next byte
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Op Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006602</td>
<td>bsr     $69F2</td>
<td></td>
<td>fsr, verify sector(s)</td>
</tr>
<tr>
<td>006605</td>
<td>moveq.l $#$-11,D0</td>
<td></td>
<td>Change, test for disk change</td>
</tr>
<tr>
<td>006608</td>
<td>bsr     $676B</td>
<td></td>
<td>'read error', as default error</td>
</tr>
<tr>
<td>00660C</td>
<td>bsr     $6948</td>
<td></td>
<td>floplock, set parameters</td>
</tr>
<tr>
<td>006610</td>
<td>bsr     $60AE</td>
<td></td>
<td>Select</td>
</tr>
<tr>
<td>006614</td>
<td>bne     $68C0</td>
<td></td>
<td>go2track, search for track</td>
</tr>
<tr>
<td>006618</td>
<td>bra     $661E</td>
<td></td>
<td>flopfail, error</td>
</tr>
<tr>
<td>00661A</td>
<td>bra     $680E</td>
<td></td>
<td>verifyl, verify sectors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>flopok, done</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Op Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00661E</td>
<td>move.w $#$-11,$6A0(A5)</td>
<td></td>
<td>defererror to 'read error'</td>
</tr>
<tr>
<td>006624</td>
<td>move.l $68E(A5),A2</td>
<td></td>
<td>cdma, DMA buffer for bad sector list</td>
</tr>
<tr>
<td>006628</td>
<td>add.l $#$200,$68E(A5)</td>
<td></td>
<td>cdma to next sector</td>
</tr>
<tr>
<td>006630</td>
<td>move.w $2,$672(A5)</td>
<td></td>
<td>retrycnt, 2 attempts</td>
</tr>
<tr>
<td>00663C</td>
<td>move.w $584,(A6)</td>
<td></td>
<td>Select sector register</td>
</tr>
<tr>
<td>00663A</td>
<td>move.w $608(A5),D7</td>
<td></td>
<td>csect, sector number</td>
</tr>
<tr>
<td>00663E</td>
<td>bsr     $69C4</td>
<td></td>
<td>wdiskctl, to disk controller</td>
</tr>
<tr>
<td>006642</td>
<td>move.b $691(A5),$FFFFF860D</td>
<td></td>
<td>dma low</td>
</tr>
<tr>
<td>00664A</td>
<td>move.b $690(A5),$FFFFF860D</td>
<td></td>
<td>dma mid</td>
</tr>
<tr>
<td>006652</td>
<td>move.b $68F(A5),$FFFFF860D</td>
<td></td>
<td>dma high</td>
</tr>
<tr>
<td>00665A</td>
<td>move.w $590,(A6)</td>
<td></td>
<td>Clear DMA status</td>
</tr>
<tr>
<td>00665E</td>
<td>move.w $590,(A6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006662</td>
<td>move.w $590,(A6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006666</td>
<td>move.w $1,D7</td>
<td></td>
<td>Data direction to READ</td>
</tr>
<tr>
<td>00666A</td>
<td>bsr     $69C4</td>
<td></td>
<td>Sector counter to 1</td>
</tr>
<tr>
<td>00666E</td>
<td>move.w $580,(A6)</td>
<td></td>
<td>wdiskctl</td>
</tr>
<tr>
<td>006672</td>
<td>move.w $580,D7</td>
<td></td>
<td>Select 1772 command register</td>
</tr>
<tr>
<td>006676</td>
<td>move.w $580,D7</td>
<td></td>
<td>Read sector command</td>
</tr>
<tr>
<td>00667A</td>
<td>bsr     $69C4</td>
<td></td>
<td>wdiskctl</td>
</tr>
<tr>
<td>00667D</td>
<td>move.l $#$40000,D7</td>
<td></td>
<td>Timeout counter</td>
</tr>
</tbody>
</table>
***floppy vertical blank handler***

- **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 protect bit
- And save
- Previous select status
- Recreate
- wpstatus
- Write wplatch
- deslflg, floppies already deselected?
- Yes
- Read 1772 status register

```
sub.l  A5,A5
lea $FFFF8606,A6
st $680(A5)
tst.w $43E(A5)
ble $6766
move.l $466,D0
move.b D0,D1
and.b #7,D1
bne $673C
move.w #$30,(A6)
lsr.b #3,D0
and.w #1,D0
lea $674(A5),A0
add.w D0,A0
cmp.w $4A6,D0
bne $671E
cir.w D0
addq.b #1,D0
asl.b #1,D0
eor.b #7,D0
bsr $6994
move.w $FFFF8604,D0
btst #6,D0
sne (A0)
move.b D2,D0
bsr $6994
move.w $674(A5),D0
or.w D0,$676(A5)
tst.w $682(A5)
bne $6762
bsr $69D8
```
Motor on bit set ?
Yes, then don't deselect
Both drives
Deselect
Set deselq
Clear motron flag
floplock
regsave
Clear A5
Address of the floppy register
Set motron flag
deferro
currerr
flock, disable floppy-vbl routine
odma
cde
ceq
iffl
ctrack
eside
count
retrystnt
dsb0
cdev
Drive 0 ?
dsb1
count, number of sectors
Times 512
cdma, start DMA address
0067D2 D1C7
0067D4 0A690000
0067DC 6A20
0067DE 61000069
0067E2 42690000
0067E4 610000EC
0067E6 6712
0067E8 720A
0067E9 6717
0067EA 6606
0067F2 610000EC
0067F8 6706
0067FD 337CDF000000
0067F8 4E7F

add.l D7,A0
move.l A0,$692(A5)
tst.w 0(A1)
ble $67FE
bsr $6968
clr.w 0(A1)
bsr $6BP4
beg $67FE
moveq.l #10,D7
bsr $6862
bne $67P9
bsr $6BD4
beg $67FE
move.w #$FF00,0(A1)

jit ***************************************************

006800 7001
006802 61000226
006806 302D06A2
006808 48CO
00680A 6002

moveq.l #1,D0
bsr $6A2A
move.w $6A2(A5),D0
ext.l DO
bra $6810

******************************************************************************

00680E 4280
006810 2F00
006812 3C8C0086
006814 3E290000
006818 610001A8
00681A 3C3C0010
006822 610000C6
006826 303900000684

c1r.l DO
move.l D0,-(A7)
move.w $886,(A8)
move.w 0(A1),D7
bsr $6BC4
move.w #$10,D6
bsr $692A
move.w $684,D0

******************************************************************************

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

flopfail, error in floppy routine
Media change to 'unsure'
Set currerr

flopop, error-free floppy routine

ok
Select 1772
Get track number
wdiskctl, send to disk controller
seek command
flopms
cdev, drive number
00682C  E548 lsl.w $2, D0
00682E  41F900000678 lea $678, A0
006834  21AD04BA0000 move.l $4BA(A5), D(A0, D0, w)
00683A  C79000100004A6 cmp.w $1, $4A6
006842  6606 bne
006844  216D04BA0004 move.l D(A5), (A0)
00684A  201F move.l
00684C  4CF978F8000006A4 clr.w $43E
006854  4E75 rts

#2, DO
$678, A0
$4BA(A5), D(A0, D0, w)
$684A
$684A
$4BA(A5), (A0)
$4A6
$6A4, D3-D7/A3-A6
$43E
As index
acctim
200 Hz counter as last access time
niflops
#1, $4A6
$6A4
$4BA(A5), 4(AO)
(A7)+, DO
movem.l $6A4, D3-D7/A3-A6
Clear flock, release vbl routine

********************************************

00685C  3E3900000686 move.w $686, D7
006862  33FCFFAA00006A2 move.w #10, $6A2
00686A  3CBC0086 move.w $86, (A6)
00686E  61000156 bsr $69C4
006872  3C3C0010 move.w $10, D6
006876  60000072 bra $68EA

hseek, head to track
ctrack
currerr to 'seek error'
Pass track number
wdiskctl
Seek command
flopcmds

********************************************

00687A  33FCFFAA00006A2 move.w #10, $6A2
006892  6150 bsr $68D4
006894  664C bne $68D2
006896  42690000 clr.w 0(A1)
00689A  3CBC0082 move.w $82, (A6)
00689E  4247 clr.w D7
0068A0  61000132 bsr $69C4
006894  3CBC0086 move.w $86, (A6)
006898  3E3C0005 move.w $5, D7
00689C  61000126 bsr $69C4
0068A0  3C3C0010 move.w $10, D6
0068A4  6164 bsr $68EA

ressek, home and seek
currerr to 'seek error'
Restore
Error ?
Current track to zero
Select track register
Track zero
wdiskctl
Select data register
Track 5
wdiskctl
Seek command
flopcmds
0068A6 662A  bne  $68D2
0068A8 337C  move.w  $5,0(A1)

***************************************************
0068AE 33FC  move.w
0068B6 3CBC  move.w
0068BA 3E2D  move.w
0068BE 6100  tosr
0068C2 7C14  bne
0068C6 660A  bne
0068CE CE3C  and.b
0068D2 CE3C  rts

Error
Track number to 5

go2track, search for track
currerr 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
Sets 0 and 1
OR in command word
Timeout counter
Select 1772 register
rdiskctl

0068D4 4246  clr.w  $68EA
0068D8 660A  bne  $68E8
0068DA 0070  btst  $2,D7
0068DE 0A3C  eor.b  $4,SR
0068E2 6604  bne  $68E9
0068E4 4269  clr.w  0(A1)
0068E8 4275  rts

***************************************************
0068EA 3029  move.w  2(A1),D0
0068EE 03C0  and.b  $3,D0
0068F2 8C00  or.b  D0,D5
0068F4 2E3C  move.l  $40000,D7
0068FA 36BC  move.w  $80,0(A6)
0068FE 6100  bsr  $69D8

N> 0068D6 6112  bsr  $68EA
0068D8 660E  bne  $68E8
0068DA 0970002  btst  $2,D7
0068DE 0A3C004  eor.b  $4,SR
0068E2 6604  bne  $68E9
0068E4 4269000  clr.w  0(A1)
0068E8 4275  rts

******************************************
0068A6 662A  bne  $68D2
0068A8 337C  move.w  $5,0(A1)

***************************************************
0068AE 33FC  move.w
0068B6 3CBC  move.w
0068BA 3E2D  move.w
0068BE 6100  tosr
0068C2 7C14  bne
0068C6 660A  bne
0068CE CE3C  and.b
0068D2 CE3C  rts
Motor on? 
Yes
Else longer timeout
wdiskctl, write command in D6
Decrement timeout counter
Timed-out? 
mfp gpip, 1772 done?
No, wait
rdiskctl?
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 desflag
cdev, drive number
Calculate bit number
cside, side in bit 0
Invert bits for hardware
Set bits in sound chip
Select track register
Get track number
wdskctl
tmpdma, clear bits 24-31
Select sector register
csect, get sector number
wdiskctl
dmamid
dmaskb

setporta, set port A in sound chip
Save status
IPL 7
Read port A
And to D20069AA C23C00F8
Clear bits 0-2
Set new bits
Write result to port A
Restore status

wdiskctl6
Delay loop for disk controller
dskctl
Delay loop for disk controller
wdiskctl
Delay loop for disk controller
dskctl
Delay loop for disk controller
rdiskct7
Delay loop for disk controller
dskctl
Delay loop for disk controller
rdiskct1
Delay loop for disk controller
dskctl
Save status
Save D7
Counter
Delay loop
D7 back
Status back
change, tests disk change
nflops
None or 2 floppies, done
Drive number
Equals diskette number ?
Yes
Drive number
'insert disk'
Critical error handler
Correct stack pointer
wpatch, Status for both drives unsure
Save diskette number
Drive number to zero
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006A2A</td>
<td>lea $3E2A,AO</td>
<td>setmode, set drive change mode</td>
</tr>
<tr>
<td>006A30</td>
<td>move.b D0,-(A7)</td>
<td>Disk mode table</td>
</tr>
<tr>
<td>006A32</td>
<td>move.w $684(A5),DO</td>
<td>Save mode</td>
</tr>
<tr>
<td>006A36</td>
<td>move.b (A7)+,0(A0,DO.w)</td>
<td>cdev, get drive number</td>
</tr>
<tr>
<td>006A3A</td>
<td>rts</td>
<td>Set drive mode</td>
</tr>
</tbody>
</table>

**Disk mode table**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006A3C</td>
<td>dc.b</td>
<td>dskf, disk flags</td>
</tr>
<tr>
<td>006A3D</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A3E</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A3F</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A40</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A41</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A42</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A43</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A44</td>
<td>dc.b</td>
<td></td>
</tr>
<tr>
<td>006A45</td>
<td>dc.b</td>
<td></td>
</tr>
</tbody>
</table>

**jdostime, IXBD format in DOS format**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006A46</td>
<td>lea $0,A5</td>
<td>Clear A5</td>
</tr>
<tr>
<td>006A4C</td>
<td>lea $A43(A5),AO</td>
<td>Pointer to clock-time buffer</td>
</tr>
<tr>
<td>006A50</td>
<td>bsr $6B30</td>
<td>bcdbin</td>
</tr>
<tr>
<td>006A54</td>
<td>sub.b #80,D0</td>
<td>Subtract offset of 80</td>
</tr>
<tr>
<td>006A58</td>
<td>move.b D0,D2</td>
<td>Year</td>
</tr>
<tr>
<td>006A5A</td>
<td>asl.l #4,D2</td>
<td>Shift in position</td>
</tr>
<tr>
<td>006A5C</td>
<td>bsr $6B30</td>
<td>bcdbin</td>
</tr>
<tr>
<td>006A60</td>
<td>add.b D0,D2</td>
<td>Add month</td>
</tr>
<tr>
<td>006A62</td>
<td>asl.l #5,D2</td>
<td>And shift in position</td>
</tr>
<tr>
<td>006A66</td>
<td>bsr $6B30</td>
<td>bcdbin</td>
</tr>
<tr>
<td>006A68</td>
<td>add.b D0,D2</td>
<td>Add day</td>
</tr>
<tr>
<td>Address</td>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>006A6A</td>
<td>EB82</td>
<td>ASL.l #5, D2</td>
</tr>
<tr>
<td>006A6C</td>
<td>6100000C2</td>
<td>And shift in position</td>
</tr>
<tr>
<td>006A70</td>
<td>D400</td>
<td>BSR $6B30</td>
</tr>
<tr>
<td>006A72</td>
<td>8D82</td>
<td>ADD.b D0, D2</td>
</tr>
<tr>
<td>006A74</td>
<td>6100009A</td>
<td>ASL.l #6, D2</td>
</tr>
<tr>
<td>006A78</td>
<td>D400</td>
<td>BSR $6B30</td>
</tr>
<tr>
<td>006A7A</td>
<td>EB82</td>
<td>ADD.b D0, D2</td>
</tr>
<tr>
<td>006A7C</td>
<td>610000B2</td>
<td>ASL.l #5, D2</td>
</tr>
<tr>
<td>006A80</td>
<td>E208</td>
<td>BSR $6B30</td>
</tr>
<tr>
<td>006A82</td>
<td>D400</td>
<td>LSR.b #1, D0</td>
</tr>
<tr>
<td>006A84</td>
<td>2B420A4C</td>
<td>ADD.b D0, D2</td>
</tr>
<tr>
<td>006A88</td>
<td>1B7C00000AA8E</td>
<td>Move l D2, $A4C(A5)</td>
</tr>
<tr>
<td>006A8E</td>
<td>4E75</td>
<td>Move b #60, $A9E(A5)</td>
</tr>
<tr>
<td>006A90</td>
<td>1B7CFPPFOA82</td>
<td>Move b #1-$A8E(A5)</td>
</tr>
<tr>
<td>006A96</td>
<td>123C001C</td>
<td>Move b #31C, D1</td>
</tr>
<tr>
<td>006A9A</td>
<td>61000234</td>
<td>BSR $6CD0</td>
</tr>
<tr>
<td>006A9E</td>
<td>4A2D0A8E</td>
<td>TST.b $A8E(A5)</td>
</tr>
<tr>
<td>006AA2</td>
<td>66FA</td>
<td>BNE $A9E</td>
</tr>
<tr>
<td>006AA4</td>
<td>202D0A4C</td>
<td>Move l $A4C(A5), D0</td>
</tr>
<tr>
<td>006AA8</td>
<td>4E75</td>
<td>RTS</td>
</tr>
<tr>
<td>006AAA</td>
<td>2B6F00040A50</td>
<td>Move b 4(A7), $A50(A5)</td>
</tr>
<tr>
<td>006AB0</td>
<td>41F900000A5A</td>
<td>Move l $A5A, A0</td>
</tr>
<tr>
<td>006AB6</td>
<td>242D0A50</td>
<td>Move l $A50(A5), D2</td>
</tr>
<tr>
<td>006ABA</td>
<td>1002</td>
<td>Move b D2, D0</td>
</tr>
<tr>
<td>006ABC</td>
<td>0200001F</td>
<td>AND.b #31, D0</td>
</tr>
<tr>
<td>006AC0</td>
<td>E300</td>
<td>ASL.b #1, D0</td>
</tr>
</tbody>
</table>

**gettime, get current clock time and date**
- Request handshake flag for time
- Get time of day command
- Send
- New time arrived?
- No, wait
- Get time in DO

**settime, set clock time and date**
- Pass time

**ikbdtime**
- Pointer to end of time buffer
- Get time to convert
- To DO
- Isolate bits 0-4, seconds
- 2-second resolution

---

**And shift in position**
**bcdbin**
**Add hour**
**And shift in position**
**bcdbin**
**Add minute**
**And shift in position**
**bcdbin**
**2-second resolution**
**And add seconds**
**Save new time**
**Clear handshake flag**
Convert
Minute
Isolate bits 0-5
Convert
Hours
Isolate bits 0-4
Convert
Month
Isolate bits 0-3
Convert
Year
Isolate bits 0-7
Convert
Add offset
Set Time Of Day command
Send to IKBD
Number of bytes to send
Address of the parameter block
Send
Get Time Of Day command
Send to IKBD

006AC2 6154
006AC4 EABA
006AC6 1002
006AC8 0200003F
006ACC 614A
006AC8 ECHA
006AD0 1002
006AD2 0200001F
006AD6 6140
006AD8 EABA
006ADA 1002
006ADC 0200001F
006AE0 6136
006AE2 EABA
006AE4 1002
006AE6 0200000F
006AED 612C
006AEC RABA
006AEE 1002
006AF0 0200000F
006AF4 6122
006AF6 061000A0
006AF8 123C0018
006AFA 610001D0
006B02 7605
006B04 45F90000A5A
006B0A 610001E4
006B0C 123C001C
006B12 610001BC
006B16 4E75

bsr $6B18
lsr.l #5, D2
move.b D2, D0
and.b #63, D0
bsr $6B18
lsr.l #6, D2
move.b D2, D0
and.b #31, D0
bsr $6B18
lsr.l #5, D2
move.b D2, D0
and.b #15, D0
bsr $6B18
lsr.l #4, D2
move.b D2, D0
and.b #57, D0
bsr $6B18
add.b #580, (A0)
move.b #1B, D1
bsr $6CD0
moveq.l #5, D3
lea $A54, A2
bsr $6CFO
move.b #1C, D1
bsr $6CD0
rts
```
moveq .1  #0,D1
moveq .1  #10,D3
sub.b    D3,D0
bmi     $6B24
addq.b  #1,D1
bra      $6B1C
add.b   #10,D0
div.b   #4,D1
add.b   D1,D0
move.b  D0,-(AO)

moveq.1 #0,D0
move.b  (AO),D0
lsr.b   #4,D0
lsl.b   #1,D0
move.b  D0,D1
asr.b  #2,D0
add.b   D1,D0
move.b  (AO)+,D1
and.w   #15,D1
add.w   D1,D0
rts

moveq.1 #1,D0
move.b  $FFFFFC04,D2
btst    #1,D2
bne     $6B58
moveq.1 #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

midioct, MIDI output status
Default to OK
Read MIDI-ACIA status
And test
OK
Status not OK
```
```assembly
006B58 4E75  rts

*****************************************************************************
006B5A 322F0006  move.w 6(A7),D1
006B5E 43F9FFFFFF04  lea $FFFFFFC04,A1
006B64 14290000  move.b 0(A1),D2
006B68 08020001  btst #1,D2
006B6C 67F6  beq $6B64
006B6E 13410002  move.b D1,2(A1)
006B72 4E75  rts

*****************************************************************************
006B74 7600  moveq.l #0,D3
006B76 362F0004  move.w 4(A7),D3
006B7A 246F0006  move.l 6(A7),A2
006B7E D21A  move.b (A2)+,D1
006B80 61DC  bsr $6B5E
006B82 51CBFFFA  dbra D3,$6B7E
006B86 4E75  rts

*****************************************************************************
006B88 41E0DA00  lea $.A00(A5),A0
006B8C 43F9FFFFFF04  lea $FFFFFFFC04,A1
006B92 70FF  moveq.l #1,D0
006B94 45EB0006  lea 6(A0),A2
006B98 47EB0008  lea 8(A0),A3
006B9C D549  cmpx.w (A3)+,(A2)+
006BA0 6652  bne $6BA2
006BA4 7000  moveq.l #0,D0
006BA8 4E75  rts

*****************************************************************************

midwc, output character to MIDI
Get character
Pointer to MIDI-ACIA
Get MIDI status
Test
Wait until OK
Output byte to ACIA

midws, send string to MIDI
Length of string -1
Address of the string
Get byte from string
Output to MIDI
Next byte

midstat, MIDI receiver status
iorec for MIDI
Pointer to MIDI-ACIA
OK as default
Head Index
Tail Index
Uninterruptable test
Not equal
Not OK
```
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

listout, 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
006BF0 7000
006BF0 286D04BACA80
006C04 4E75
006C0E 40C3
006C08 007C0700
006C0C 7207
006C0E 61000E20
006C12 00000080
006C16 7287
006C18 61000E1E
006C1C 46C3
006C1E 302F0006
006C22 728F
006C24 61000E12
006C28 610C
006C2A 6104
006C2C 70FF
006C2E 4E75
moveq.l #0, D0
move.l $4BA(A5), $A00(A5)
rts
move.w SR, D3
or.w #$700, SR
moveq.l #7, D1
bsr $7A38
or.b #$80, DC
moveq.l #$87, D1
bsr $7A38
move.w D3, SR
move.w $A7, D0
moveq.l #$8F, D1
bsr $7A38
bsr $6C36
bsr $6C30
moveq.l #1, D0
rts

******************************************************************************
006C30 7620
006C32 69000E46
moveq.l #$20, D2
bra $7A7A

******************************************************************************
006C36 4DFF
006C38 60000E66
moveq.l #$DF, D2
bra $7A94

******************************************************************************
006C3C 7207
006C3E 61000DF8
006C42 0200007F
006C46 7287
moveq.l #7, D1
bsr $7A38
and.b #$7F, D0
moveq.l #$87, D1

Flag for time out
200 Hz counter as last time-out time

Save status
IPL 7, no interrupts
Mixer
Select registers
Port B to output
Write enable

Restore status
Character to output
Write to port B

Strobe low
Strobe high
Flag for OK

Strobe high
Bit 5
Set

Strobe low
Bit 5
Clear

listin, Get character from parallel port
Mixer
Select register
Port B to input
**Strobe high = not busy**
Get parallel port status

**Wait until character arrives**

**Strobe low = busy**

Select port B

Read byte from port

---

**laststat, 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
Isolate bits 0-7
auxostat, RS232 output status
iorec for RS232
Default to ok
Tail index
Test for wrap around
Compare with head index
No more space in buffer
Isolate auxout, RS232 output routine
Get data byte
rs232put, and output
Try again
ikbdost, IKBD output status
IKBD ACIA status
Test
OK
Not ready
ikbdwc, send character to IKBD
Get byte
Address of the keyboard ACIA
Ready
Abacus Software

**Atari ST Internals**

---

```
006CDE 67F6  beq $6CD6
006CE0 13410002  move.b D1,2(A1)
006CE4 4275  rts

**************************************************************************
006C6E 7600  moveq.l #0,D3
006C78 36FD0004  move.w 4(A7),D3
006C7C 246F0006  move.l 6(A7),A2
006C80 121A  move.b (A2)+,D1
006C84 61DC  bsr $6CD0
006C88 51CF00FA  dbra D3,$5CF0
006C8C 4275  rts

**************************************************************************
006CFA 41ED0972  lea $9F2(A5),A0
006CF2 7CF8  moveq.l #1,D0
006CFC 45E80000  lea 6(A0),A2
006CDD 47E80008  lea 8(A0),A3
006CF0 B54B  cmp.s (A3)+,(A2)+
006CF4 6602  bne $6D0E
006CF8 7000  moveq.l #0,D0
006CFC 4275  rts

**************************************************************************
006D10 61E8  bsr $6CFA
006D12 4A40  tst.w D0
006D14 67FA  beq $6D10
006D18 007C0707  move.w SR, -(A7)
006D1C 32200006  or.w $700,SR
006D20 D6B00006  move.w 6(A0),D1
006D24 671C  bne $6D42

No, wait
Output character

ix6s, send string to keyboard

Number of characters -1
Address of the string
Get byte from string
And output
Next byte

constat, keyboard input status
iorec for keyboard
Default to ox
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
```
Head index + 2
Greater than or equal buffer size?
No
Buffer pointer back to start
Connector to keyboard buffer
Get character and scan code
Save new head index
Scan code to bits 16-23
ASC II code to bits 0-7
Restore status
***************
conoutstat, 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

Atari ST Internals

initnfp, initialize MFP 69901
Address of the MFP
Initialize registers with zero
gpip to iera
ierb to isrb
isrb to vr
MFP Non-autovector # to $48, set 8-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
moveq.l #1,D1
moveq.l #2,D2
bje $7090
move.l #$980101,DO
movep.l DO,$26(A0)
bje $7A70
bje $7A68
lea $(0000),A0
lea $(0000),A1
moveq.l #$33,DO
bje $7036
lea $(0000),A0
lea $(0000),A1
moveq.l #$13,DO
bje $7036
move.l #$759c,DO
move.l DA12(A5)
move.l DA16(A5)
move.l #$7568,DA2A(A5)
move.l #$7568,DA2E(A5)
move.b #$3,FF3FC04
move.b #$95,FF3FC04
move.b #$7,FF3304
move.l #$6A16,DA22(A5)
move.l #$7034,DO
move.l DA12(A5)
move.l DA16(A5)
move.l DA26(A5)
moveq.l #0,DO
move.l DA86(A5)
move.b DA8A(A5)

/4 for 9600 Baud
9600 baud
Initialize timer and interrupt vector
$00, $98, $01, $01
To scr, ucr, rcr, tsr
DIR on
RTS on
Pointer to iorec for $232
Start data for iorec
34 bytes
Copy into RAM
Pointer to iorec MIDI
Start data for iorec
14 bytes
Copy into RAM
Keyboard and MIDI error vector
Pointer to error routine keyboard
Pointer to error routine MIDI
MIDI interrupt vector

MIDI-ACIA master reset
/16, 8 bit, 1 stop bit, no parity
Key click, repeat and bell enable
clonvec joyvec & statvec

Clear sound variables
Sound pointer
Delay timer
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006FBC</td>
<td>1B400A9B</td>
<td>Move byte D0, $A8B (A5)</td>
</tr>
<tr>
<td>006FC0</td>
<td>2B400A80</td>
<td>Move byte D0, $A80 (A5)</td>
</tr>
<tr>
<td>006FC1</td>
<td>6100FC6A</td>
<td>BSR $6C30</td>
</tr>
<tr>
<td>006FC8</td>
<td>1B7C000FC07E</td>
<td>Move byte $SP, $A7E (A5)</td>
</tr>
<tr>
<td>006FCE</td>
<td>1B7C00020A7F</td>
<td>Move byte $2, $A7F (A5)</td>
</tr>
<tr>
<td>006FD4</td>
<td>41ED909F2</td>
<td>Lea $9F2 (A5), A0</td>
</tr>
<tr>
<td>006FD8</td>
<td>63F900007042</td>
<td>Lea $7042, A1</td>
</tr>
<tr>
<td>006FDE</td>
<td>700D</td>
<td>Moveq l #13, D0</td>
</tr>
<tr>
<td>006FE0</td>
<td>6154</td>
<td>BSR $7036</td>
</tr>
<tr>
<td>006FE2</td>
<td>6100FC0E</td>
<td>BSR $7BF2</td>
</tr>
<tr>
<td>006FE6</td>
<td>13FC0003FFFFC00</td>
<td>Move byte $3, $FFFFFC00</td>
</tr>
<tr>
<td>006FE8</td>
<td>13FC0096FFFFFC00</td>
<td>Move byte $96, $FFFFFC00</td>
</tr>
<tr>
<td>006FF6</td>
<td>267C00007080</td>
<td>Move l $7080, A3</td>
</tr>
<tr>
<td>006FFC</td>
<td>7203</td>
<td>Move l #3, D1</td>
</tr>
<tr>
<td>006FFE</td>
<td>2401</td>
<td>Move l D1, D0</td>
</tr>
<tr>
<td>007000</td>
<td>2001</td>
<td>Move l D1, D0</td>
</tr>
<tr>
<td>007002</td>
<td>06000009</td>
<td>Move d #9, D0</td>
</tr>
<tr>
<td>007005</td>
<td>ES52</td>
<td>Move d #2, D2</td>
</tr>
<tr>
<td>007008</td>
<td>24732000</td>
<td>Move l 0(A3, D2, w), A2</td>
</tr>
<tr>
<td>00700C</td>
<td>61000138</td>
<td>BSR $7146</td>
</tr>
<tr>
<td>007010</td>
<td>51C9FFFE</td>
<td>DBRA D1, $6FFE</td>
</tr>
<tr>
<td>007014</td>
<td>45ED752A</td>
<td>Lea $752A (A5), A2</td>
</tr>
<tr>
<td>007018</td>
<td>7006</td>
<td>Moveq l #6, D0</td>
</tr>
<tr>
<td>00701A</td>
<td>6100012A</td>
<td>BSR $7146</td>
</tr>
<tr>
<td>00701E</td>
<td>45ED73C0</td>
<td>Lea $73C0 (A5), A2</td>
</tr>
<tr>
<td>007022</td>
<td>7002</td>
<td>Moveq l #2, D0</td>
</tr>
<tr>
<td>007024</td>
<td>61000120</td>
<td>BSR $7146</td>
</tr>
<tr>
<td>007028</td>
<td>247C0000703E</td>
<td>Move l #8703E, A2</td>
</tr>
<tr>
<td>00702E</td>
<td>7603</td>
<td>Moveq l #3, D3</td>
</tr>
<tr>
<td>007030</td>
<td>6100FC3E</td>
<td>BSR $6CF0</td>
</tr>
<tr>
<td>007034</td>
<td>4B75</td>
<td>RTS</td>
</tr>
</tbody>
</table>

**Notes:**
- Temp value
- Printer timeout
- Strobe to high
- Pointer to iorec to keyboard
- Start data for iorec
- 14 bytes
- Copy into RAM
- Pointer to BIOS keyboard table
- Reset keyboard ACIA
- $74, 0 bits, 1 top bit, no parity
- Pointer to MFIP interrupt vectors
- Initialize four vectors
- Interrupt number
- Plus offset

**Instructions:**
- Get vector from table
- Initint, install interrupt
- Next vector
- MIDI and keyboard vector
- Vector number 6
- Initint, install interrupt vector
- CTS interrupt routine
- Vector number 2
- Initint, install interrupt
- Pointer to init data for IXBD
- 6 bytes
- Send string to IXBD
007036 10D9  move.b  (A1)+,(A0)+
007038 51C8FFFC  dbra  D0,$7036
00703C 4E75  rts

move.b  (AI)+,(AO)+
dbra  D0,$7036
rts

************************************************
00703E 8001121A  dc.b  $80,$01,$12,$1A
*********************************** *************
007042 000008DO  dc.l
007046 0080  dc.w
007048 0000  dc.w
00704A 0000  dc.w
00704C 0020  dc.w
00704E 0060  dc.w

Block move
Next byte

Reset keyboard, disable mouse and joystick

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

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

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

$iorec$ 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

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

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
Head index
Tail index
Low water mark, 1/4 buffer length
High water mark, 3/4 buffer length
rsrbyte, receiver status
tsrbyte, transmitter status
rxoff,
txoff
rsmode, XON/XOFF mode
filler

Interrupt vectors for MFP
#9, transmitter error
#10, transmitter interrupt
#11, receiver error
#12, receiver interrupt

setimer, initialize timer in MFP
Save registers
Address MFP
0070D2 267C0000712C          move.l  #5712C,A3
0070D6 247C00007130          move.l  #57130,A2
0070DE 6122                 bsr    57102
0070E0 C749                exg    A3,A1
0070E2 47F900007134         lea    #57134,A3
0070E8 7600               moveq.l #0,D3
0070EA 16330000             move.b  D2,0(A0,D3.w),D3
0070F2 84303000         move.b  D2,0(A0,D3.w),D2
0070F6 66F6                     bne    8705E
0070F8 C749                  exg    A3,A1
0070FA 8313                  or.b   DI,(A3)
0070FC 4CDF0F1F       movem.l (A7)+,D0-D4/A0-A3
007100 4E75               rts

007102 6106                 bsr    8710A
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 D4CD               add.w   D0,A2
007116 4E75               rts

**************************************************************************
007118 06060808          dc.b    6,6,8,8
00711C 0A0A0C0C          dc.b    10,10,12,12
movem.l DO-D2/AO-A2,-(A7)

********************
move.w 4(A7),DO
move.l 6(A7),A2
and.l #15,00

********************
48E7H0D0
6120
2400
E542
0682000010C
2242
228A
614A
4C90707

********************
move.w 4(A7),DO
and.l #15,00
movem.l D0-D1/A0-A1,-(A7)
lea SFFFFFA01,A0
lea 18(A0),A1
bar $71C6

********************
disint, disable MFP interrupts
Get interrupt number
Convert to long word index
Save registers
Address the MFP
Address inra
Calculate bit number to clear

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

mfpint, set MFP interrupt vector
Interrupt number
Interrupt vector
Number 0-15, long word
bclr $D1, (A1)
lea $6(A0), A1
bsr $71C6
bclr $D1, (A1)
lea $10(A0), A1
bsr $71C6
bclr $D1, (A1)
lea $14(A0), A1
bsr $71C6
bclr $D1, (A1)
movem.l (A7)+, D0-D1/A0-A1
rts

And clear bit
Address iera
Calculate bit number to clear
And clear bit
Address ipra
Calculate bit number to clear
And clear bit
Address isra
Calculate bit number to clear
And clear bit
Restore registers

***************
move.w 4(A7), DO
and.l #15, DO
movem.l D0-D1/A0-A1, -(A7)
lea $FFFFFFFA01, A0
lea $E(A0), A1
bsr $71C6
bset $D1, (A1)
lea $18(A0), A1
bsr $71C6
bset $D1, (A1)
movem.l (A7)+, D0-D1/A0-A1
rts

jenabint, enable MFP interrupts
Vector number
In long word index
Restore registers
Address MFP
Address iera
Calculate bit number to set
And set bit
Address imra
Calculate bit number to set
And set bit
Restore registers

bselect, calculate bit and register number

move.b D0, D1
cmp.b #8, D0
blt $71D0

Save interrupt number
Greater than 8?
No
Else subtract offset
Greater than 8 ?
Yes
Pointer from a to b register

rs232ptr
Pointer to RS232 iorec
Pointer to MFP

rs232ibuf
Tail index
Head index
Head > tail ?
No
Add buffer size
Tail - head

rtschk
RTS/CTS mode ?
No
rtson

rs232put, RS232 output
Save status
IPL 7, disable interrupts
rs232ptr, get buffer pointer
XON/XOFF mode ?
No
; XON active ?
; Yes
; Is MFP still sending ?
; Yes
; Head index
; Tail index
; Still characters in buffer?
; Pass byte to MFP

00723A 4428001F  tst.b  31(A0)   ; XON active ?
00723B  6B18 bne  $7238
00723C  08290007002C btst  #7,44(A1)  ; Is MFP still sending ?
00723D  6710 beq  $7238
00723E  34280014 move.w 20(A0),D2  ; Head index
00723F  B4680015 cmp.w 22(A0),D2  ; Tail index
007240  6606 bne  $7238
007241  1341002E move.b D1,46(A1)  ; Still characters in buffer?
007242  601A bra  $7252  ; Pass byte to MFP

007243  34280016 move.w 22(A0),D2  ; Tail index
007244  B4680014 cmp.w 20(A0),D2  ; Test for wrap around
007245  6716 beq  $725C
007246  2268000E move.l 14(A0),A1  ; Compare with head index
007247  13812000 cmp.w D1,0(A1,D2),w  ; Same, buffer full
007248  31620016 move.b D2,22(A0)  ; Get current buffer address
007249  61A8 bsr  $71FC
00724A  46DF move.w (A7)+,SR  ; Write character in buffer
00724B  02C000FE and.b #$254,SR  ; Save new tail index
00724C  4E75 rts  ; rtschk, set RTS ?

00724D  619E bsr  $71FC
00724E  46DF move.w (A7)+,SR  ; Restore status flag
00724F  003C0001 or.b #$1,SR  ; Clear carry flag
007250  4E75 rts  ; rtschk, set RTS ?

******************************************************************************
007251  40E7 move.w SR, -(A7)  ; Restore status
007252  007C0700 or.w #$700,SR  ; Set carry flag, don't output char
007253  6100FF6C bsr  $71DA
007254  32280006 move.w 6(A0),D1  ; rs232get, RS232 input

rs232get, RS232 input
Save status
IPL 7, disable interrupts
rs232ptr, get RS232 pointer
Head index
cmp.w 8(A0), D1
beq $7294
ber $7512
move.l 0(AO), A1
moveq.l #0, D0
move.b 0(A1), D1.w, D0
move.w D1, 6(A0)
move.w (A7)+, SR
and.b #$254, SR
bra $729A

move.w (A7)+, SR
or.b #1, SR
btst #0, 32(A0)
beq $72BE
tst.b 30(A0)
beq $72BE
bar $71BE
comp.w 10(A0), D2
bne $72BE
move.b #$11, D1
bar $720A
cir.b 30(A0)
rs

**************************************************
Tail index
No character in buffer?
Test for wrap around
Get buffer address
Get character from buffer
Set new head index
Restore status
Clear carry flag, OK

Restore status
Set carry flag, no character there
XON/XOFF mode?
No
XON active?
No
Get input buffer length used
Equal low water mark?
No
XON
Send
Clear XON flag
rcvint, RS232 receiver interrupt routine
Save registers
rs232ptr, get RS232 pointer
Read receiver status register
Interrupt through receiver buffer full?
No
RTS/CTS mode?
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>007306</td>
<td>0C000013</td>
<td>CMP.B #19,D0</td>
</tr>
<tr>
<td>00730A</td>
<td>6608</td>
<td>BNE $7314</td>
</tr>
<tr>
<td>00730C</td>
<td>117C00FF001F</td>
<td>MOVE.B #$FF,31(A0)</td>
</tr>
<tr>
<td>007312</td>
<td>6054</td>
<td>BRA $7368</td>
</tr>
<tr>
<td>007314</td>
<td>32280008</td>
<td>MOVE.W B(A0),D1</td>
</tr>
<tr>
<td>007318</td>
<td>610001F0</td>
<td>BSR $7512</td>
</tr>
<tr>
<td>00731C</td>
<td>B2680006</td>
<td>CMP.W #0(A0),D1</td>
</tr>
<tr>
<td>007320</td>
<td>6746</td>
<td>BEQ $7368</td>
</tr>
<tr>
<td>007322</td>
<td>24680000</td>
<td>MOVE.L 0(A0),A2</td>
</tr>
<tr>
<td>007326</td>
<td>15801000</td>
<td>MOVE.B D0,0(A2,D1.w)</td>
</tr>
<tr>
<td>00732A</td>
<td>31410008</td>
<td>MOVE.W D1,8(A0)</td>
</tr>
<tr>
<td>00732E</td>
<td>6100FE88</td>
<td>BSR $71E0</td>
</tr>
<tr>
<td>007332</td>
<td>B468000C</td>
<td>CMP.W 12(A0),D2</td>
</tr>
<tr>
<td>007336</td>
<td>6624</td>
<td>BNE $735C</td>
</tr>
<tr>
<td>007336</td>
<td>08280001002D</td>
<td>BTST $1,32(A0)</td>
</tr>
<tr>
<td>00733E</td>
<td>6628</td>
<td>BNE $7368</td>
</tr>
<tr>
<td>007340</td>
<td>08280000002D</td>
<td>BTST $0,32(A0)</td>
</tr>
<tr>
<td>007345</td>
<td>6714</td>
<td>BEQ $735C</td>
</tr>
<tr>
<td>007348</td>
<td>4A28001E</td>
<td>TST.B 30(A0)</td>
</tr>
</tbody>
</table>

- No
- rtsoff
- Read data from receiver register
- RTS/CTS mode?
- Yes
- XON/XOFF mode?
- No
- XON received?
- No
- Clear XOFF flag
- Character not in buffer
- Receive XOFF?
- No
- Set XOFF flag
- Character not in buffer
- Tail index
- Text for wrap around
- Head equal tail?
- Yes
- Get buffer address
- Received byte in buffer
- Save new tail index
- Get input buffer length used
- Equal high water mark?
- No
- RTS/CTS mode?
- Yes
- XON/XOFF mode?
- No
- XOFF already sent?
Yes
Set flag for XOFF
XOFF
Send
RTS/CTS mode?
No
rtson
Clear interrupt service bit
Restore registers

***************

movem.l D2/A0-A2,-(A7)
brr $71DA
btst $1,32 (A0)
ble $73B4
btst $0,32 (A0)
beq $7392
tsb3 $A0
ble $73B4
move.b 44(A1),29(A0)
move.w 20(A0),D2
cmp.w 22(A0),D2
beq $7392
brr $781E
move.l 14(A0),A2
move.b 0(A2,D2.w),66(A1)
move.w D2,20(A0)
bclr #2,14(A1)
movem.l (A7)+,D2/A0-A2
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 MFP for sending
Save new head index
Clear interrupt service bit
Restore registers
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 $73DO!)
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 1169002C01D  move.b  4(A7),29(A0)
007434 08A90001000E  bclr  #1,14(A1)
00743A 4CDFG300  movem.1 (A7)+/A0-A1
00743E 4273  rte

******************************************************************************
007440 7200  moveq.1 #0,D1
007442 322F0004  move.w  4(A7),D1
007446 40E7  move.w  SR,-(A7)
007448 007C0700  or.w  #$700,SR
00744C 45F9000745C  lea  $745C,A2
007452 E581  asll.1 #2,D1
007454 2D321800  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 6100F6DC  bsr  $71DA
007470 CF490028  movem.1  $28(A1),D7
007474 7C00  move.l  #0,D0
007476 1340002A  move.b  D0,42(A1)
00747A 1340002C  move.b  D0,40(A1)
00747E 4A6FD006  tst.w  6(A7)
007482 680A  bnl  $748E
007484 116F0070020  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
IRBD
MIDI

rsconf, configure RS232
IPL 7, disable interrupts
rs232ptr, get RS232 pointer
Save ucr, rcr, rcr, scr

Disable receiver
Disable transmitter
Mode
Negative, then don't set
Reset rmode
Baud rate
Negative, then don't change
Get baud rate number
Table for baud rate control
Get value from table
Table for baud rate data
Get value from table
Pointer to timer D
Set timer D for new baud rate
Set ucr?
No
Set ucr
Set rsr?
No
Set rsr
Set tsr?
No
Set tsr
Set scr?
No
Set scr
Enable receiver
Enable transmitter
Old value of the control register

Timer values for RS-232 baud rates
High byte
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
Equal to buffer size?
No
Else start with zero again

midikey, keyboard + MIDI interrupt
Save registers
Clear A5
mbufrec, $7558
Interrupt from MIDI-ACIA?
xbufrec, $7568
Interrupt from on Keyboard-ACIA?
Still interrupt?
Yes
Clear interrupt service bit
Restore registers
MIDI interrupt
irec for MIDI
Address of the MIDI ACIA
$759C, MIDI error routine

Keyboard interrupt
irec 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
0075BE 0280000000FF
0075C4 47F9000075F2
0075CA 1B7300000A32
0075C0 47F9000075FC
0075C6 1B7300000A33
0075C2 064000F6
0075C8 0C0000F8
0075CA 600A
0075CC 0C0000FB
0075CE 6E04
0075D0 1B400A40
0075D0 4275
sub.b #$F6, #DO
and.l #$FE, #DO
lea $75F2, A3
move.b 0(A3, DO.w), $A32(A5)
lea $75FC, A3
move.b 0(A3, DO.w), $A33(A5)
add.w #$F6, DO
cmp.b #$F8, DO
blt $75F0
cmp.b #$FB, DO
bgt $75F0
move.b D0, $A40(A5)

Subtract offset
Pointer to IKBD code table
Save IKBD merken
Pointer to IKBD lengths table
IKBD index
Add offset again
Mouse position record ?
No
Mouse position record ?
No
Save mouse position

IKBD parameter
Status codes
Lengths

007606 0C200060A32
00760C 64000008
007610 45F90007656
007616 7400
007618 142D0A32
00761C 5302
00761E E342
007620 D2E0A32
007624 5302
007626 E542
007628 20722200
00762C 22722204
007630 24722208

cmp.b #$6, $A32(A5)
bcc $7692
lea $7656, A2
moveq.l #0, D2
move.b $A32(A5), D2
subq.b #1, D2
asl.w #1, D2
add.b $A32(A5), D2
subq.b #1, D2
asl.w #2, D2

Joystick record ?
Yes
Pointer to IKBD parameter table
Kstate
1 - 5 => 0 - 4
Times 2
+ 1
1 - 5 => 0 - 4
Times 4
IKBD record pointer
IKBD index base
IKBD interrupt routine
007634 2452  move.l (A2),A2  Get interrupt vector
007636 7400  moveq.l #0,D2  Get IKBD index
007638 1280  sub.l D2,(A1)  Minus base
00763A 42D0A33  move.b $A33(A5),D2  IKBD index minus 1
00763C 93C2  tst.b $A33(A5)  Test index
00763E 1280  subq.b #1,$A33(A5)  Pass record pointer
007640 532D0A33  tst.b $A33(A5)  Execute interrupt routine
007642 660A  bne $7654  Correct stack pointer
007644 4A2DOA33  moveq.l A0,-(A7)  Clear IKBD state
007646 660A  move.l A0,-(A7)  Parameter table for IKBD
007648 56F4  jsr (A2)  Parameter table for IKBD
00764A 2FC8  move.l A0,-(A7)  Parameter table for IKBD
00764C 5292  jsr (A2)  Parameter table for IKBD
00764E 584F  moveq.l A0,-(A7)  Parameter table for IKBD
007650 42D0A32  moveq.l A0,-(A7)  Parameter table for IKBD
007652 4E75  rts  Parameter table for IKBD
007654 0000A34  dc.l $A34  Parameter table for IKBD
007656 0000A3B  dc.l $A3B  Parameter table for IKBD
007658 0000A1A  dc.l $A1A  Parameter table for IKBD
00765A 0000A3B  dc.l $A3B  Parameter table for IKBD
00765C 0000A1A  dc.l $A1A  Parameter table for IKBD
00765E 0000A3B  dc.l $A3B  Parameter table for IKBD
007660 0000A40  dc.l $A40  Parameter table for IKBD
007662 0000A40  dc.l $A40  Parameter table for IKBD
007664 0000A40  dc.l $A40  Parameter table for IKBD
007666 0000A40  dc.l $A40  Parameter table for IKBD
007668 0000A40  dc.l $A40  Parameter table for IKBD
00766A 0000A40  dc.l $A40  Parameter table for IKBD
00766C 0000A40  dc.l $A40  Parameter table for IKBD
00766E 0000A40  dc.l $A40  Parameter table for IKBD
007670 0000A40  dc.l $A40  Parameter table for IKBD
007672 0000A40  dc.l $A40  Parameter table for IKBD
007674 0000A40  dc.l $A40  Parameter table for IKBD
007676 0000A40  dc.l $A40  Parameter table for IKBD
007678 0000A40  dc.l $A40  Parameter table for IKBD
00767A 0000A40  dc.l $A40  Parameter table for IKBD
00767C 0000A40  dc.l $A40  Parameter table for IKBD
00767E 0000A40  dc.l $A40  Parameter table for IKBD
007680 0000A40  dc.l $A40  Parameter table for IKBD
007682 0000A40  dc.l $A40  Parameter table for IKBD
007684 0000A40  dc.l $A40  Parameter table for IKBD
007686 0000A40  dc.l $A40  Parameter table for IKBD
007688 0000A40  dc.l $A40  Parameter table for IKBD
00768A 0000A40  dc.l $A40  Parameter table for IKBD
00768C 0000A40  dc.l $A40  Parameter table for IKBD
00768E 0000A40  dc.l $A40  Parameter table for IKBD
Joystick interrupt routine

Address of joystick data

Process received keyboard codes
Load shift status

Left shift key pressed ?
No

Set bit for left shift key

Left shift key released ?

Clear bit for left shift key

Right shift key pressed ?
No

Set bit for right shift key

Right shift key released ?

Clear bit for right shift key

CTRL key pressed ?
No

Set bit for CTRL key

CTRL key released ?
0076F0 6606           bne  $76F8
0076F2 08810002         bclr  $2,D1
0076F6 6038             bra  $7730
0076F8 0C000038         cmp.b  $38,D0
0076FC 6606             bne  $7704
0076FE 08C10003         bset  $3,D1
007702 602C             bra  $7730
007704 0C0000B8         cmp.b  $88,D0
007708 6606             bne  $7710
00770A 08810003         bclr  $3,D1
00770E 6020             bra  $7730
007710 0C00003A         cmp.b  $3A,D0
007714 6620             bne  $7736
007716 082D000001B4       btst  $0,$4B4(A5)
00771C 670E             beq  $772C
00771E 2B7C0007D780A86  move.l  $7D78,$A86(A5)
007726 1B7C00000A8A   move.b  $5D,$A9A(A5)
00772C 08410004          bchg  $4,D1
007730 1B410A5D         move.b  D1,$A5D(A5)
007734 4E75             rts

007736 08000007       btst  $7,D0
007738 662A           bne  $7766
00773C 4A200A7B       tst.b  $A7B(A5)
007740 6616           bne  $7758
007742 1B400A7B       move.b  D0,$A7B(A5)
007746 1B79000007E0A7C  move.b  $A7E,$A7C(A5)
00774E 1B79000007F0A7D  move.b  $A7F,$A7D(A5)
007756 603A           bra  $7792
007758 1B7C00000A7C    move.b  $S0,$A7C(A5)
00775E 1B7C00000A7D    move.b  $S0,$A7D(A5)

No  Clear bit for CTRL key
 ALT key pressed ?
No  Set bit for ALT key
 ALT key released ?
No  Clear bit for ALT key
CAPS LOCK key pressed ?
No  Get console configuration
No key click
Address of the key click sound table
Start sound
Invert CAPS LOCK status
Save new shift status
Was key released ?
Yes  Repeat ?
Yes  Save key code for repeat
Delay 1
Delay 2
Clear delay counter
Clear delay counter
HOME key released?
Yes
INSERT key released?
No
ALT key still pressed?
No
Console status
No key click
Address of the sound table for key click
Start sound
Save lores for keyboard

Scan code to D1
Address of the standard keyboard table

CAPS LOCK active?
No
Address of the CAPS LOCK keyboard table
Right shift key pressed?
Yes
Left shift key pressed?
No
Function key?
No
Function key?
No
Add offset to GSX standard
Address of the shift keyboard table
Get ASCII code from the table
CTRL key pressed?
No
Carriage return
No
Convert to linefeed
CTRL HOME?
No
Add $30 to GSX standard
CTRL cursor left?
Convert to GSX standard
CTRL cursor right?
Convert to GSX standard
007834 60000162  bra $7998  
br  
007838 0C000036  cmp.b #$36,D0  
00783C 6606  bne $7866  
00783E 701E  moveq.l #$12,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 082D0030A5D  btst #3,8A5D(A5)  
00785E 67000138  beq $7998  
007862 0C01001A  cmp.b #$1A,D1  
007866 6618  bne $7880  
007868 103C0040  move.b #$40,D0  
00786C 142D0A5D  move.b 8A5D(A5),D2  
007870 020200C3  and.b #3,D2  
007874 67000122  beq $7998  
007878 103C005C  move.b #$5C,D0  
00787C 6000011A  bra $7998  
007880 0C010027  cmp.b #$27,D1  
007884 6618  bne $7892  
007886 103C005B  move.b #$5B,D0  
00788A 142D0A5D  move.b 8A5D(A5),D2  
00788E 02020003  and.b #3,D2  
007892 67000104  beq $7998  
007896 103C007B  move.b #$7B,D0  
00789A 600000FC  bra $7998  
00789E 0C010028  cmp.b #$28,D1  
0078A2 6618  bne $788C  
0078A4 103C005D  move.b #$5D,D0  

ALT key pressed?  
No  
'0'?  
No  
'g'  
Shift status  
Left and/or right shift key pressed?  
No  
'\'  
'Q'?  
No  
'!'  
Shift status  
Left and/or right shift key pressed?  
No  
'!'  
'X'?  
' '
move.b $A5D(A5),D2
and.b $3,D2
beq $7998
move.b $7D,D0
bra $7998
cmp.b $62,D1
bne $78CC
addq.w #1,84EE(A5)
move.l (A7)+,A0
bra $79BE
lea $7A2C,A2
moveq.l $3,D2
cmp.b 0(A2,D2),D1
beq $79E8
dbra D2,$78D4
cmp.b $48,D1
bne $7902
move.b #0,D1
move.b #8,D2
move.b $A5D(A5),D0
and.b $3,D0
beq $7A06
move.b #1,D2
bra $7A06
cmp.b $48,D1
bne $7924
move.b #0,D2
move.b #8,D1
move.b $A5D(A5),D0
and.b $3,D0
beq $7A06
Shift status
Left and/or right shift key pressed?
No
'
ATL HELP?
No
Set dumpfig for hardcopy
Restore keyboard iorec
Done
Pointer to mouse scancode table
Test 4 values
Value found?
Yes
Next value
Cursor up?
No
X offset for cursor up
Y offset for cursor up
Get shift status
Left and/or right shift key pressed?
No
Only one pixel up with shift
Cursor left?
No
Y offset for cursor left
X offset for cursor left
Get shift status
Left and/or right shift key pressed?
No
Only one pixels left with shift

Cursor right?
No
X offset for cursor right
Y offset for cursor right

Shift status
Left and/or right shift key pressed?
No
Only one pixel right with shift

Cursor down?
No
X offset for cursor down
Y offset for cursor down

Shift status
Left and/or right shift key pressed?
No
Only one pixel down with shift

'A'
Not greater or equal

'B'
Not less or equal

'Z'
00798A OC000061  cmp.b    #61,00
00798E  6508  bcs       $7998
007990  OC00007A  cmp.b    #67A,00
007994  6202  bhi       $7998
007996  60EE  bra       $7986
007998  E141  asl.w    #8,01
00799A  D041  add.w    D1,00
00799C  205F  move.l (A7),A0
00799E  32280008  move.w  6(A0),D1
0079A2  5401  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  24600A0E  move.l  SACE(A5),A2
0079CC  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
0079DA  B2680006  beq       $79FA
0079DB  670C

'a'
'z'
Scan code to bits 8-15
ASCII code to bits 0-7
Get ioreg pointer
Tail index
Plus 2
End of buffer reached?
No
Buffer pointer back to buffer start
Head equal tail?
Yes, buffer full
Buffer address
Write data in buffer
New tail index
mido
Byte
Pointer to MIDI interrupt handler
Execute routine
sysmd
Tail index
Increment
End of buffer reached?
No
Buffer pointer back to start
Head equal tail?
Yes, buffer full
move.l 0(A0),A2
move.b D0,0(A2,D1.w)
move.w D1,8(A0)

Buffer address
Write received byte in buffer
New tail index

keymausl
Accept right button

is right button ($47/$C7)
left button
Pressed or released ?
Pressed
Clear bit for key

Set bit for key

lea $A5A(A5),A0
move.l $A1E(A5),A2
clr.l D0
move.b $A5D(A5),D0
lsr.b $F8,D0
add.b $F8,D0
move.b D0,0(A0)
move.b D1,1(A0)
move.b D2,2(A0)
jsr (A2)
move.l (A7),+A0
rts

keymaus
Pointer to mouse-emulator buffer
Mouse interrupt vector

Get status of the "mouse" buttons
Bit for right button to bit 0
Plus relative mouse header
Byte in buffer
Store X value
Store Y value
Call mouse interrupt routine
Restore iorec for keyboard
muskeyl
Scan code mouse substitute

giaccess, read/write sound chip
Data
Save status
IPL 7, disable interrupts
Save registers
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

007A74  7400  moveq.l  #0,D2
007A76  342F0004  move.w  4(A7),D2
007A78  48E7E000  movem.l  D0-D2,-(A7)
007A7A  40E7  move.w  SR,-(A7)
007A80  007C0700  or.w  #$700,SR
007A84  7202  moveq.l  #14,D1
007A86  2FC2  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  moveq.l  (A7)+,D0-D2
007A98  4E75  rts

dtroff, enable DTR
Clear bit in port A
onbit, set 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
offbit, clear bit in sound chip port A
Get bit pattern
Save registers
Save status
IPL 7, disable interrupts
Port A
Save bit pattern
Select port A
Restore bit pattern
and.b  D2, D0
moveq.l #$8E, D1
bsr  $7A38
move.w  (A7)+, SR
movem.l (A7)+, D0-D2
rts

****initmouse****

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

**initmouse**

Disable mouse
Disable mouse command
Send to IKBD
Mouse interrupt vector to RTS

**disable mouse**

Relative mouse
Transfer buffer pointer
Relative mouse
Relative mouse threshold x, y
Set mouse parameters

set mouse parameters
moveq.l #6,D3
lea $A6A(A5),A2
bra $6CFO

lea $A6A(A5),A2
move.b #9,(A2)+
move.b 4(A3),(A2)+
move.b 5(A3),(A2)+
move.b 6(A3),(A2)+
move.b 7(A3),(A2)+
move.b #5C,(A2)+
bar $7B70
move.b #$SE,(A2)+
move.b #0,(A2)+
move.b 8(A3),(A2)+
move.b 9(A3),(A2)+
move.b 10(A3),(A2)+
move.b 11(A3),(A2)+
moveq.l #16,D3
lea $A6A(A5),A2
bar $6CFO
bra $7B6C

lea $A6A(A5),A2
move.b #5A,(A2)+
bar $7B10
moveq.l #5,D3
lea $A6A(A5),A2
bra $6CFO
moveq.l #-1,D0
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
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
Abacus Software

Atari ST Internals

**Key functions**

- **keytrans, set keyboard table**
  - Change standard 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**

```assembly
007BC6 4AAFO004  tst.l $A(A7)
bmi $7BD2
move.l $A(A7),SA62(A5)
tst.l $A(A7)
bmi $7BD2
move.l $A(A7),SA62(A5)
tst.l $A(A7)
bmi $7BEA
move.l $A(A7),SA66(A5)
move.l #SA62,DO
007BF0 4E75
tst.l $A(A7),DO
move.l 4(A7),D1
bmi $7C1E
move.l D1,SA86(A5)
cir.b SA86(A5)
007C02 236F0000A66
move.l #SA62,SA66(A5)
move.l #SA62,DO
move.l $A86(A5),DO
007C08 4A6F0004
move.l 4(A7),D1
bmi $7C1E
move.l D1,SA86(A5)
cir.b SA86(A5)
007C0A 203C0000A5E
move.l #SA62,SA66(A5)
move.l #SA62,DO
move.l $A86(A5),DO
007C0C 203D00A86
move.l 4(A7),D1
bmi $7C1E
move.l D1,SA86(A5)
cir.b SA86(A5)
007C10 222F0004
007C14 6B08
007C16 2B410A86
007C18 44220A8A
007C1E 4E75

007C20 303D00A8C
007C24 4A6F0004
007C28 6B06
007C2A 3E8F0000A8C
```
007C30 4275  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

***************************************************
move.1 $A02,D0
007C5A 4E75  rts

******************************************************************************
007C54 52B9000004BA  addq.l #1,$4BA
007C58 4AE7FFFE  ro1.w $A84
007C5C 6A4E  bpl  $7CB8
007C60 48E7FFFE  movem.l D0-D7/A0-A6,-(A7)
007C64 614C  lea  $0,A5
007C68 6A44  bsr  $7CC2
007C6C 2A2D00010484  btst #1,$484(A5)
007C70 672A  beq  $7CB8
007C74 4A2D0A7B  tst.b $A7B(A5)
007C78 6724  beq  $7CA5
007C7C 4A2D0A7C  tst.b $A7C(A5)
007C80 6706  beq  $7CC0
007C84 532D0A7C  subq.b #1,$A7C(A5)

******************************************************************************
tkbrate, 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

ikbdvecs, pointers to IKBD + 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
Abacus Software

Atari ST Internals

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

---------------------------------------------------------------------------------------------------------------------

007C8E 6618  bne  $7CA8
007C90 532DA7D  subq.b  #1,$A7D(A5)
007C94 6612  bne  $7CA8
007C96 1B6DOA7FOA7D  move.b  $A7F(A5),$A7D(A5)
007C9C 10200A7B  move.b  $A7B(A5),D0
007CA0 41ED09F2  lea  $9F2(A5),A0
007CAC 610F06B  hsr  $7992
007CA8 3F2DO412  move.w  $442(A5),-(A7)
007CAB 206D0400  move.l  $4C0(A5),A0
007CB0 4E80  jsr  (A0)
007CB2 544F  addq.w  #2,A7
007CB4 4CF7FF  movem.l  (A7)+,D0-D7/A0-A6
007CB8 08B90C05FFFFFA11  bclr  #5,FFFFFA11
007CC0 4E73  rte

*************************************************

007CC2 48E7C8C0  movem.l  D0-D1/A0,-(A7)
007CC6 202DA86  movem.l  $A86(A5),D0
007CCA 6700C88  beq  $7D54
007CCC 2040  move.l  D0,A0
007CDA 102DA8A  move.b  $A8A(A5),D0
007CD4 6708  beq  $7CDE
007CD6 5390  subq.b  #1,D0
007CDE 1B4C0A8A  move.b  D0,$A8A(A5)
007CDO 6576  bra  $7D54
007CDE 1018  move.b  (A0)+,D0
007CED 632E  bmi  $7D10
007CEE 13C0FFFF8800  move.b  D0,FFFF8800
007CE8 0CD00007  cmp.b  #7,D0
007CEC 661A  bne  $7D08
007CEE 1218  move.b  (A0)+,D1

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
and.b  #S3F,D1
move.b $FFFF8800,D0
and.b  #S0C,D0
or.b  D1,D0
move.b D0,$FFFF8802
bra  $7CDE
move.b (A0)+,$FFFF8802
bra  $7CDE
addq.b  #1,D0
bpl  $7D46
cmp.b  #$81,D0
bne  $7D20
move.b (A0)+,SABB(A5)
bra  $7CDE
cmp.b  #$82,D0
bne  $7D46
move.b (A0)+,$FFFF8800
move.b (A0)+,D0
add.b  D0,SABB(A5)
move.b (A0)+,D0
move.b SABB(A5),$FFFF8802
cmp.b  SABB(A5),D0
beq  $7D50
subq.w  #1,A0
bra  $7D50
move.b (A0)+,SABB(A5)
bne  $7D50
move.w  #0,A0
move.l  A0,SABB(A5)
movem.l (A7)+,D0-D1/A0

Isolate bits 0 - 5
Read mixer
Isolate bits 6-7
OR data
Write byte in sound chip register
New sound command
Write byte directly in sound chip
Next sound command

Was command $FF ?
Yes
Was command $80 ?
No
Save byte for later
Next sound command

Command > $81 ?
Yes, set timer
Select register
Add increment-value

End value
Write value in sound chip register
End value reached ?
Yes
Sound pointer back to same command

Next value as delay timer
Sound pointer to zero
And save
Restore registers
007D58 4E75 rts

***************************************************** bellsnd, tone after CTRL G
007D5A 0034 dc.b 0,534
007D5C 0100 dc.b 1,0
007D5E 0200 dc.b 2,0
007D60 0300 dc.b 3,0
007D62 0400 dc.b 4,0
007D64 0500 dc.b 5,0
007D66 0600 dc.b 6,0
007D68 07FE dc.b 7,6FE
007D6A 0810 dc.b 8,810
007D6C 0900 dc.b 9,0
007D6E 0A00 dc.b 10,0
007D70 0B00 dc.b 11,0
007D72 0C10 dc.b 12,810
007D74 0D09 dc.b 13,9
007D76 FF00 dc.b $FF,0

***************************************************** keyclk, key click
007D78 003B dc.b 0,53B
007D7A 0100 dc.b 1,0
007D7C 0200 dc.b 2,0
007D7E 0300 dc.b 3,0
007D80 0400 dc.b 4,0
007D82 0500 dc.b 5,0
007D84 0600 dc.b 6,0
007D86 07FE dc.b 7,6FE
007D88 0810 dc.b 8,810
007D8A 0D03 dc.b 13,3
007D8C 0B80 dc.b 11,880
007D8E 0C01 dc.b 12,1

Envelope channel A
Envelope single attack
007D90 FF00  dc.b  STF, 0

******************************************************************************
007D92 0E560000  link  A6,#0
007D96 48E7070C  movem.1 D5-D7/A4-A5,-(A7)
007D98 2A6E0008  move.1 @(A6),A5
007D9E 287C0002600  move.1 #$2600,A4
007DA4 7E1E  moveq.1 #$30,D7
007DA6 6004  bra  $7DAC
007DA8 18D0  move.b (A5)+,(A4)+
007DAA 5347  subq.w #1,D7
007DAC 4A47  tst.w D7
007DDE 52F8  bgt  $7DA8
007DB0 4AD90000261A  tst.l $261A
007DB6 6708  beq  $7DC0
007DB8 20390000261A  move.1 $261A,D0
007DBE 6006  bra  $7DC6
007DCC 23C00000261A  move.1 $1604C,D0
007DCE 0C79000100002618  move.1 D0,$261A
007DD4 6306  cmp.w #1,$2618
007DD6 7CFF  bls  $7DC8
007DD8 66C00BC6  moveq.1 #-1,D0
007DDC 4A7900002618  brc  $89A0
007DE2 6704  tst.w $2618
007DE4 4240  beq  $7DE8
007DE6 6002  clr.w D0
007DE8 7001  bra  $7DEA
007DEA 13C0000025FE  moveq.1 #1,D0
007DF0 4A7900002608  move.b D0,$25FE
007DF6 6642  tst.w $2608
007DF8 6643  bne  $7E3A

Hardcopy
Save registers
Address if the parameter block
Address of the working memory
30 bytes
Put parameters in working memory
Next byte
p masks, half-tone mask
Not used ?
Load mask
Default mask
p masks
p port
Set flag
Error, stop
p port
Centronics ?
D= RA232
l= Centronics
Printer port
Height
Not zero ?
Abacus Software

Atari ST Internals

007DF9 6028
007DFA 4A79000004EE
007E00 6632
007E02 207900002600
007E08 1010
007E0A 6860
007E0C 3E80
007E0E 61000B9A
007E12 52B90000260C
007E18 4A40
007E1A 6706
007E1C 7CFF
007E1E 60000B90

bra $7E22
tst.w $9E4E
bne $7E34
move.l $2600, A0
move.b (A0), D0
ext.w D0
move.w D0, (A7)
brr $89AA
addq.l #1,$2600
tst.w D0
beg $7E32
moveq.l #-1, D0
bra $89A0
clr.w D0
move.w $2606, D0
subq.w #1,$2606
tst.w D0
bne $7DFA
clr.w D0
bra $89A0
cmp.w #3, $2616
bhs $7E4A
moveq.l #-1, D0
bra $89A0
cmp.w #1, $2610
bhs $7E5A
moveq.l #-1, D0
bra $89A0

Scrump flag, ALT HELP pressed again?
Yes, stop
blikptr
Get byte from video RAM
On the stack
Output byte
Increment blikptr
Output OK?
Yes
Set flag
Error, stop

width, screen width
Decrement width
Already zero?
No, output next byte
Flag for OK
Done

p type, Epson B/W matrix
Set flag
Error, stop

dstres, printer resolution
Set flag
Error, stop
Abacus Software

Atari ST Internals

007E5A 0C790002000026DE
007E52 6306
007E54 70FF
007E56 60000B38
007E6A 0C79000700002604
007E72 6306
007E74 70FF
007E76 60000B28
007E7A 4A790000260E
007E80 6704
007E82 4240
007E84 6002
007E86 7001
007E88 13C00004F82
007E8E 0C7900010000260E
007E96 6704
007E98 4240
007E9A 6002
007E9C 7001
007E9E 13C00004ED6
007EA4 0C7900020000260E
007EAC 6704
007EAE 4240
007EB0 6002
007EB2 7001
007EB4 13C00004ED8
007EBA 4A7900002610
007EC0 6704
007EC2 4240
007EC4 6002

cmp.w #2, $260E
bne $7E5A
moveq.l #1, D0
bra $89A0

 cmp.w #7,$260A
bne $7E7A
moveq.l #1,D0
bra $89A0

tst.w $260E
beq $7E86
clr.w D0
bra $7E68
moveq.l #1,D0
move.b D0,$4F82
cmp.w #1,$260E
beq $7E9C
clr.w D0
bra $7E9E
moveq.l #1,D0
move.b D0,$4ED6
cmp.w #2,$260E
beq $7EB2
clr.w D0
bra $7EB4
moveq.l #1,D0
move.b D0,$4ED8
tst.w $2610
beq $7EC6
clr.w D0
bra $7EC8

screen, screen resolution
Set flag
Error, stop

Test offset
Set flag
Error, stop

screen, screen resolution
Low resolution?

Flag for low resolution
screen
Medium resolution?

Flag for medium resolution
screen, screen resolution
High resolution?

Flag for high resolution
desktop, printer resolution
Test mode?
Quality mode
007EC6 7001
007EC8 13C0000042ES
007EC2 0C79000100002616
007ED6 6704
007ED8 4240
007EDA 6002
007EDC 7001
007EE3 13C0000047CE
007EE2 0C79000200002616
007EEC 6704
007EFC 6202
007EF0 6002
007EF2 7001
007EF4 13C000004F84
007EFA 0C79000300002616
007F02 6704
007F04 4240
007F06 6002
007F08 7001
007F0A 13C0000047D0
007F10 4A3900009F84
007F16 6706
007F18 70FF
007F1A 6000AA84

moveq.l #1, D0
move.b D0, $4E88
cmp.w #1, $2616
beq $7EDC
c1r.w D0
bra $7EDE
moveq.l #1, D0
move.b D0, $47CE

007F1E 4A39000047D0
007F24 670C
007F26 4A3900004EE8
007F2C 6604
007F2E 7001
007F30 6008
007F32 103900004EE8
moveq.l #1, D0
move.b $4EE8, D0

Printer mode
p type, ATARI color matrix

ATARI color dot-matrix printer
p type, ATARI daisy wheel?
Yes

Flag for ATARI daisy wheel printer
p type, Epson B/W Matrix?
Yes

Flag for Epson B/W dot matrix
ATARI daisy wheel printer?
No

Error, stop
Epson B/W dot-matrix printer?
No
Printer test mode?
No

Printer resolution
Printer resolution
Low resolution ?
No
width, 320 points per line

width
-320
Plus right
width, 320 points per line

width, 640 points per line

width
-640
Plus right
Width to 640
High resolution ?
Yes
Clear color counter
colpal

Get color
Mask irrevant bits
Save color
colpal pointer to next color
Color equal white ?
Yes
Load color
007FC2 C07C0007
007FD2 33C000035C2
007FDB 3039000047BA
007FDC EB40
007FDE C07C0007
007FE4 33C000004E6A
007FEA 3039000047BA
007FF0 E040
007FF2 C07C0007
007FF6 33C000004694
007FEC 4A39000047CE
008002 67000114
008006 3047
008008 D1C8
00800A D1FC00C04E6C
008010 30B900C04694
008016 3047
008018 D1C8
00801A 227C00004E6C
008020 30390800
008024 B07900004EDA
00802A 5F08
00802C 303900004EDA
008032 6002
008034 3047
008036 D1C8
008038 227C00004E6C
00803E 30390800
008042 3247
008044 D3C9
008046 D3FC00004E6C
00804C 3280

and.w $7, D0
move.w D0, 335C2
move.w $47BA, D0
asr.w $4, D0
and.w $7, D0
move.w D0, $4E6A
move.w $47BA, D0
asr.w $8, D0
and.w $7, D0
move.w D0, $4694
tst.b $47CE
beg $8118
move.w D7, A0
add.l A0, A0
add.l $84E6C, A0
move.w $4694, (A0)
move.w D7, A0
add.l A0, A0
move.l $84E6C, A1
move.w C(A0, A1, 1), D0
cmp.w $42ED, D0
ble $8034
move.w $42ED, D0
bra $8042
move.w D7, A0
add.l A0, A0
move.l $84E6C, A1
move.w C(A0, A1, 1), D0
move.w D7, A1
add.l A1, A1
add.l $84E6C, A1
move.w D0, (A1)
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00804E 3047</td>
<td>move.w D7,A0</td>
<td>add.l A0,A0</td>
<td>Blue level</td>
</tr>
<tr>
<td>008050 D1C8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008052 227C0004EEC</td>
<td>move.l #$4EEC,A1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008058 30309800</td>
<td>move.w @(A0,A1.l),D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00805C B079CC0035C2</td>
<td>cmp.w $35C2,D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008062 6F08</td>
<td>ble $806C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008066 3039CC0035C2</td>
<td>move.w $35C2,D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00806A 600E</td>
<td>bra $807A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00806C 3047</td>
<td>move.w D7,A0</td>
<td>add.l A0,A0</td>
<td>Red level</td>
</tr>
<tr>
<td>00806E D1C8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008070 227C0004EEC</td>
<td>move.l #$4EEC,A1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008076 30309800</td>
<td>move.w @(A0,A1.l),D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00807A 3247</td>
<td>move.w D7,A1</td>
<td>add.l A1,A1</td>
<td></td>
</tr>
<tr>
<td>00807C D3C9</td>
<td>add.l #$4EEC,A1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00807E D3FC0004EEC</td>
<td>move.w D0,(A1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008084 3280</td>
<td>move.w #$694,D0</td>
<td>add.l A1,A1</td>
<td></td>
</tr>
<tr>
<td>008088 3039CC004694</td>
<td>move.w D7,A1</td>
<td>add.l #$4EEC,A1</td>
<td></td>
</tr>
<tr>
<td>00808C 3247</td>
<td>move.w D7,A1</td>
<td>move.w (A1),D1</td>
<td></td>
</tr>
<tr>
<td>00808E D3C9</td>
<td>move.w #$4EEC,A1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008090 D3FC0004EEC</td>
<td>addq.w #1,D1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008096 3211</td>
<td>sub.w D1,D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00809A 5041</td>
<td>bgt $80A2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00809C 6204</td>
<td>clr.w D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00809E 4240</td>
<td>bra $80A4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0080A0 6002</td>
<td>moveq.l #1,D0</td>
<td></td>
<td>Red level</td>
</tr>
<tr>
<td>0080A2 7001</td>
<td>move.w D0,$4694</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0080A6 33C0004694</td>
<td>move.w $4694,D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0080AA 3039CC004EDA</td>
<td>move.w D7,A1</td>
<td>add.l A1,A1</td>
<td>Green level</td>
</tr>
<tr>
<td>0080BA 3247</td>
<td>move.w D7,A1</td>
<td>add.l #$4EEC,A1</td>
<td></td>
</tr>
<tr>
<td>0080BB 32C9</td>
<td>add.l A1,A1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0080BD D3FC0004EEC</td>
<td>add.l A1,A1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
move.w (A1), D1
addq.w #1, D1
sub.w D1, D0
bgt $80C6
clr.w D0
bra $80C8
moveql #1, D0
move.w D0, $4EDA
move.w $35C2, D0
move.w D7, A1
add.l A1, A1
add.l $84EEC, A1
move.w (A1), D1
addq.w #1, D1
sub.w D1, D0
bgt $80EA
clr.w D0
bra $80EC
moveql #1, D0
move.w D0, $35C2
move.w $4694, D0
asl.w #2, D0
move.w $4ED8, D1
asl.w #1, D1
add.w D1, D0
add.w $35C2, D0
move.w D7, A1
add.l A1, A1
add.l $84694, A1
move.w Dc, (A1)
bra $8158
move.w $4694, D0

Green level
Red level
Blue level
Red level
Green level
Blue level
Red level
00811E C1FC0012
008122 323900004EDA
008128 C3FC003B
00812C D041
00812E 3239000035C2
008134 C3FC000B
008139 D041
00813A 48C0
00813C 81FC0064
008140 3247
008142 D3C9
008144 D3FC0004E6C
00814A 3280
00814C 3047
00814E D1CB
008150 D1FC0004698
008156 4250
008158 601C
00815A 3047
00815C D1CB
00815E D1FC0004698
008164 30BC0007
008168 3047
00816A D1CB
00816C D1FC0004E6C
008172 30BCC006
008176 5247
008178 BE7C0010
00817C 6D00FE24
008180 4A3900004F82
008188 6716
008188 7004

muls.w $S1E, D0
move.w $4EDA, D1
muls.w $3B, D1
add.w D1, D0
move.w $35C2, D1
muls.w $3B, D1
add.w D1, D0
ext.l D0
divs.w $S64, D0
move.w D7, A1
add.l A1, A1
add.l $34EEC, A1
move.w D0, (A1)
move.w D7, A0
add.l A0, A0
add.l $34698, A0
clr.w (A0)
bra $8176:1ln1
move.w D7, A0
add.l A0, A0
add.l $34698, A0
move.w $7, (A0)
move.w D7, A0
add.l A0, A0
add.l $34EEC, A0
move.w $8, (A0)
addq.w $1, D7
cmp.w $310, D7
blt $7FA2
tst.b $4F82
beq $819E
moveq.l #4, D0

Times 30, 30% weighting
Times 59, 59% weighting
Times 11, 11% weighting
Green level
Blue level
Divide by 100, scaling
Increment color counter
16 colors handled already?
No, next color
Low resolution?
No
00818A 33C00004F0C
00818D 33C00004EE2
008196 33C00004768
0081A0 6038
00819E 4A3900004ED6
0081A4 6718
0081A6 7002
0081A8 33C00004F0C
0081A8 33C00004F68
0081BE 33FC00004F0C
0081BE 6010
0081C4 33FC000400004EE2
0081CE 33FC000200004F0C
0081D6 4A39000047DC
0081DC 6706
0081DE 3F3C0002
0081E2 6004
0081E4 3F3C0001
0081EC 303900004FOC
0081EE 4BC0
0081F0 81DF
0081F2 33C00004F0C
0081FA 4260
00820A 33C0000C60A
00820C 007900002606
00820E 00790000260C
00820C C0F900004768
008212 E848
008214 33C00004696
008216 303900004696
008220 C1F900004EE2

move.w D0,$4F0C
move.w D0,$4EE2
move.w D0,$4768
bra $81D6
move.l #2,D0
move.w D0,$4F0C
move.w D0,$4768
move.w #4,$4EE2
bra $81D6:lnl
move.w #1,$4768
move.w #9,$4EE2
move.w #2,$4F0C
tst.b $47DD
beq $81E4
move.w #2,-(A7)
bra $81E8
move.w #1,-(A7)
move.w $4F0C,D0
ext.l D0
div.s (A7)+,D0
move.w D0,$4F0C
clt.w D0
move.w $260A,D0
add.w $260D,D0
add.w $260C,D0
mul.w $4768,D0
lsr.w #4,D0
move.w D0,$4696
move.w $4696,D0
mul.w $4EE2,D0

Medium resolution?
No

Epson B/W dot-matrix printer?
No

Left
Height
Right
008226 3300003E80 move.w DO,$3E80
00822C 20390002600 move.l $2600,DO
008232 C05CFFFFFE and.l #$FFFFFFFE,DO
008238 23C0000046B8 move.l DO,$46B8
00823E 20390002600 move.l $2600,DO
008244 00B9000046B8 cmp.l $46B8,DO
00824A 660A bne $8256
00824C 4240 clr.w DO
00824E 303900002604 move.w $2604,DO
008254 600A bra $8260
008256 4240 clr.w DO
008258 3039000C2604 move.w $2604,DO
00825E 5060 addq.w #0,DO
008260 33C000047BC move.w DO,$47BC
008266 427900012EAD clr.w $12EA
00826C 6000071E bra $996
008270 4A7900004EE tst.w $4EE
008276 6600071E bne $8996
dumpflag, ALT HELP pressed again?
00827A 13FC0010003628 move.b #1,$3628
008282 4240 clr.w DO
008284 303900002606 move.w $2606,DO
00828A C0F00004768 mulu.w $4768,DO
008290 EE48 lsr.w #0,DO
008292 9C7900004768 sub.w $4768,DO
008298 E348 lsl.w #1,DO
00829A 4B40 swap DO
00829C 4240 clr.w DO
00829E 4B40 swap DO
0082A0 00B9000046B8 add.l $46B8,DO
0082A6 23C000004EDC move.l DO,$4EDC
0082AC 706F moveq.l #15,DO
0082AE 4241 clr.w DO

Pointer to video RAM
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0082B0</td>
<td>move.w</td>
<td>$2606,D1</td>
</tr>
<tr>
<td>0082B6</td>
<td>and.w</td>
<td>$8F,D1</td>
</tr>
<tr>
<td>0082BA</td>
<td>sub.w</td>
<td>D1,D0</td>
</tr>
<tr>
<td>0082BC</td>
<td>move.w</td>
<td>D0,$4F12</td>
</tr>
<tr>
<td>0082C2</td>
<td>move.w</td>
<td>$2606,$3E2C</td>
</tr>
<tr>
<td>0082CC</td>
<td>bra</td>
<td>$83F2</td>
</tr>
<tr>
<td>0082D0</td>
<td>clr.w</td>
<td>D0</td>
</tr>
<tr>
<td>0082D2</td>
<td>move.w</td>
<td>$2608,D0</td>
</tr>
<tr>
<td>0082D8</td>
<td>sub.w</td>
<td>$12EA,D0</td>
</tr>
<tr>
<td>0082DE</td>
<td>swap</td>
<td>D0</td>
</tr>
<tr>
<td>0082E0</td>
<td>clr.w</td>
<td>D0</td>
</tr>
<tr>
<td>0082E4</td>
<td>swap</td>
<td>D0</td>
</tr>
<tr>
<td>0082EC</td>
<td>dlvu.w</td>
<td>$4EE2,D0</td>
</tr>
<tr>
<td>0082EA</td>
<td>beg</td>
<td>$82F4</td>
</tr>
<tr>
<td>0082E2</td>
<td>move.w</td>
<td>$4EE2,D0</td>
</tr>
<tr>
<td>0082F2</td>
<td>bra</td>
<td>$8392</td>
</tr>
<tr>
<td>0082F4</td>
<td>clr.w</td>
<td>D0</td>
</tr>
<tr>
<td>0082F6</td>
<td>move.w</td>
<td>$2608,D0</td>
</tr>
<tr>
<td>0082FC</td>
<td>sub.w</td>
<td>$12EA,D0</td>
</tr>
<tr>
<td>008302</td>
<td>move.w</td>
<td>D0,$48D2</td>
</tr>
<tr>
<td>008306</td>
<td>move.l</td>
<td>$4EDC,$493C</td>
</tr>
<tr>
<td>008312</td>
<td>clr.w</td>
<td>D7</td>
</tr>
<tr>
<td>008314</td>
<td>bra</td>
<td>$83B4</td>
</tr>
<tr>
<td>008318</td>
<td>clr.w</td>
<td>$4F1A</td>
</tr>
<tr>
<td>00831E</td>
<td>move.w</td>
<td>$1,$4F0E</td>
</tr>
<tr>
<td>008326</td>
<td>move.l</td>
<td>$493C,$47BE</td>
</tr>
<tr>
<td>008330</td>
<td>clr.w</td>
<td>D6</td>
</tr>
<tr>
<td>008332</td>
<td>bra</td>
<td>$8364</td>
</tr>
<tr>
<td>008334</td>
<td>move.l</td>
<td>$47BE,A0</td>
</tr>
<tr>
<td>00833A</td>
<td>move.w</td>
<td>(A0),D0</td>
</tr>
<tr>
<td>00833C</td>
<td>moveq.l</td>
<td>#15,D1</td>
</tr>
<tr>
<td>00833E</td>
<td>sub.w</td>
<td>$4F12,D1</td>
</tr>
</tbody>
</table>
008344 E260  eor.w  D1,DC
008346 C07C0001  and.w  #1,DC
00834A C1F900004F0E  muls.w  $4F0E,D0
008350 D17900004F1A  add.w  D0,4F1A
008356 54B8000047BE  adqu.l  #2,$47BE
00835C 51F900004F0E  asl.w  $4F0E
008362 5246  addqu.w  #1,D6
008368 BC7900004769  cmpl.w  $4769,D6
00836E 6DC8  blt  $8334
00836C 4A3900004ED8  tst.b  $4ED8  High resolution?
008372 6712  beq  $8386
008374 4A7900004F1A  tst.w  $4F1A
00837A 6708  beq  $8384
00837C 423900003628  clrl.b  $3628
008382 603A  bra  $83BE
008384 601C  bra  $83A2
008386 307900004F1A  move.w  $4F1A,A0
008388 D1C8  add.l  A0,A0
00838E D1FC00004EEC  add.l  $4EEC,A0
008394 0C500008  cmpl.w  $0,(A0)
008398 6708  beq  $83A2
00839A 423900003628  clrl.b  $3628
0083A0 601C  bra  $83BE
0083A2 303900004696  move.w  $4696,D0
0083A8 E34C  asl.w  #1,D0
0083AA 48C0  ext.l  D0
0083AC D1B90000493C  add.l  D0,$493C
0083B2 5247  addqu.w  #1,D7
0083B6 BE7900004ED2  cmpl.w  $4ED2,D7
0083BA 6D00FF5C  blt  $8318
0083BE 4A3900003628  tst.b  $3628
0083C4 6736  beq  $83FC
Epson B/W dot-matrix printer?

No
ATARI color dot matrix printer?
No
High resolution?
Yes

ESC 'X' 5
Output string to the printer
Error?
No
Flag for error
Error, terminate

ESC 'X' 5
Output string to printer
Error?
No
Flag for error
Error, terminate

ESC 'X' 3
Output string to printer
Error?
No
Flag for error

008446 427900004F88
cir.w $4F88
00844C 6C000450
bga $689E
008450 4A3900004ED8
tst.b $4ED8
008456 675A
beq $84A2
008458 4A390004ED8
tst.b $4ED8
00845E 6652
bne $84B2
008460 4A790004F88
tst.w $4F88
008466 6616
bne $847E
008468 2FBC00016D5E
move.l $816D5E,(A7)
00846E 61000582
bca $89FE
008472 4A4D
tst.w D0
008474 6706
beq $847C
008476 70FF
moveq.l $-1,D0
008478 60000526
bra $89A0
00847C 5034
bra $84B2
00847E CC79000100004F88
cmp.w #1,$4F88
008486 6616
bne $849E
008488 2FBC0016D63
move.l #(16D63),D0
00848E 6100056E
bca $89FE
008492 4A4D
tst.w D0
008494 6706
beq $849C
008496 70FF
moveq.l $-1,D0
008498 60000526
bra $89A0
00849C 5014
bra $84B2
00849E 2FBC0016D68
moveq.l #(16D68),D0
0084A4 61000538
bca $89FE
0084A8 4A4D
tst.w D0
0084AA 6706
beq $84B2
0084AC 70FF
moveq.l $-1,D0

Abacus Software
Atari ST Internals

398
<table>
<thead>
<tr>
<th>Address</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0084AE 60000010</td>
<td>bra $89A0</td>
</tr>
<tr>
<td>0084B2 4A39000017D0</td>
<td>tst b $47D0</td>
</tr>
<tr>
<td>0084B8 6700</td>
<td>beq $64C2</td>
</tr>
<tr>
<td>0084BA 28BC0016D6D</td>
<td>move l #$16D6D, (A7)</td>
</tr>
<tr>
<td>0084C0 6006</td>
<td>bra $84C9</td>
</tr>
<tr>
<td>0084C2 28BC0016D71</td>
<td>move l #$16D71, (A7)</td>
</tr>
<tr>
<td>0084C8 610000534</td>
<td>bsr $89FE</td>
</tr>
<tr>
<td>0084CC 4A10</td>
<td>tst w DO</td>
</tr>
<tr>
<td>0084CE 6706</td>
<td>beq $84D6</td>
</tr>
<tr>
<td>0084D0 70FF</td>
<td>move q l #$1, DO</td>
</tr>
<tr>
<td>0084D2 600004CC</td>
<td>bra $89AD</td>
</tr>
<tr>
<td>0084D6 103900003E86</td>
<td>move b #$3E86, DO</td>
</tr>
<tr>
<td>0084DC 6880</td>
<td>ext w DO</td>
</tr>
<tr>
<td>0084DE 3E80</td>
<td>move w DO, (A7)</td>
</tr>
<tr>
<td>0084EO 610004C8</td>
<td>bsr $89AA</td>
</tr>
<tr>
<td>0084F0 4A40</td>
<td>tst w DO</td>
</tr>
<tr>
<td>0084F6 6706</td>
<td>beq $84EE</td>
</tr>
<tr>
<td>0084FE 70FF</td>
<td>move q l #$1, DO</td>
</tr>
<tr>
<td>008500 600004B4</td>
<td>bra $89AD</td>
</tr>
<tr>
<td>008506 103900003E88</td>
<td>move b #$3E88, DO</td>
</tr>
<tr>
<td>00850E 4880</td>
<td>ext w DO</td>
</tr>
<tr>
<td>008516 3E80</td>
<td>move w DO, (A7)</td>
</tr>
<tr>
<td>00851F 610004B0</td>
<td>bar $89AA</td>
</tr>
<tr>
<td>008524 4A40</td>
<td>tst w DO</td>
</tr>
<tr>
<td>00852E 6706</td>
<td>beq $8506</td>
</tr>
<tr>
<td>008530 70FF</td>
<td>move q l #$1, DO</td>
</tr>
<tr>
<td>008532 6000049C</td>
<td>bra $89A0</td>
</tr>
<tr>
<td>008536 13FC00010004EAA</td>
<td>move b #$1, $4EEA</td>
</tr>
</tbody>
</table>

Error, terminate

Epson B/W dot-matrix printer?
No
ESC 'L', Bit image 960 points/line
Output string to printer
Error?
No
Flag for error
Error, terminate

Get byte
Low byte of the number
On stack
Output
Error?
No
Flag for error
Error terminate

Get byte
High byte of the number
On stack
Output
Error?
No
Flag for error
Error, terminate
00850E 23F9000046B80000EDC move.l $46B8,$4EDC
008518 33F9000047BC0000F12 move.w $47BC,$4F12
008522 4279000012E8 clr.w $12E8
008526 60000036C bra $8876

00852C 4247 clr.w D7
00852E 60DC 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 6DBE
008542 4247 clr.w D7
008544 6010 bra $8556
008546 3047 move.w D7,A0
008548 D1C8 add.l A0,A0
00854A D1FC00003B8A
008550 30BC0007 move.w #7,(A0)
008554 5247 addq.w #1,D7
008556 BE7C0004 cmp.w #4,D7
00855A 6DEA
00855C 4240 clr.w D0
00855E 303900002608 move.w $2608,D0
008564 979000012EA sub.w $12EA,D0
008568 4940 swap D0
00856C 4240 clr.w D0
00856E 4840 swap D0
008570 80F900004EE2 divu.w $4EE2,D0
008574 6708 beq $8560
008578 303900004EE2 move.w $4EE2,D0
00857C 600E bra $8560
008580 4240  clr.w D0
008582 303900002508  move.w $2608,D0
008586 9079000012EA  sub.w $12EA,D0
00858E 33C0000004ED2  move.w D0,$4ED2
008594 23F9000004EDC0000493C  move.l $4EDC,$493C
008598 4247  clr.w D7
00859C 600000F8  bra $869A

0085A0 427900004F1A  clr.w $4F1A
0085A4 33FC000000004F6E  move.w #1,9FCE
0085A8 23F9000004F6C000043BE  move.l $493C,$47BE
0085AC 4246  clr.w D6
0085B0 6030  bra $85F0
0085B4 2079000047BE  move.l $47BE,A0
0085BC 301D  move.w (A0),D0
0085C0 720F  moveq.l $15,D1
0085C8 927900004F12  sub.w $4F12,D1
0085D0 626D  asz.w D1,D0
0085D4 C07C0001  and.w #1,D0
0085D8 1F9C00004F0E  multi.w $4F0E,D0
0085DC 517900004F1A  add.w D0,$4F1A
0085E0 54B9000047BE  addq.l $2,547BE
0085E8 1F9C00004F0E  asl.w $4F0E
0085E8 5246  addq.w #1,D6
0085F0 BC7900004768  cmp.w $4768,D6
0085F4 6DC8  b1t $85C0
0085F8 4A3900004ED8  tst.b $4ED8
0085FE 6722  beq $8622
008600 4A7900004F1A  tst.w $4F1A
008606 670C  beq $8614
00860C 20790000251A  move.l $261A,A0
008610 101D  move.b (A0),D0

Height

High resolution?

No

p masks
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>008610</td>
<td>4880</td>
<td>ext.w DD</td>
</tr>
<tr>
<td>008612</td>
<td>6002</td>
<td>bra $8616</td>
</tr>
<tr>
<td>008614</td>
<td>4240</td>
<td>clr.w DD</td>
</tr>
<tr>
<td>008616</td>
<td>3247</td>
<td>move.w D7, A1</td>
</tr>
<tr>
<td>008619</td>
<td>D3FC000047D4</td>
<td>add.l $547D4, A1</td>
</tr>
<tr>
<td>00861E</td>
<td>1280</td>
<td>move.b D0, (A1)</td>
</tr>
<tr>
<td>008620</td>
<td>6066</td>
<td>bra $8688</td>
</tr>
<tr>
<td>008622</td>
<td>3047</td>
<td>move.w D7, A0</td>
</tr>
<tr>
<td>008624</td>
<td>D1C8</td>
<td>add.l A0, A0</td>
</tr>
<tr>
<td>008626</td>
<td>D1FC0003E8A</td>
<td>add.l $83E8A, A0</td>
</tr>
<tr>
<td>00862C</td>
<td>327900004F1A</td>
<td>move.w $4F1A, A1</td>
</tr>
<tr>
<td>008632</td>
<td>D3C9</td>
<td>add.l A1, A1</td>
</tr>
<tr>
<td>008634</td>
<td>D3FC0004698</td>
<td>add.l $84698, A1</td>
</tr>
<tr>
<td>00863A</td>
<td>3091</td>
<td>move.w (A1), (A0)</td>
</tr>
<tr>
<td>00863C</td>
<td>3047</td>
<td>move.w D7, A0</td>
</tr>
<tr>
<td>00863E</td>
<td>D3C8</td>
<td>add.w A0, A0</td>
</tr>
<tr>
<td>008640</td>
<td>D1FC00047D4</td>
<td>add.l $547D4, A0</td>
</tr>
<tr>
<td>008646</td>
<td>327900004F1A</td>
<td>move.w $4F1A, A1</td>
</tr>
<tr>
<td>00864C</td>
<td>D3C9</td>
<td>add.l A1, A1</td>
</tr>
<tr>
<td>00864E</td>
<td>D3FC0004EEC</td>
<td>add.l $84EEC, A1</td>
</tr>
<tr>
<td>008654</td>
<td>3251</td>
<td>move.w (A1), A1</td>
</tr>
<tr>
<td>008656</td>
<td>D2C9</td>
<td>add.w A1, A1</td>
</tr>
<tr>
<td>008658</td>
<td>D3F900CD261A</td>
<td>add.l $261A, A1</td>
</tr>
<tr>
<td>00865E</td>
<td>1091</td>
<td>move.b (A1), (A0)</td>
</tr>
<tr>
<td>008660</td>
<td>3047</td>
<td>move.w D7, A0</td>
</tr>
<tr>
<td>008662</td>
<td>D3C8</td>
<td>add.w A0, A0</td>
</tr>
<tr>
<td>008664</td>
<td>D1FC00047D4</td>
<td>add.l $547D4, A0</td>
</tr>
<tr>
<td>00866A</td>
<td>327900004F1A</td>
<td>move.w $4F1A, A1</td>
</tr>
<tr>
<td>008670</td>
<td>D3C9</td>
<td>add.l A1, A1</td>
</tr>
<tr>
<td>008672</td>
<td>D3FC0004EEC</td>
<td>add.l $84EEC, A1</td>
</tr>
<tr>
<td>008678</td>
<td>3251</td>
<td>move.w (A1), A1</td>
</tr>
</tbody>
</table>

*mask*
ABACUS Software

Atari ST Internals

C
M
G.
X
!8
C
O
-H
O
O1
ffl
m
O>
ffl
O
-H 0>
390C00493C
add.l D0, $493C
008698 5247
addq.w #1, D7
00869B BB79C004ED2
cmp.w $4ED2, D7
0086A0 6DCCF02
blt $85A4
0086A4 4A390C0047CE
tst.b $47CE
0086AA 670000DE
beq $878A
0086AE 4A390C004ED8
tst.b $4ED8
0086B4 660000D4
bne $878A
0086B8 4247
clr.w D7
0086BA 6DCCF0C4
bra $8780
0086BE 423900004EE0
clr.b $4EE0
0086C4 4A7900004FB8
tst.w $4F80
0086CA 6624
bne $86F0
0086CC 3047
move.w D7, A0
0086CE D1C8
add.l A0, A0
0086D0 227C00003E8A
move.l $3E8A, A1
0086D6 30300900
move.w 0(A0, A1, 1), D0
0086DA 48C0
ext.l D0
0086DC 81FC0002
divs.w $52, D0
0086E0 4840
swap D0
0086E2 4A40
tst.w D0
0086E4 6708
beq $86EE
0086E6 13FC00100004EE0
move.b $1, $4EE0
0086EE 606C
bra $875C
0086F0 0C79C00100004F88
cmp.w $1, $4F88
0086F8 664A
bne $8744

p masks

ATARI color dot-matrix printer?
No
High resolution?
Yes
0086FA 3047
0086FC D1C8
0086FE D1FC00003E8A
008704 0C500002
008708 6730
00870A 3047
00870C D1C8
00870E D1FC00003E8A
008714 0C500003
008718 6720
00871A 3047
00871C D1C8
00871E D1FC00003E8A
008724 0C500006
008728 6710
00872A 3047
00872C D1C8
00872E D1FC00003E8A
008734 0C500007
008738 6608
00873A 13FC000100004EEO
008742 6018
008744 3047
008746 D1C8
008748 D1FC00003E8A
00874E 0C500003
008752 6F08
008754 13FC000100004EEO
00875C 4A3900004EEO
008762 671A
008764 3047
008766 D0C8

move.w D7, A0
add.l A0, A0
add.l #$3E8A, A0
cmp.w #2, (A0)
beq $873A
move.w D7, A0
add.l A0, A0
add.l #$3E8A, A0
cmp.w #3, (A0)
beq $873A
move.w D7, A0
add.l A0, A0
add.l #$3E8A, A0
cmp.w #4, (A0)
beq $873A
move.w D7, A0
add.l A0, A0
add.l #$3E8A, A0
cmp.w #5, (A0)
bne $8742
move.b #$1, $4EEO
bra $875C
move.w D7, A0
add.l A0, A0
add.l #$3E8A, A0
cmp.w #6, (A0)
blo $875C
move.b #$1, $4EEO
tst.b $4EEO
beq $8775
move.w D7, A0
add.w A0, A0
008768 D1FC000047D4  add.l  #47D4, A0
00876E 4210  clrb  (A0)
008770 3047  move.w D7, A0
008772 DCC8  add.w A0, A0
008774 D1FC00047D4  addl  #47D4, A0
00877A 422800C1  clrb  #1(A0)
00877E 5247  addq.w #1, D7
008780 B87900004ED2  cmp.w $4ED2, D7
008786 6D00FF36  blt  $86BE
00879A 7E04  moveq.l #4, D7
00879C 60000086  bra  $8614
00879E 423900035BE  clrb  $35BE
00879F 33FC008000004F10  move.w #$0080, #4F10
00879E 4246  clrw  D6
0087AC 603E  bra  $87E0
0087AE 7207  moveq.l #7, D1
0087B6 9247  sub.w D7, D1
0087B2 E280  asr.w D1, D0
0087B4 C07C0001  and.w #$1, D0
0087BB 1IF800004F10  muls.w #$4F10, D0
0087BE 123900035BE  move.b $35BE, D1
0087C4 D200  add.b D0, D1
0087C6 3C10000035BE  move.b D1, $35BE
0087CC 303900004F10  move.w #$4F10, D0
0087D2 48C0  ext.l D0
0087D4 81FC00002  divu.w #$2, D0
0087DC 33C000004F10  move.w D0, #$4F10
0087DE 5245  addq.w #1, D6
0087E0 BC7C0008  cmp.w #493, D6
bit $87A2
move.b $35BE, D0
lea $87AE, D0
move.w D0, {A7}
b.beq $07FB
moveq.l #-1, D0
bra $89A0
Output

$tst.b $4EEA
beq $880A
c.l.w D0
bra $880C
moveq.l #1, D0
move.b D0, $4EEA
addq.w #1, D7
move.w $4FOC, D0
addq.w #1, D0
cmp.w D0, D7
blt $8790
$tst.b $47D0
beq $884A
$tst.b $4EEA
beq $884A
move.b $35BE, D0
lea $87AE, D0
move.w D0, {A7}
moveq.l #1, D0
Output
Epson B/W dot-matrix printer?
No

$tst.b $4EEA
beq $884A
$tst.b $4EEA
beq $884A
move.b $35BE, D0
lea $87AE, D0
move.w D0, {A7}
moveq.l #1, D0
Output
OK?
Yes
Set flag
Abacus Software

Atari ST Internals

H

CO 0000158

bra $89A0

Error, terminate

00884A 527900004F12
addq.w #1,$4F12
cmp.w $15,$4F12
ble $887C
move.w $4768,D0
asl.w #1,D0
ext.l D0
add.l D0,$4EDC
cld.w $4F12
addq.w #1,$512E8
move.w $12E8,D0
cmp.w $3E2C,D0
bit $852C
move.w #$6D,(A7)
bsr $89A0

tst.w D0
beq $8898
movel.l #$l,D0
bra $89A0

Error, terminate

CR
Output
OK?
Yes
Set flag

ATARI color dot-matrix printer?
No
High resolution?
Yes

ESC '3' 1, 1/216" line spacing
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0088C4</td>
<td>bsr $89FE</td>
<td>Output string to printer</td>
<td></td>
</tr>
<tr>
<td>0088C8</td>
<td>tst.w D0</td>
<td>Set flag</td>
<td></td>
</tr>
<tr>
<td>0088CA</td>
<td>beq $88D2</td>
<td>Error, terminate</td>
<td></td>
</tr>
<tr>
<td>0088CC</td>
<td>moveq.l #1,-1,DO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088CE</td>
<td>bra $89A0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088D2</td>
<td>move.w #$A,(A7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088D6</td>
<td>bsr $89AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088DA</td>
<td>tst.w D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088DC</td>
<td>beq $88E4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088E0</td>
<td>moveq.l #1,-1,DO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088E4</td>
<td>bra $89A0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088EA</td>
<td>addq.w #1,$47D2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088F0</td>
<td>tst.b $4EE8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088F2</td>
<td>beq $88F6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088F4</td>
<td>moveq.l #1,-1,DO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088F6</td>
<td>bra $88F9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088F8</td>
<td>moveq.l #2,-1,DO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0088FE</td>
<td>cmp.w $47D2,D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008902</td>
<td>hlt $8446</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008906</td>
<td>tst.b $4EE8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00890A</td>
<td>beq $8948</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00890C</td>
<td>cir.w D7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00890E</td>
<td>bra $8996</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008910</td>
<td>moveq.l $16D7A,(A7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008914</td>
<td>bsr $89FE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>008918</td>
<td>tst.w D0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00891A</td>
<td>beq $8922</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00891C</td>
<td>moveq.l #1,-1,DO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00891E</td>
<td>bra $89A0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- Output string to printer
- Set flag
- Error, terminate
- LF
- Output
- OK?
- Yes
- Error, terminate
- Printer resolution
- Printer resolution
- ESC '3' 1, 1/216 line spacing
- Output string to printer
- Error on output?
- No
- Set flag
- Error, terminate
Abacus Software
Atari ST Internals

008922 3EBC000A
008926 61000082
00892A 4A40
00892C 6706
00892E 70FF
008930 6000000E

008934 5247
008936 4A39000047D0
00893C 6704
008940 7002
008942 6C02
008944 7C80
008946 6DC6
008948 2E8CD0016D7F
00894E 610000A2
008952 4A40
008954 6704
008956 70FF
008958 6046

00895A 3EBC000A
00895E 614A
008960 4A40
008962 6704
008964 70FF
008966 6038
008968 303900003E80
00896E 2340
008970 48C0

move.w $A, (A7)
bsr $89AA
tst.w D0
beq $8934
moveq.l #1, D0
bra $89A0
addq.w #1, D7
tst.b $47D0
beq $8942
moveq.l #2, D0
bra $8944
moveq.l #1, D0
cmp.w D0, D7
blt $890E
move.l $816D7F, (A7)
bsr $89FE
tst.w D0
beq $895A
moveq.l #1, D0
bra $89A0
move.w $A, (A7)
bsr $89AA
tst.w D0
beq $8968
moveq.l #1, D0
bra $89A0
move.w $3E80, D0
asl.w #1, D0
ext.l DO

LF
Output
Error during output?
No
Set flag
Error, terminate

Epson B/W dot-matrix printer?
No

ESC 'l', 7/12" line spacing
Output string to printer
Error during output?
No
Set flag
Error, terminate

LF
Output
Error during output?
No

Error, terminate
008972 D1B9000046B8
008978 303900004EE2
00897E D179000012EA
008984 4240
008986 303900002608
00898C B079000012EA
008992 62007BCD
008996 2EB00016D83
00899C 6160
00899E 4240
0089A0 4A9F
0089A2 4CDF30C0
0089A6 4E5E
0089A8 4275

add.l D0,$46B8
move.w $4EE2,D0
add.w D0,$12EA
clr.w D0
move.w $2608,D0
cmp.w $12EA,D0
bhi $8270
move.l #816D83,(A7)
bar $89FE
clr.w D0
tst.l (A7)+
movem.1 (A7)+,D6-D7/A4-A5
unlk A6
rts

******************************
0089AA 4E56FFFC
0089AE 4A39000025FE
0089B4 6722
0089B6 102E0009
0089BA 4880
0089BC 3E80
0089BE 102E0009
0089C2 4880
0089C4 3F00
0089C6 4EB900008A2C
0089CC 548F
0089CE 4A4C
0089D0 6604
0089D2 70FF
0089D4 6024
0089D6 6020

link A6,4-4
tst.b $25F2
beq $89D8
move.b 9(A6),D0
ext.w D0
move.w D0,(A7)
move.b 9(A6),D0
ext.w D0
move.w D0,-(A7)
jar $88A2C
addq.l #2,A7
tst.w D0
bne $89D6
moveq.l #1,D0
bra $89FA
bra $89F0

ESC '2', 1/6" line spacing
Output string to printer
Flag for OK
Restore registers
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
Character to output
Extend to word
And on stack
Character to output
Extend to word
And back on stack (?)
Output via RS-232
Correct stack pointer
OK?
Yes
Set error flag

Output string to printer
Get string address
String character
Extend to word
Output character
Pointer to next character
Error-free output?
Yes
Error
Terminate output
Pointer to string
$FF as end indicator
Centronics output
Character to output
Save registers
Character to output
Clear A5
Output character to Centronics port
Correct stack pointer
Restore registers

RS232 output
Character to output
Save registers
Character to output
Clear A5
Output character via RS-232
Correct stack pointer
Restore registers

VDI ESCAPE functions
Pointer to CONTRL array
Function number
Greater than 19?
Abacus Software

Atari ST Internals

**08A72** 307BC00A
**08A76** D1FC00008C7A
**08A7C** 4ED0

`move.w $8A7E(PC,D0.w),A0`  
`add.l $8C7A,A0`  
`jmp {A0}`

Get relative address from table  
Add base address  
Execute routine

0, rts
1, Inquire addressable alpha character cells
2, Exit alpha mode
3, Enter alpha mode
4, Alpha cursor up
5, Alpha cursor down
6, Alpha cursor right
7, Alpha cursor left
8, Home alpha cursor
9, Erase to end of alpha screen
10, Erase to end of alpha text line
11, Direct alpha cursor address
12, Output cursor addressable alpha text
13, Reverse video on
14, Reverse video off
15, Inquire current alpha cursor address
16, Inquire tablet status
17, Hardcopy
18, Place graphic cursor at location
19, Remove last graphic cursor

**08A9E** 0114
**08A96** 0120
**08A98** 014E
**08A9A** 0158
**08A9C** 0162
**08A9E** 018C
**08AA0** 0002
**08AA2** 01A4
**08AA4** 01B4

dc.w $8C7A-$8C7A

do.w $8C82-$8C7A

do.w $8C8C-$8C7A

do.w $8C8E-$8C7A

do.w $8C94-$8C7A

do.w $8CA8-$8C7A

do.w $8CC2-$8C7A

do.w $8CDC-$8C7A

do.w $8CF0-$8C7A

do.w $8CP0-$8C7A

do.w $8D24-$8C7A

do.w $8D8E-$8C7A

do.w $8DA2-$8C7A

do.w $8DC8-$8C7A

do.w $8DD2-$8C7A

do.w $8DDC-$8C7A

do.w $8EO6-$8C7A

do.w $8C7C-$8C7A

do.w $8E1E-$8C7A

do.w $8E2E-$8C7A

ESC VDI 101 ?
ESC VDI 102 ?
Yes, initialize font data
008AB6 61000448
008ABA 207900002584
008AC0 3010
008AC2 COF90000257E
008AC9 33C00000255E
008ACE 6000041E

bsr $8F00
move.l $2584,A0
move.w (A0),DO
mulu.w $257E,DO
move.w DO,$255E
bra $8EEE

008AD2 322F0006 move.w 6(A7),D1
008AD6 024100FF and.w #$FF,D1
008ADA 600005D2 bra $90AE

008AD8 322F0006
008AEC 4EDO
move.w 6(A7),D1
move.l $4A8,A0
jmp (A0)

008AEE B27C0020 cmp.w #$20, D1
008AF2 6C0005BA bge $8CAE
008AF6 B23C001B
008AFA 660C
008AFC 23FC000084A00004A8 move.l #$8B4A,$4A8
008B06 6E75 rts

008B08 5F41
008B0A 6B22
008B0C B27C0006
008B10 6E1C

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
Abacus Software

**Atari ST Internals**

**008B12** E349
**008B14** 3079100A
**008B18** D1FC0008B30
**008B1E** 4E0C

---

**lsl.w #1, D1**
**move.w $8B20(FC,D1.w),A0**
**add.l $8B30,A0**
**jmp (A0)**

---

**Jump table for control codes**

7, BEL
8, BS
9, TAB
10, LF
11, VT
12, FF
13, CR

---

**Bring to word processing**
**Get relative address from table**
**Add base address to it**
**And execute corresponding routine**

---

**BEL**
**Output tone**

---

**TAB**
**Cursor column**
**Convert to number divisible by 8**
**And add 8**
**Cursor line**
**Reset cursor**

---

**Process character after ESC**
**concat vector back to standard**
**Minus 'A'**
**Less, then ignore**

**'H**
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 access
Get relative address from table
Add base address
And execute routine
008BBE E349  lsl.w $1,D1
008BC0 307B1064  move.w $8C26(PC,D1.w),A0
008BC4 D1FC0008BB2E  add.l $8BB2E,A0
008BC8 4ED0  jmp (A0)

008BCC 23FC0008BD800004A8  move.l #$8BDS,$4A8
008BD6 4E75  rts

008BD8 23FC0008AEED00004A8  move.l #$8AAE,$4A8
008BE2 927C0020  sub.w $520,D1
008BE6 3001  move.w D1,D0
008BEB 60000292  bra $8E7C

008BEC 23FC0008BF800004A8  move.l #$8BF9,$4A8
008BF6 4E75  rts

008BF8 23FC0008AEED00004A8  move.l #$8AAE,$4A8
008C02 927C0020  sub.w $520,D1
008C06 3001  move.w D1,D0
008C0B 6000027E  bra $8E88

008CC0 0166  dc.w $8B2E-$8B2E
008CC2 017A  dc.w $8B2E-$8B2E
008CC6 0194  dc.w $8B2E-$8B2E
008CC2 01AE  dc.w $8B2E-$8B2E
008CC4 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 standard 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
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>ASCII Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008C16 0000</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC t, rts</td>
<td></td>
</tr>
<tr>
<td>008C18 0000</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC r, rts</td>
<td></td>
</tr>
<tr>
<td>008C1A 01C2</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC H</td>
<td></td>
</tr>
<tr>
<td>008C1C 0306</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC I</td>
<td></td>
</tr>
<tr>
<td>008C1E 01CA</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC J</td>
<td></td>
</tr>
<tr>
<td>008C20 01F6</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC K</td>
<td></td>
</tr>
<tr>
<td>008C22 0320</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC L</td>
<td></td>
</tr>
<tr>
<td>008C24 033E</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC M</td>
<td></td>
</tr>
</tbody>
</table>

Address table for ESC lowercase:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>ASCII Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008C26 009E</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC b</td>
<td></td>
</tr>
<tr>
<td>008C28 00BE</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC c</td>
<td></td>
</tr>
<tr>
<td>008C2A 0366</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC d</td>
<td></td>
</tr>
<tr>
<td>008C2C 0382</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC e</td>
<td></td>
</tr>
<tr>
<td>008C2E 03D2</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC f</td>
<td></td>
</tr>
<tr>
<td>008C30 0000</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC g, rts</td>
<td></td>
</tr>
<tr>
<td>008C32 0000</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC h, rts</td>
<td></td>
</tr>
<tr>
<td>008C34 0000</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC i, rts</td>
<td></td>
</tr>
<tr>
<td>008C36 03F2</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC j</td>
<td></td>
</tr>
<tr>
<td>008C38 040E</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC k</td>
<td></td>
</tr>
<tr>
<td>008C3A 0428</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC l</td>
<td></td>
</tr>
<tr>
<td>008C3C 0000</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC m, rts</td>
<td></td>
</tr>
<tr>
<td>008C3E 0000</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC n, rts</td>
<td></td>
</tr>
<tr>
<td>008C40 0446</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC o</td>
<td></td>
</tr>
<tr>
<td>008C42 029A</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC p</td>
<td></td>
</tr>
<tr>
<td>008C44 02A4</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC q</td>
<td></td>
</tr>
<tr>
<td>008C46 000C</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC r, rts</td>
<td></td>
</tr>
<tr>
<td>008C48 000C</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC s, rts</td>
<td></td>
</tr>
<tr>
<td>008C4A 000C</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC t, rts</td>
<td></td>
</tr>
<tr>
<td>008C4C 000C</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC u, rts</td>
<td></td>
</tr>
<tr>
<td>008C52 048C</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC v</td>
<td></td>
</tr>
<tr>
<td>008C50 0496</td>
<td>dc.w $8B2E-$8B2E</td>
<td>ESC w</td>
<td></td>
</tr>
</tbody>
</table>
**VDI ESC 1, get screen size**

Pointer to CONTRO 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 A, Cursor up, VDI ESC 4**

Cursor line

Zero, then done already
Subtract one
Get cursor column
And set cursor

ESC B, Cursor down, VDI ESC 5
Cursor line
Compare to maximum cursor line
Already in lowest line, done
Increment by one
Cursor column
Set cursor

ESC C, Cursor right, VDI ESC 6
Cursor column
Compare with maximum value (79)
Increment by one
Get cursor line
And set cursor

ESC D, DEL, Cursor left, VDI ESC ?
Cursor column
Already zero ?
Decrement by one
Cursor line
Set cursor

ESC H, Cursor Home, VDI ESC 8
Column
And line to zero
Set cursor
bcr $8D24
move.w $2562,D1
        cmp.w $2552,D1
        beq $8C7A
        add.w #1,D1
        swap D1
        move.w #0,D1
        move.w $2552,D2
        swap D2
        move.w $2550,D2
        bra $915B

bclr $3,%2576
move.w SR,-(A7)
bcr $8F00
bcr $8F20
        move.w $2560,D1
        bst $0,D1
        beq $8D58
        cmp.w $2550,D1
        beq $8DB4
        move.w $20,D1
        bcr $90AB
        move.w $2560,D1
        swap D1
        move.w $2562,D1
        move.w D1,D2
        swap D1
        swap D2
        move.w $2550,D2

ESC J, Clear rest of screen, VDI 9
ESC K, Clear rest of line
Cursor line
Compare with maximum cursor line

Maximum cursor line

Maximum cursor column (79)

ESC K, Clear rest of line, VDI ESC 10
Clear flag for line overflow
Save old value
ESC f, Cursor off
ESC j, Save cursor position
Cursor column

Compare with maximum value (79)
In last column, then output space
Space
Output
Cursor column

Cursor line

Maximum cursor column
Abacus Software

Atari ST Internals

$9158, w (A7) +, CCR Restore flag

rl > 0

$8D7C Not set ?

$8D7E Reset

beq 00300002576 bset o

$8F3C ESC k, Restore cursor position

$8EEE Re-enable cursor

*************************************************** VDI ESC 11, Set cursor

008D6E 207900002584 move.l $2584, A0

008D94 3210 move.w (A0), D1

008D96 5341 subq.w $1, D1

008D98 30280000 move.w 2(A0), D0

008D9C 5340 subq.w $1, D0

008D9E 600005CC bra $92AC

***************************************************

008DA2 207900002584 move.l $2584, A0

008DB8 30280000 move.w (A0), D0

008DAB 207900002584 move.l $2584, A0

008DB2 600E bra $8DC2

008DB4 3218 move.w (A0) +, D1

008DB6 4B780080 movem.l D0/ A0, -(A7)

008DBA 6100F2D6 bsr $8A22

008DBE 4CDF0101 movem.l (A7) +, D0/A0

008DEC 51C6FF0 dbra D0, $8DB4

008DC6 4275

******************* VDI ESC 12, Text output

Pointer to INLIN array

Get line

Minus offset

Get column

Minus offset

Set cursor

008DA2 20790000258A move.l $258A, A0

008DA8 30280006 move.w 6(A0), D0

008DAC 207900002584 move.l $2584, A0

008DB2 600E bra $8DC2

008DB4 3218 move.w (A0) +, D1

008DB6 4B780080 movem.l D0/ A0, -(A7)

008DBA 6100F2D6 bsr $8A22

008DBE 4CDF0101 movem.l (A7) +, D0/A0

008DEC 51C6FF0 dbra D0, $8DB4

008DC6 4275

Get pointer to D1

Restore registers

Output character in D1

Registers back

Output next character
**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
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008E2E</td>
<td>4EF90C00FF70</td>
<td><strong>jmp $FF70</strong></td>
</tr>
<tr>
<td>008E34</td>
<td>32390C002562</td>
<td>move.w $2562, D1</td>
</tr>
<tr>
<td>008E3A</td>
<td>660FE650</td>
<td>bne $8C9C</td>
</tr>
<tr>
<td>008E3D</td>
<td>37390C002560</td>
<td>move.w $2560, -(A7)</td>
</tr>
<tr>
<td>008E44</td>
<td>6108</td>
<td>bsr $8E4E</td>
</tr>
<tr>
<td>008E46</td>
<td>301F</td>
<td>move.w (A7)+, D0</td>
</tr>
<tr>
<td>008E48</td>
<td>7200</td>
<td>moveq.l #0, D1</td>
</tr>
<tr>
<td>008E4A</td>
<td>66000460</td>
<td>bra $92AC</td>
</tr>
<tr>
<td>008E4E</td>
<td>610006B0</td>
<td>bsr $8F00</td>
</tr>
<tr>
<td>008E52</td>
<td>32390C002562</td>
<td>move.w $2562, D1</td>
</tr>
<tr>
<td>008E56</td>
<td>6100056E</td>
<td>bsr $93C8</td>
</tr>
<tr>
<td>008E5C</td>
<td>4240</td>
<td>clr.w D0</td>
</tr>
<tr>
<td>008E5E</td>
<td>323900002562</td>
<td>move.w $2562, D1</td>
</tr>
<tr>
<td>008E64</td>
<td>61000446</td>
<td>bsr $92AC</td>
</tr>
<tr>
<td>008E66</td>
<td>60000984</td>
<td>bra $9BE8</td>
</tr>
<tr>
<td>008E6C</td>
<td>61000992</td>
<td>bsr $8F00</td>
</tr>
<tr>
<td>008E70</td>
<td>323900002562</td>
<td>move.w $2562, D1</td>
</tr>
<tr>
<td>008E76</td>
<td>61000508</td>
<td>bsr $9380</td>
</tr>
<tr>
<td>008E7A</td>
<td>60E0</td>
<td>bra $8E5C</td>
</tr>
<tr>
<td>008E7C</td>
<td>C67C000F</td>
<td>end.w #$F,D0</td>
</tr>
<tr>
<td>008E80</td>
<td>33C000002558</td>
<td>move.w D0, $2558</td>
</tr>
<tr>
<td>008E86</td>
<td>4275</td>
<td>rts</td>
</tr>
</tbody>
</table>

**Additional Notes:**
- 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
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
008EE 4A7900002422      tst.w  $2422      Cursor enabled ?
008EF4 679C             beq  $8E92      Yes, rts
008EF6 537900002422      subs.w #1,$2422      See above
008EFC 67C3             beq  $8EE8
008EFF 4E75             rts

008F00 527900002422      addq.w #1,$2422      ESC f, Cursor off
008F06 41F900002576      lea  $2576,(A0)      Flag for cursor off
008F0C 08900002          bclr  #2,(A0)      Clear flag for cursor
008F10 6780             beq  $8E92      Cursor was already off, rts
008F12 06100000          btest  #0,(A0)
008F16 67B6             beq  $8ECE
008F18 06900001          bclx  #1,(A0)
008F1C 66B9             bne  $8ECE
008F1E 4E75             rts

008F20 08F900002576      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
008F30 30B900002562      move.w  $2562,(A0)      Cursor column
008F34 30B900002562      move.w  $2562,(A0)      Cursor line
008F3A 4E75             rts

008F3C 08B9000000002576      bclr  #5,$2576      ESC k, Cursor to saved position
008F44 6700FDAA             beq  $8CFD      Was cursor position saved ?
008F48 41F90000242E      lea  $242E,(A0)      No
008F4E 3618             move.w  (A0)+,(D0)      Address of the temp. storage
008F50 3210             move.w  (A0),D1      Cursor column
008F54 60000358             bra  $92AC      Cursor line
008F58 60000358             bra  $92AC      Setcursor
Abacus Software

restore the cursor line

ESC F, turn cursor off

Cursor line

Maximum cursor column

Cursor in column zero

ESC O, Clear line to cursor
ESC f, Turn cursor off
ESC j, Save cursor position
Cursor column
Zero, then done

Space

Output

Cursor column

ESC k, Restore cursor position
And turn cursor back on
Space

008556 61A8
008558 323900002562
00855E 3401
008560 4841
008562 4241
008564 4842
008566 343900002550
00856C 610001EA
008570 6000FE8A

008574 61A8
008576 61A8
008578 343900002560
00857E 6730
008580 08020000
008584 6610
008586 323C0020
008588 61000122
00858E 343900002560
008594 5542
008596 4842
008598 343900002562
00859E 3202
0086A0 6842
008FA2 4841
008FA4 4241
008FAS 610001B0
008FAA 6190
008FAC 6000FF40
008FB0 323C0020

bsr $8F00
move.w $2562,D1
move.w D1,D2
swap D1
clr.w D1
swap D2
move.w $2550,D2
bsr $9158
bra $8ESC

bsr $8F00
bsr $8F20
move.w $2560,D2
beq $8FB0
btst $0,D2
bne $8F96
move.w #$20,D1
bsr $90AE
move.w $2560,D2
subq.w #$2,D2
swap D2
move.w $2562,D2
move.w D2,D1
swap D2
swap D1
clr.w D1
bsr $9158
bsr $8F3C
bra $8EEE
move.w #$20,D1
008FB4 61000DF8  bsr $90AE
008FB8 60F0  bra $8F01

*******************************************************************************
008FB9 08F9000000002576  bset #3,$2576
008FC2 4E75  rts

*******************************************************************************
008FC4 08B9000000002576  bclr #3,$2576
008FCC 4E75  rts

*******************************************************************************
008FCE 32390000002562  move.w $2562,D1
008FD4 4260  clr.w D0
008FD6 600002D4  bra $92AC

*******************************************************************************
008FDA 303900002562  move.w $2562,D0
008FEC 307900002552  cmp.w $2552,D0
008FEC 6600FCC0  bne $9CA8
008FED 6100FF14  bsr $8FD0
008FEE 4241  clr.w D1
008FF0 610003BE  bsr $9390
008FF4 6000FEB8  bra $8EE2

*******************************************************************************
008FF8 41F900002576  lea $2576,A0
008FFE 08100002  btst #2,(AO)
009002 671E  beq $9022
009004 08100000  btst #0,(AO)
009008 671E  beq $9022
00900A 43F900002565  lea $256D,A1

Output
ESC v, New line at end of line
Set flag
ESC w, No new line at end of line
Clear flag
CR, Cursor in column zero
Column to zero
Set cursor
LF, (VT, FF), Cursor down
Compare with maximum cursor line
Not in lowest line, just cursor down
ESC f, Turn cursor off
Scroll screen down
And turn cursor back on
Flash cursor
Address of the flag word
Cursor on ?
No
Cursor flashing ?
No
Address of the flash counter
subq.b $1, (Al)
bne $9022
move.b $2561, (Al)
bchg $1, (AO)
bra $8ECE
Decrement counter
Not yet run down?
Reload flash counter
Invert cursor phase
Turn cursor off

move.w 4(A7),DO
bmi $9022
cmp.w #5,DO
bgt $9022
asl.w $1,DO
lea $904A, AO
add.w $903E(PC,DO.w),AO
jmp (AO)

Cursor configuration, XBIOS No. 21
Get number from stack
Negative, ignore
Greater than 5?
Yes, ignore
Base address of the table
Plus relative address
Execute function
Address of the routines

0
ESC f, Turn cursor off
1
ESC e, Turn cursor on
009052 6100FEAC  bsr  $8F00
009056 08ED00002576  bset  #0,$2576(A5)
00905C 6000FE90  bra  $8EEB

009060 6100FE9E  bsr  $8F00
009064 08AD00002576  bclr  #0,$2576(A5)
00906A 6000FE82  bra  $8EEB

00906E 1B6F00072564  move.b  7(A7),$2564(A5)
009074 4E75  rts

009076 7000  moveq.l #0,DO
009078 102D2564  move.b  $2564(A5),DO
00907C 4E75  rts

00907E 36390000256C  move.w  $256C,D3
009084 B243  cmp.w  D3,D1
009088 6522  bcs  $90AA
009088 B2790000256A  cmp.w  $256A,D1
00908E 621A  bhi  $90AA
009090 257900002572  move.l  $2572,A0
009096 D241  add.w  D1,D1
009098 32301000  move.w  (A0,D1.w),D1
00909C 5549  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
3  ESC f, Turn cursor off
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
Character not present in font

ascout, ignore control codes

Yes

Screen address of the character
Background color
In upper word
Character color in lower word
Reverse turned on?
No
Exchange character and background colors

Save status
Calculate new position
Screen address
Cursor column
Cursor line
Calculate relative screen address

Number of bytes per character line
Address of the screen RAM

Compare with maximum cursor line
Plus number of bytes per character line
00911A 5241  addq.w #1, D1
00911C 600E  breq $912C
00911E 48E7C040 movem.l DO-D1/AI,-(A7)  Save registers
009122 7200  moveq.l #0, D1
009124 6100025A  bar $9380
009128 4CFD0203 movem.l (A7)+,DO-D1/A1  Restore registers
00912C 23C90000255A move.l A1,$255A  Screen address
009132 33C000002560 move.w DO,$2560  Cursor column
009138 33C100002562 move.w D1,$2562  Cursor line
00913E 44DF  move.w (A7)+,CCR  Restore status
009140 6716  beq $9156
009142 610001DA  bar $931E
009146 08F900002576 bset #1,$2576
00914E 08F900002576 bset #2,$2576
009156 4E75  rts
009158 9481  sub.l D1,D2
00915A 3001  move,w D1,D0
00915C 4841  swap D1
00915E 6100098  bar $91F8  Calculate cursor position
009162 E242  add.w #1,D2
009164 36390000257C move.w $257C,D3  Number of screen planes (1,2 or 4)
00916A DC330004  cmp.w #4,D3
00916E 6602  bne $9172
009170 5363  subq.w #1,D3
009172 3202  move,w D2,D1
009174 5241  addq.w #1,D1
009176 E761  asl.w D3,D1
009179 34790000257E move.w $257E,A2  Bytes per screen line
00917E 94C1  sub.w D1,A2
009180 3202  move,w D2,D1
009182 4642  swap D2
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 51CDFFFC dbra  D5,§91EA
0091EC D3CA  add.l  A2,A1
0091F2 51CAFFF4 dbra  D2,§91EA
0091F6 4E75  rts

*****************************************************************************
0091F8 363900002550  move.w  $2550,D3
0091FA B640  cmp.w  D0,D3
009200 6A02  bpl  $9204
009202 3C03  move.w  D3,D0
009204 363900002552  move.w  $2552,D3
009206 B641  cmp.w  D1,D3
009208 6A02  bpl  $9210
00920A 3203  move.w  D3,D1
009210 36390000257C  move.w  $257C,D3
009212 3A00  move.w  D0,D5
009214 0850000  bcir  #0,D5
009216 C6C5  mulu.w  D5,D3
009218 0800000  btest  #0,D0
00921A 6702  beq  $9226
00921C 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
move.w $2554,D5
mulw.w D1,D5
move.l $44E,A1
add.l D5,A1
add.l D3,A1
add.w $255E,A1
rta
move.w $2562,A2
move.w $257E,A3
move.w $254E,D4
subq.w #1,D4
move.w $257C,D6
subq.w #1,D6
move.w D4,D5
move.l A0,A4
move.l A1,A5
asr.l #1,D7
btst #15,D7
beq $9270
bcs $9296
moveq.l #1,D3
bra $9274
bcx $9284
moveq.l #0,D3
move.b D3,(A5)
add.w A3,A6
dbr a D5,$9274
addq.w #2,A1
dbra D6,$925C
rts

Number of bytes per character line
Times line value
Base address of the screen RAM
Plus offset for line
Plus offset for column
Plus number of bytes per raster line

Formwidth
Bytes per screen line
Height of a character
Number of screen planes
009284 1A94  move.b (A4), (A5)
009286 DACB  add.w A3, A5
009288 D8CA  add.w A2, A4
00928A 51CDFF8  cbra D5, 9294
00928E 5449  addq.w #2, A1
009290 51CEFFCA  cbra D6, #925C
009294 4E75  rts

009296 1614  move.b (A4), D3
009298 4603  not.b D3
00929A 1A83  move.b D3, (A5)
00929C DACB  add.w A3, A5
00929E D8CA  add.w A2, A4
0092A0 51CDFF4  cbra D5, #9296
0092A4 5449  addq.w #2, A1
0092A6 51CEFFB4  cbra 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
0092CC 33C100002562  move.w DL, $2562
0092D4 41F900002576  lea $2576, A0
0092DA 08100002  btst #2, (A0)
0092DB 6732  beq $9312
0092ED 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
bcir $2,(A0)
bset $1,(A0)
beq $930B
move.l $255A,A1
bar $931E
bar $91F8
move.l A1,$255A
bset $2,$2576
rts

dset $2,(A0)
bar $91F8
move.l A1,$255A
rts

*******************************************************************************
move.w $257E,A2
move.w $254E,D4
subq.w #1,D4
move.w $257C,D6
subq.w #1,D6
move.w D4,D8
move.l A1,A8
not.b (A4)
add.w A2,A6
dbra D5,$9338
addq.w #2,A1
dbra D6,$9334
rts

*******************************************************************************
Screen address of old cursor
Invert character at cursor position
Calculate new cursor address
Screen address of the cursor
Invert character at cursor position
Flag for character is inverted

Calculate new cursor position
Screen address of the cursor position

Invert character at cursor position
Bytes per screen line
Height of a screen line
As dbra counter
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
Abacus Software

O'HER

009348 B07900002550  cmp.w $2550, D0
00934E 6612  bne $9362
009350 0839000030002576  btst #3, $2576
009358 6604  bne $9358
00935A 4263  clr.w D3
00935C 4E75  rts
00935E 7601  moveq.l #1, D3
009360 4275  rts

009362 5240  addq.w #1, D0
009364 08000000  btst #0, D0
009368 6706  beq $9370
00936A 5249  addq.w #1, A1
00936C 4263  clr.w D3
00936E 4E75  rts
009370 36390000257C  move.w $257C, D3
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 267900000446  move.l $446E, A3
009386 363900002554  move.w $2554, D3
00938C C6C1  mulw.w D1, D3
00938E #7F33000  lea 0(A3, D3.w), A3
009392 4441  neg.w D1
009394 D27900002552  add.w $2552, D1
00939A 363900002554  move.w $2554, D3

Maximum cursor column
Flag for line overflow set?
Yes

Number of screen levels

Scroll screen down at line D1
Address of the screen RAM
Number of bytes per character line
Multiply by number of lines
Yields address of the line
Current line
Maximum cursor line
Number of bytes per character line
lea 0(A3,D3.w),A2
mulu.w D1,D3
asl.w $2,D3
bra $93AC
move.l (A2)+,(A3)+
dbra D3,$93AA
move.w $2552,D1
move.w D1,D2
swap D1
swap D2
clr.w D1
move.w $2550,D2
bra $9158

Yields address of line 1
Number of bytes to move
Number of long words
Copy screen lines
Maximum cursor line

Maximum cursor column

Scroll screen at line D1 up
Address of the screen RAM
Maximum line
Mult by number of bytes per character line
Yields address of last character line
Number of bytes per line
Yields address of line 1
Current line

Add maximum cursor line

Divide by 4 for long word transfer
Copy screen lines
<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>009400</td>
<td>207900002584</td>
<td>move.l $2584, A0</td>
</tr>
<tr>
<td>009406</td>
<td>2050</td>
<td>move.l (A0), A0</td>
</tr>
<tr>
<td>009406</td>
<td>30280052</td>
<td>move.w 62 (A0), D0</td>
</tr>
<tr>
<td>009412</td>
<td>33C0000254E</td>
<td>move.w D0, $254E</td>
</tr>
<tr>
<td>009412</td>
<td>3239000257E</td>
<td>move.w $257E, D1</td>
</tr>
<tr>
<td>009419</td>
<td>C20C</td>
<td>mulu.w D0, D1</td>
</tr>
<tr>
<td>00941A</td>
<td>33C100002554</td>
<td>move.w D1, $2554</td>
</tr>
<tr>
<td>009420</td>
<td>7200</td>
<td>moveq.l #0, D1</td>
</tr>
<tr>
<td>009422</td>
<td>32390002578</td>
<td>move.w $2578, D1</td>
</tr>
<tr>
<td>009428</td>
<td>820C</td>
<td>divu.w D0, D1</td>
</tr>
<tr>
<td>00942A</td>
<td>5341</td>
<td>subq.w #1, D1</td>
</tr>
<tr>
<td>009432</td>
<td>7200</td>
<td>move.w D1, $2552</td>
</tr>
<tr>
<td>009432</td>
<td>32390002570</td>
<td>moveq.l #0, D1</td>
</tr>
<tr>
<td>00943A</td>
<td>82E80034</td>
<td>move.w $2570, D1</td>
</tr>
<tr>
<td>00943B</td>
<td>5341</td>
<td>divu.w 52 (A0), D1</td>
</tr>
<tr>
<td>009440</td>
<td>33C100002550</td>
<td>subq.w #1, D1</td>
</tr>
<tr>
<td>009446</td>
<td>33EB00500000256E</td>
<td>move.w D1, $2550</td>
</tr>
<tr>
<td>00944E</td>
<td>33E800240000256C</td>
<td>move.w 90 (A0), $256E</td>
</tr>
<tr>
<td>009456</td>
<td>33EB00260000256A</td>
<td>move.w 36 (A0), $256C</td>
</tr>
<tr>
<td>00945E</td>
<td>2358003C00002566</td>
<td>move.w 38 (A0), $256A</td>
</tr>
<tr>
<td>009466</td>
<td>23E8004800002572</td>
<td>move.l 76 (A0), $2566</td>
</tr>
<tr>
<td>00946E</td>
<td>A75</td>
<td>move.l 78 (A0), $2572</td>
</tr>
</tbody>
</table>

**Notes:**
- **VDEV ESC 102, initialize font parameters**
- Pointer to INTIN array
- Address of the font header
- Font height, height of a character
- Number of raster lines on the screen
- Divide by font height
- Minutes
- Yields maximum cursor line
- Screen width in bits
- Divide by maximum character width
- Minus 1
- Yield maximum cursor column
- Form width, width of the font
- Smallest ASCII code in font
- Largest ASCII code in font
- Pointer to font data
- Pointer to offset table
**** Initialize screen output
s

set up, screen resolution
I

Isolate bits 0 and 1
3

No

Replace with 2 (high resolution)
S

Save resolution
R

Set parameters for screen resolution
R

Restore resolution
R

Address of the 8x8 system font
H

High resolution?
N

No

Address of the 8x16 system font
I

Initialize font data
C

Character color to black
B

Background color white
C

Cursor column zero
C

Cursor line zero
C

Address of the video RAM
A

As screen address of the cursor
S

Set cursor flag
C

Cursor flash counter to 30
C

Cursor flash rate to 30
C

Set flag for cursor on
C

8000 long words
C

Clear screen
C

conout vector to standard
C

**********
Q0F6C4 31390000004C
Q0F6CA CF7C0003
Q0F6CE BF7C0003
Q0F6D2 6604
Q0F6D4 303C0002
Q0F6D8 3F00
Q0F6DA 6100007E
Q0F6E2 301F
Q0F6F0 41F900017EAA
Q0F6F6 070C0002
Q0F6F8 6606
Q0F6F6 33FCFFFF0002558
Q0F6FE 7000
Q0F700 33C000002556
Q0F706 33C000002560
Q0F70C 33C000002562
Q0F712 33C00000255E
Q0F718 20790000004E
Q0F722 23C80000235A
Q0F724 13FC000100002376
Q0F72C 13FC001E00002365
Q0F734 13FC001E00002364
Q0F73C 33FC000100002422
Q0F744 323C1F3F
Q0F748 200C
Q0F74A 51CFFFC
Q0F74B 23FC00008AE000048
Q0F758 4E75

move.b $44C,D0
and.w $3,D0
cmp.w $3,D0
bne $F6D8
move.w $2,D0
move.w D0,-(A7)
lea $F75A
move.w (A7)+,D0
lea $17EAA,A0
cmp.w $2,D0
bne $F6F2
lea $18906,A0
bar $9408
move.w #$FF8F,$2558
moveq.l #0,D0
move.w D0,$2556
move.w D0,$2560
move.w D0,$2562
move.w D0,$255E
move.l $44E,A0
move.l A0,$255A
move.b $1,$2576
move.b #$1E,$2565
move.b #$1E,$2564
move.w $1,$2422
move.w #$1F3F,D1
move.l D0,(A0)+
dbra D1,$F748
move.l #$8ABE,$4A8
rts
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
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.
### 4.2 Alphabetical listing of GEMDOS functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Opcode (hex)</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cauxin</td>
<td>03</td>
<td>106</td>
</tr>
<tr>
<td>Cauxis</td>
<td>12</td>
<td>111</td>
</tr>
<tr>
<td>Cauxos</td>
<td>13</td>
<td>111</td>
</tr>
<tr>
<td>Cauxout</td>
<td>04</td>
<td>106</td>
</tr>
<tr>
<td>Cconin</td>
<td>01</td>
<td>105</td>
</tr>
<tr>
<td>Cconis</td>
<td>0B</td>
<td>110</td>
</tr>
<tr>
<td>Cconos</td>
<td>10</td>
<td>111</td>
</tr>
<tr>
<td>Cconout</td>
<td>02</td>
<td>106</td>
</tr>
<tr>
<td>Ccours</td>
<td>0A</td>
<td>109</td>
</tr>
<tr>
<td>Cconws</td>
<td>09</td>
<td>109</td>
</tr>
<tr>
<td>Cnecin</td>
<td>08</td>
<td>108</td>
</tr>
<tr>
<td>Cprnos</td>
<td>11</td>
<td>111</td>
</tr>
<tr>
<td>Cprnout</td>
<td>05</td>
<td>107</td>
</tr>
<tr>
<td>Crawcin</td>
<td>07</td>
<td>108</td>
</tr>
<tr>
<td>Crawio</td>
<td>06</td>
<td>107</td>
</tr>
<tr>
<td>Dcreate</td>
<td>39</td>
<td>117</td>
</tr>
<tr>
<td>Ddelete</td>
<td>3A</td>
<td>118</td>
</tr>
<tr>
<td>Dfree</td>
<td>36</td>
<td>116</td>
</tr>
<tr>
<td>Dgetdrv</td>
<td>19</td>
<td>111</td>
</tr>
<tr>
<td>Dgetpath</td>
<td>47</td>
<td>128</td>
</tr>
<tr>
<td>Dsetdrv</td>
<td>0E</td>
<td>110</td>
</tr>
<tr>
<td>Dsetpath</td>
<td>3B</td>
<td>119</td>
</tr>
<tr>
<td>Fattrib</td>
<td>43</td>
<td>126</td>
</tr>
<tr>
<td>Fclose</td>
<td>3E</td>
<td>122</td>
</tr>
<tr>
<td>Fcreate</td>
<td>3C</td>
<td>120</td>
</tr>
<tr>
<td>Fdatetime</td>
<td>57</td>
<td>137</td>
</tr>
<tr>
<td>Fdelete</td>
<td>41</td>
<td>124</td>
</tr>
<tr>
<td>Fdup</td>
<td>45</td>
<td>128</td>
</tr>
<tr>
<td>Fforce</td>
<td>46</td>
<td>128</td>
</tr>
<tr>
<td>Fgetdta</td>
<td>2F</td>
<td>115</td>
</tr>
<tr>
<td>Fopen</td>
<td>3D</td>
<td>121</td>
</tr>
<tr>
<td>Fread</td>
<td>3F</td>
<td>123</td>
</tr>
<tr>
<td>Frename</td>
<td>56</td>
<td>136</td>
</tr>
<tr>
<td>Fseek</td>
<td>42</td>
<td>125</td>
</tr>
<tr>
<td>Fsetdta</td>
<td>1A</td>
<td>112</td>
</tr>
<tr>
<td>Fsfirst</td>
<td>4E</td>
<td>134</td>
</tr>
<tr>
<td>Fsnext</td>
<td>4F</td>
<td>136</td>
</tr>
<tr>
<td>Fwrite</td>
<td>40</td>
<td>124</td>
</tr>
<tr>
<td>Abacus Software</td>
<td>Atari ST Internals</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>Malloc</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Mfree</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>Mshrink</td>
<td>4A</td>
<td></td>
</tr>
<tr>
<td>Pexec</td>
<td>4B</td>
<td></td>
</tr>
<tr>
<td>Pterm</td>
<td>4C</td>
<td></td>
</tr>
<tr>
<td>Pterm0</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>Ptermres</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Super</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Sversion</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Tgetdate</td>
<td>2A</td>
<td></td>
</tr>
<tr>
<td>Tgettime</td>
<td>2C</td>
<td></td>
</tr>
<tr>
<td>Tsetdate</td>
<td>2B</td>
<td></td>
</tr>
<tr>
<td>Tsettime</td>
<td>2D</td>
<td></td>
</tr>
</tbody>
</table>
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
DMAs—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
IACK, 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 Musical 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 Instrument 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

454
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.
FESSIONAL
DUCTIVITY

New ST software from a name you can count on...

PCBoard Designer
Create printed circuit board layouts
Features: Auto-routing, component list, pinout list, net list

TextPro
Word processor for the ST

FilePro
The electronic filing system for the ST

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 BUSH-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

TextPro
Word processor 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; automatic printer installation; automatic index and table of contents; up to 160 charlines; 30 definable function keys; metallic output; much more. $49.95

FilePro
A simple-to-use and versatile database manager. Features help screens; lightning-fast operation; tolerable display using multiple font; user-definable add 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

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. TOSLINES commands. Floating point and complex arithmetic. Available Sept. 96. $49.95

PaintPro
A GEM™ among ST drawing programs. Very friendly, but very powerful. A must for everyone's artistic or graphics needs. Up to three windows. Free-form sketching, lines, circles, ellipses, boxes, text, fill, copy, move, zoom, spray, paint, undo, help. $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. 96. $39.95

Other software and books coming soon. Call or write for your free catalog. Dealer inquiries welcome—over 1400 dealers nationwide.
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
Word processor for the ST

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
With Text Designer you are now able to combine text with graphics on your screen. Add artwork to your page layouts. Liven up your pages with pictures to add that personal flair.

**Text Designer**

Combine graphics with your text

**Text Designer** is an ideal package for page layout on the ST allowing you to interlace graphics with text. **Text Designer** 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. **Text Designer** 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
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, and the various interfaces. LOGO, 200pp $16.95

ST INTERNALS
Essential guide to the inside information of the ST. Detailed descriptions of sound and graphics chips, internal hardware, and GEM. Documented BIOS listing. 450pp $19.95

MACHINE LANGUAGE
Program in the fastest language for your ATARI ST. Learn 68000 assembly language, its numbering system, use of registers, structure of important details of Instruction set, and use of internal system routines. Geared for the ST. 260pp $19.95

ST TRIKS & TIPS
Fantastic collection of programs and info for the ST. Complete programs include: super-fast RAM dump; time-saving printer spooler; color print file; plotter output hardcopy. Money saving tricks and tips. 260pp $19.95

ST LOGO GUIDE
Take control of your ST by learning ST LOGO—*he easy-to-use, powerful language. Topics include: file handling, recursion, fractals, 2D and 3D function plots, data structures. Helpful guide for STLOGO users. $19.95

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 American Express 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.
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 knowledgable concerning the subjects presented in this book.

ISBN 0-916439-46-1

A Data Becker book published by Abacus Software