ATARI®400/800™

ATARI HOME COMPUTER SYSTEM

TECHNICAL REFERENCE NOTES

includes:

Operating System User's Manual
Operating System Source Listing
and
Hardware Manual

ATARI®

A Warner Communications Company

C016555 Rev. A

ATARI® 400/800™

ATARI® HOME COMPUTER SYSTEM

OPERATING SYSTEM
USER'S MANUAL

ATARI® A Warner Communications Company

Every effort has been made to ensure that this manual accurately
documents this product of the ATARI Home Computer Division.
However, due to the ongoing improvement and update of the computer
software and hardware, ATARI, INC. cannot guarantee the accuracy
of printed material after the date of publication and disclaims
liability for changes, errors, or omissions.

APPENDICES

TABLE OF ILLUSTRATIONS

PREFACE

This manual describes the resident Operating System (OS) for the
ATARI@ Home Computer, for readers who are familiar with the
internal behavior of the system. It discusses:

    o   System functions and utilization techniques

    o   Subsystem relationships and organization

    o   Characteristics of the ATARI peripheral devices that can
        be attached to the ATARI400[TM] and ATARI 800[TM] Home
        Computer

    o   Advanced techniques for going beyond the basic OS
        capabilities

    o   The general features of the computer system hardware used
        by the OS.

It would be helpful to have a familiarity with programming concepts
and terminology, assembly language programming in general, the
Synertek 6502 in particular, and digital hardware concepts and
terminology. you will be provided with the information you need to
use the OS resources, without resorting to trial-and-error techniques
or the OS listing. Supporting information for tasks that involve OS
listing references is also provided.

This manual does not present a comprehensive description of the
hardware used to provide OS capabilites. The programmer who needs to
go beyond the capabilities described should consult the ATARI Home
Computer Hardware Manual.

1 INTRODUCTION

GENERAL DESCRIPTION OF THE ATARI HOME COMPUTER SYSTEM

Operating systems in the ATARI@ 400[TM] and ATARI 800[TM] Home
Computer are identical. The primary differences between the two are:

o   Physical packaging

o   The ATARI 400 Computer console has one cartridge slot, the
     ATARI 800 Computer console has two cartridge slots

o   The ATARI 400 Home Computer contains 16K RAM and cannot be
     expanded. The ATARI 800 Home Computer can be expanded to a
     maximum of 48K RAM.

o   The ATARI 800 Computer has a monitor jack; the ATARI 400
     Computer does not.

The Hardware Circuitry

o Produces both character and point graphics for black and
  white (B/W) or color television.

o Produces four independent audio channels (frequency
  controlled) which use the television sound system.

o Provides one bi-level audio output in the base unit.

o Interfaces with up to four Joysticks and eight Paddle
  Controllers.

o Interfaces with a serial I/O bus for expansion.

o Contains a built-in keyboard

Figure 1-1 presents a simplified block diagram of the hardware.
See the hardware manual for supporting documentation.

```
    +----------+
    :   6502   :
    :processor :
    +----+-----+     +----------+
         :           :    OS    :
    +----+----+      :   ROM    :
    :           +----+ :D800-FFFF:
    :           :    +----------+     +----------+
    :           :         :           +----------+ :
    :           :         :      +----------+ : :
    :           +------------------+   RAM    : : :
    :           :                  :          : :-+
    :           :                  :0000-xxxx :-+
    :           :     +----------+  +----------+
    :           :     :Cartridge:
    :        +--+-----+slots A&B:
    :        :        :8000-BFFF:
 processor  :        +----------+
 external   :
 bus        :        +----------+ joysticks    +----------+
            :        :    PIA   +----------+----------+control- :
    +-------+--+     +----------+----------:-----+ :ler ports:
    :.........:D300-D31F+---------:-----+ :  :          :
    :  IRQ    :     +----------+         :  :  : +----------+
    :        :                           :  :  :
    :        :     +----------+ pots     :  :  : +----------+
    :        :     :   POKEY  +----------+  :  : :keyboard :
    +-------+--+   +----------:-----:-+--+ :data keys:
    :.........:D200-D21F+--------:---+ :  : + BREAK  :
    :  IRQ    :     +----+----+ audio  :  :  : +----------+
    :        :     +----------:-+  :  :  : on/off
    :        :     +----+----+ trigs :  :  : +---+cassette :
    :        :     :   CTIA   +---------+  : :  :interface:
    +-------+--+   +-----+----+ :  +-:-:----+        :
    :        :     :D000-D01F+-----:--:-+ : :  +----------+
    :        :     +----+----+     :  : : : +----+ serial  :
    :        :     : video   :     :  : :  :    : I/O     :
    :  DMA   +----+----+ litpen :  :  +------+    : bus     :
    :.........:   ANTIC  +--------+  :          +----------+
    +-------+--+     :        :      :
    :.........:D400-D41F:      :      :          +----------+
    :  NMI    +----------+     :      :  from    : console :
    :                          :      :  CTIA -+ speaker :
    :        composite +----+  :      :          :        :
    :        audio/video :    :       :          +----------+
    :                    :    :       :
    :               +----+----+      :
    :               :        :       :          +----------+
    :               :   TV   : +--------+ : START,  :
    :               :        : +---------+ SELECT, :
    :               :        :       : OPTION  :
    :               +----------+      +----------+
```

Figure 1-1. ATARI Home Computer Block Diagram

CONVENTIONS USED IN THIS MANUAL

This manual uses the following special notations:

Hexadecimal Numbers

All two-digit numbers preceded by a dollar sign ($) designate
hexadecimal numbers. All other numbers (except memory addresses)
are in decimal form unless otherwise specified in the supporting
text.

Memory Addresses

All references to computer memory and mapped I/O locations are in
hexadecimal notation. Memory addresses may or may not be contained
in square brackets. (Example: [D20F] and D20F are the same
address.)

Kilobytes of Memory

Memory sizes are frequently expressed in units of kilobytes, such
as 32K, where a kilobyte is 1024 bytes of memory.

PASCAL As an Algorithm-Specification Language

The PASCAL language (procedure block only) is used as the
specification language in the few places where an algorithm is
specified in detail. PASCAL syntax is similar to any number of
other block-structured languages, and you should have no
difficulty following the code presented.

Memory Layouts

Diagrams similar to Figure 1-2 are used whenever pictures of bytes
or tables are presented:

```
         7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+
        |               |  --- This is a single byte.
        +-+-+-+-+-+-+-+-+
        |               |
        +               +  --- This is a word (2 bytes).
        |               |
        +-+-+-+-+-+-+-+-+
        |               |
        =               =  --- This is a block of memory
        |               |      of unspecified length.
        +-+-+-+-+-+-+-+-+
```
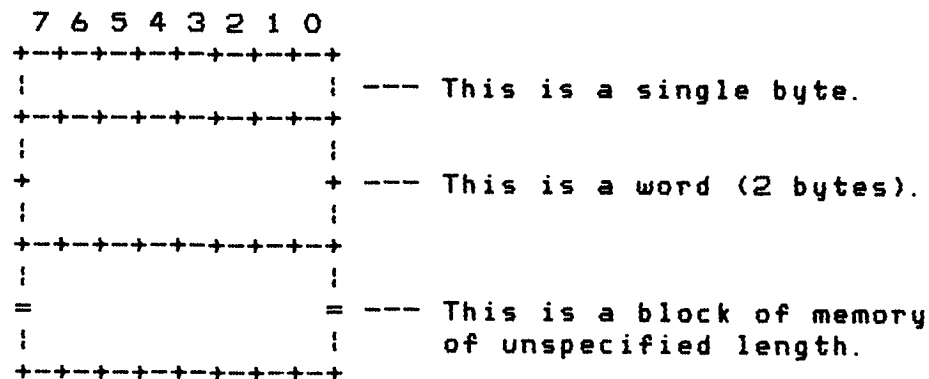
Figure 1-2. Memory Layout Chart

Bit 7 is the most significant bit (MSB) of the byte, and Bit 0
is the least significant bit (LSB).

In tables and figures, memory addresses always increase toward the
bottom of the figure.

Backus-Naur Form

A modified version of Backus-Naur Form (BNF) is used to express some
syntactic forms, where the following metalinguistic symbols are used:

    ::=   is the substitution (assignment) operator.

    < >   a metasyntactic variable.

    !   separates alternative substitutions.

    [ ]   an optional construct.

    Anything else is a syntactic literal constant, which stands for
    itself.

For Example:

    <device specification> ::= <device name>[<device number>]:

    <device name> ::= C!D!E!K!P!R!S

    <device number> ::= 1!2!3!4!5!6!7!8

A "device specification" consists of a mandatory "device name,"
followed by an optional "device number," followed by the mandatory
colon character. The device name in turn must be one of the
characters shown as alternatives. The device number (if it is present)
must be a digit 1 through 8.

OS Equate Filenames

Operating System ROM (Read Only Memory) and RAM (Random Access
Memory) vector names, RAM database variable names and hardware
register names are all referred to by the names assigned in the OS
program equate list. When one of these names is used, the memory
address is usually provided, such as BOOTAD [0242].

2 OPERATING SYSTEM FUNCTIONAL ORGANIZATION

This section describes the various subsystems of the resident OS in general terms.

Input/Output Subsystem

The Input/Output (I/O) subsystem provides a high-level interface between the programs and the hardware. Most functions are device-independent, such as the reading and writing of character data; yet provisions have been made for device-dependent functions as well. All peripheral devices capable of dealing with character data have individual symbolic names (such as K, D, P, etc). and can be accessed using a Central I/O (CIO) routine.

A RAM data base provides access to controllers (joysticks and paddle controllers), which do not deal with character data. This RAM data base is periodically updated to show the states of these devices.

INTERRUPT PROCESSING

The interrupt system handles all hardware interrupts in a common and consistent manner. By default, all interrupts are fielded by the OS. At your discretion, individual interrupts (or groups of interrupts) can be fielded by the application program.

INITIALIZATION

The system provides two levels of initialization: power up and system reset. The OS performs power-up initialization each time the system power is switched to ON, and system reset initialization is performed each time the [SYSTEM.RESET] key is pressed.

Power-Up

The OS examines and notes the configuration of the unit whenever the system power is switched to ON. The system performs the following tasks at power up:

OPERATING SYSTEM C016555 -- Section 2

o   Determines the highest RAM address.

o   Clears all of RAM to zeros.

o   Establishes all RAM interrupt vectors.

o   Formats the device table.

o   Initializes the cartridge(s).

o   Sets up the screen for 24 x 40 text mode.

o   Boots the cassette if directed.

o   Checks cartridge slot(s) for diskette-boot instructions.

o   Boots the diskette if directed to do so and a disk drive unit
    is attached.

o   Transfers control to the cartridge, diskette-booted program,
    cassette-booted program, or blackboard program.

[SYSTEM.RESET]

Pressing the [SYSTEM.RESET] key causes the OS to perform these
following tasks:

o   Clears the OS portion of RAM.

o   Rechecks top of RAM.

o   Reestablishes all RAM interrupt vectors.

o   Formats the device table.

o   Initializes the cartridge(s).

o   Sets up the screen for 24 x 40 text mode.

o   Transfers control to the cartridge, a diskette-booted program,
    a cassette-booted program, or the blackboard program.

Note that [SYSTEM.RESET] does not perform all the power-up
tasks listed in the power-up section.

FLOATING POINT ARITHMETIC PACKAGE

The OS ROM contains a Floating Point (FP) package that is available
to nonresident programs such as ATARI BASIC.
The package is not used by the other parts of the OS itself. The
floating point numbers are stored as 10 BCD digits of mantissa, plus a
1-byte exponent. The package contains these routines:

   o   ASCII-to-FP and FP-to-ASCII conversion.

   o   Integer-to-FP and FP-to-integer conversion.

   o   FP add, subtract, multiply and divide.

   o   FP log, exp, and polynomial evaluation.

   o   FP number clear, load, store, and move.

3 CONFIGURATIONS

The ATARI 400 and ATARI 800 Home Computers support a
wide variety of configurations, each with a unique operating
environment:

    o Cartridge(s) may or may not be inserted

    o Memory can be optionally added to the ATARI 800 Computer
       console in 16K increments

    o Many different peripheral devices can be attached to the
       serial I/O bus.

The OS accounts for all of these variables without requiring a
change in the resident OS itself (see Section 2). The machine
configuration is checked when power is first turned on and then
is not checked again, unless system reset is used. A general
discussion of some of the valid configurations follows.

PROGRAM ENVIRONMENTS

The OS allows one of four program types to be in control at any
point in time:

    o The OS blackboard (ATARI Memo Pad) program

    o A cartridge-resident program

    o A diskette-booted program

    o A cassette-booted program

Control choice is based upon information in the cartridge(s), upon
whether or not a disk drive is attached, and upon operator keyboard
inputs. The exact algorithms are discussed in detail in Section 7.

Blackboard Mode

In blackboard mode, the screen is established as a 24 x 40 text
screen. Anything entered from the keyboard goes to the screen
without being examined, although all of the screen editing
functions are supported. Blackboard mode is the lowest priority
environment. You go there only by command from a higher

priority environment, or by default, if there is no other
reasonable environment for the OS to enter. For example, typing BYE
in BASIC causes the OS to enter the blackboard mode. The blackboard
mode can be exited by pressing the [SYSTEM.RESET] key if it was
entered from a higher environment.

Cartridge

An inserted cartridge normally provides the main control after
initialization is complete (for example: ATARI BASIC, SUPER
BREAKOUT[TM], BASKETBALL, COMPUTER CHESS, and others. All these
cartridge programs interface directly with you in some way). Although
a cartridge can provide a supporting function for some other program
environment, this has not yet been done. Some cartridges (particularly
keyboard-oriented ones) can change environments by entering special
commands (such as "BYE") to go to blackboard mode or "DOS" to enter
the disk utility. Other cartridges cannot change environments. Note
that a hardware interlock prevents the removal or insertion of a
cartridge with the power on; this feature causes the entire system to
reinitialize with every cartridge change.

Diskette Boot

The diskette may or may not be booted when the system powers up
with diskette-bootable software. This paragraph assumes that a
diskette boot did occur. See Section 7 for boot condition
explanations.

The diskette-booted software can take control as the Disk Utility
Program (DUP) does under certain conditions, or can provide a
supporting function as the File Management System (FMS) does. This
environment is so flexible that it is difficult to generalize on its
capabilities and restrictions. The only machine requirement (other
than the disk drive) is that sufficient RAM be installed to support
the program being booted.

Cassette-Boot

The cassette-boot environment is similar to the diskette-boot
environment, although the cassette is limited as an I/O device. It
is slow and can access only one file at a time in sequence. Note
that the cassette-boot facility has no relation to the use of
cassettes to store high-level language programs (e.g., programs
written in ATARI BASIC), nor to the use of cassettes to store data.

RAM EXPANSION

Although you can expand RAM noncontiguously in the
ATARI 800 Home Computer, the OS will only recognize RAM
that is contiguous starting from location 0. Installation
directions are provided with the purchased RAM modules. RAM can be
added until it totals 48K. After 32K, additional RAM overlays first
the right-cartridge addresses (32K to 40K) and then the
left-cartridge addresses (40K to 48K). Note that in cases of
conflict, the inserted cartridge has higher priority and disables
the conflicting RAM in 8K increments. See Section 4 for a detailed
discussion of system memory.

As a result of power-up, the OS will generate two pointers that
define the lowest available RAM location and the highest available
RAM location. The OS and diskette or cassette-booted software will
determine the location of the lowest available RAM, while the
number of RAM modules and the current screen mode will determine
the highest available RAM.

PERIPHERAL DEVICES

Peripheral devices of several types can be added to the system
using standard cables to either the serial bus or the connectors at
the front of the computer console. The most common types deal with
either transmission of bytes of data (usually serial bus) or
transmission of sense information (usually game controllers).

Game Controllers

The OS periodically senses (50 or 60 times per second) the standard
game controllers (Paddles and Joysticks) and the values read are
stored in RAM. You can plug in, remove, and rearrange these
controllers at will without affecting system operation, because the
system will always try to read all of these controllers.
The Driving Controllers are read, but not decoded, by the OS. Special
instructions are required to read the keyboard controller (see
Section 11).

Program Recorder

The ATARI 410[TM] Program Recorder is a special peripheral. It uses
the serial bus to send and receive data, but does not conform to
the protocol of the other peripherals that use the serial bus. The
Program Recorder must also be the last device on the serial bus,
because it does not have a serial bus extender connector as the
other peripherals do. There can never be more than one Program
Recorder connected to any system for the same reason. The system
cannot sense the presence or absence of the Program Recorder, so it
can be connected and disconnected at will.

Serial Bus Devices

A serial bus device conforms to the serial I/O bus protocol as
defined in Section 9, but this does not include the Program
Recorder. Each serial bus device has two identical connectors: a
serial bus input, and a serial bus extender. Either connector can
be used for either purpose. Peripherals can be "daisychained" by
cabling them together in a sequential fashion. There are usually no
restrictions on the cabling order because each device has a unique
identifier. Where restrictions exist, they will be mentioned in
Section 5.

4   SYSTEM MEMORY UTILIZATION

Memory in the system is decoded in the full 64K range of the 6502
microcomputer and there are no provisions for additional mapping to
extend memory. Memory is divided into four basic regions (with some
overlap possible): RAM, cartridge area, I/O region and the resident
OS ROM. The regions and their address boundaries are listed below
(all addresses are in hexadecimal):

    0000-1FFF = RAM (minimum required for operation)
    2000-7FFF = RAM expansion area
    8000-9FFF = Cartridge B, Cartridge A (half of 16K size) or RAM
    A000-BFFF = Cartridge A or RAM
    C000-CFFF = Unused
    D000-D7FF = Hardware I/O decodes
    D800-DFFF = Floating Point Package (OS)
    E000=FFFF = Resident Operating System ROM

Figure 4-1   6502 System Memory Map

This section will break these regions into even smaller functional
divisions and provide detailed explanations of their usage.

RAM REGION

The OS and the control program share the RAM region. The RAM region
can be further subdivided into the following sub regions for
discussion purposes:

    Page 0 = 6502 page zero address mode region.
    Page 1 = 6502 stack region.
    Pages 2-4 = OS database and user workspace.
    Pages 5-6 = User program workspace.
    Pages 7-XX = Bootable software area/free RAM.*
    Pages XX-top of RAM = Screen display list and data.*

  * Note that XX is a function of the screen graphics mode and the
    amount of RAM installed.

The paragraphs that follow describe how the OS uses RAM subregions,
and presents user program recomendations.

Page 0

The architecture of the 6502 microcomputer instruction set and
addressing modes gives page 0 special significance. References to
addresses in that page (0000 to 00FF) are faster, require fewer
instruction bytes, and provide the only mechanism for hardware
indirect addressing. Page 0 should be used sparingly so that all
possible users can have a portion of it. The OS permanently takes the
lower half of page 0 (0000 to 007F). This portion can never be used by
any outer environment unless the OS is completely disabled and all
interrupts to the OS are eliminated.

The upper half of page 0 (0080 to 00FF) is available to outer
environments with the following restriction: the floating point
package, if used, requires 00D4 through 00FF.

Page 1

Page 1 is the 6502 hardware stack region; JSR instructions, PHA
instructions, and interrupts all cause data bytes to be written to
page 1. Conversely RTS, PLA, and RTI instructions all cause data bytes
to be read from page 1. The 256 byte stack is adequate for normal
subroutine calls plus interrupt process nesting, so no restrictions
have been made on page 1 usage. It is obvious that a stack of this
size is totally inadequate for deeply recursive processes or for
nested processes with large local environments to be saved. So, for
sophisticated applications, software maintained stacks must be
implemented.

The 6502 stack pointer is initialized at power-up or system reset to
point to location 01FF. The stack then pushes downward toward 0100.
The stack will wrap around from 0100 to 01FF if a stack overflow
condition occurs, because of the nature of the 6502's 8-bit stack
pointer register.

OS Data Base

Locations 0200 through 047F are allocated by the OS for working
variables, tables and data buffers. Portions of this region can be
used only after you determine that nonconflict with the OS
is guaranteed. For example, the printer and cassette buffers could be
used if I/O operations to these devices are impossible within the
controlling environment. The amount of work involved in determining
nonconflict seems to be completely out of line with the benefits to be
gained (except for a few trivial cases) and it is recommended that
pages 2 through 4 not be used except by the OS.

User Workspace

Locations 0480 through 06FF are dedicated for outer environment use
except when the floating point package is used. The floating point
package uses locations 057E through 05FF.

Boot Region

Page 7 is the start of the "boot region." When software is booted from
either the diskette or the cassette, it can start at the lowest free
memory address (that is 0700) and proceed upward (although it can also
start at any address above 0700 and below the screen display list).
The top of this region defines the start of the "free memory" region.
When the boot process is complete, a pointer in the data base contains
the address of the next available location above the software just
booted. When no software has been booted, this pointer contains the
value 0700.

Screen Display List and Data

When the OS is handling the screen display, the display list that
defines the screen characteristics and the current data that is
contained on the screen are placed at the high address end of RAM. The
bottom of this region defines the end of the free memory region and
its location is a function of the screen mode currently in effect. A
pointer in the data base contains the address of the last available
location below the screen region.

Free Memory Region

The free memory region is all the RAM between the end of the boot
region and the start of the screen region. The outer level application
is responsible for managing the free memory region.

CARTRIDGES A AND B

There are two 8K regions reserved for plug-in cartridges. Cartridge B,
that is the right-hand cartridge slot found only in the ATARI 800
Home Computer, has been allocated memory addresses 8000
through 9FFF. Cartridge A (the left-hand cartridge slot in the ATARI
800 Computer console, and the only slot in the ATARI 400 Computer
console) has been allocated memory addresses A000 through BFFF and
optionally 8000 through BFFF, for 16K cartridges. If a RAM module is
plugged into the last slot such as to overlay any of these addresses,
the RAM takes precedence as long as a cartridge is not inserted.
However, if a cartridge is inserted, it will disable the entire
conflicting RAM module in the last slot in 8K increments.

MAPPED I/O

The 6502 performs input/output operations by addressing the external
support chips as memory; some chip registers are read/write while
others are read-only or write-only (the ATARI Home Computer
Hardware Manual gives descriptions of all of the external registers).
While the entire address space from D000 to D7FF has been allocated
for I/O decoding, only the following subregions are used:

        D000-D01F = CTIA
        D200-D21F = POKEY
        D300-D31F = PIA
        D400-D41F = ANTIC

Figure 4-2.   Mapped I/O

RESIDENT OS AND FLOATING POINT PACKAGE ROM

The region from D800 through FFFF always contains the OS and the
floating point package. Care should be taken to avoid using any entry
points that are not guaranteed not to move, to allow for the
possibility that another, but functionally compatible, OS can be
generated in the future. The OS contains many vectored entry points at
the end of the ROM and in RAM that will not move. The floating point
package is not vectored, but all documented entry points will be
fixed: Do not use undocumented routines found by scanning the listing.
A list of the fixed ROM vectors can be found in Appendix J.

CENTRAL DATA BASE DESCRIPTION

See Appendix L.

MEMORY DYNAMICS

The free memory region is the area between the end of the boot region
and the start of the screen region. As such, its limits are variable.
MEMLO [02E7] defines the bottom of the free region, and MEMTOP [02E5]
defines the top of the region. This section presents the conditions
that cause the setup or alteration of these variables.

System Initialization Process

The OS determines the extent of the lowest block of contiguous RAM, and saves the limits. The Screen Editor is then opened, thus setting a new (and lower) value in MEMTOP. Diskette or cassette-booted software might be brought into memory, that would probably set a new (and higher) value in MEMLO (see Section 7). MEMLO and MEMTOP will define the maximum amount of free memory available when the application program finally gets control. That amount of free memory can later decrease, as described in the next paragraph.

Changing Screen Modes

The Display Handler interprets the variable APPMHI '[000E]' to contain the address below which MEMTOP cannot extend. This allows you to protect the portion of free memory space that you are using from being overwritten as a result of screen mode change. The display handler will set the screen for mode 0, update MEMTOP, and return an error status to you, if it determines that the screen memory will extend below APPMHI as a result of a screen mode change. In other cases the Display Handler effects the desired mode change and updates MEMTOP.

5 I/O SUBSYSTEM

This section discusses the I/O subsystem of the Operating System. The I/O subsystem comprises a collection of routines that allow you to access peripheral and local devices at three different levels. The CIO (Central I/O Utility), provides the highest level, device independent access to devices. The second level allows communication with the device handlers. The lowest level is the SIO (Serial I/O bus Utility) routine. Any lower level access to a device involves the direct reading and writing of the hardware registers associated with the device.

The data byte is the basic unit of input/output. A data byte can contain either "binary" (non text) information, or encoded text information. The text encoding scheme supported by the OS is called ATASCII, derived from the words "ATARI ASCII." Most ATASCII codes are the same as ASCII, with the primary deviations being the control codes. Appendix D shows the ATASCII character set, and Appendices E, F, and G show device-specific implementations for the display, keyboard, and printer.

The structure of the I/O subsystem is shown on the following page.

```
                          +------------+
                          :   user     :
           +--------------: program    :----------------------------+
           :              +------------+                            :
           :       +------------+       :                           :
           :       :  IOCB's    :*********:                         :
           :       +------------+       :                           :
          ++------------+       +------------+                      :
          : Resident    :       :   CIO      :     +----------+     :
          : Handler     :       : Utility    :     :  DCB     :*******:
          : Vector Table:       +------------+     +----------+     *  :
          ++------------+       :                                  *  :
           :                    :                                  *  :
           :    +------------+  :         +------------+           *  :
           :    :  ZIOCB     :  :         :Disk File:  *           :
           :    +------------+  :         +----:----: Manager :----+  :
           :         *          :   +----: Manager :-----+ :
           :         *          :   :    +----------+     :  :
          +-+---+-----------+----+---+--+---....---+      :  :
           :               :              :           :       :  :
        +----------+  +----------+  +----------+  +----------+  +----------+
        : Printer  :  : Cassette :  :  ...     :  : Keyboard :  : Disk     :
        : Handler  :  : Handler  :  : Handler  :  : Handler  :  : Handler  :
        +----------+  +----------+  +----------+  +----------+  +----------+
           :              :              :                          :
          +----------+----------+-----+------+----....------------------+
           :                          :
        +----------+                  :
        :  DCB     :*********:         :
        +----------+                  :
                          +----------+
                          :   SIO    :
                          : Utility  :
                          +----------+
```
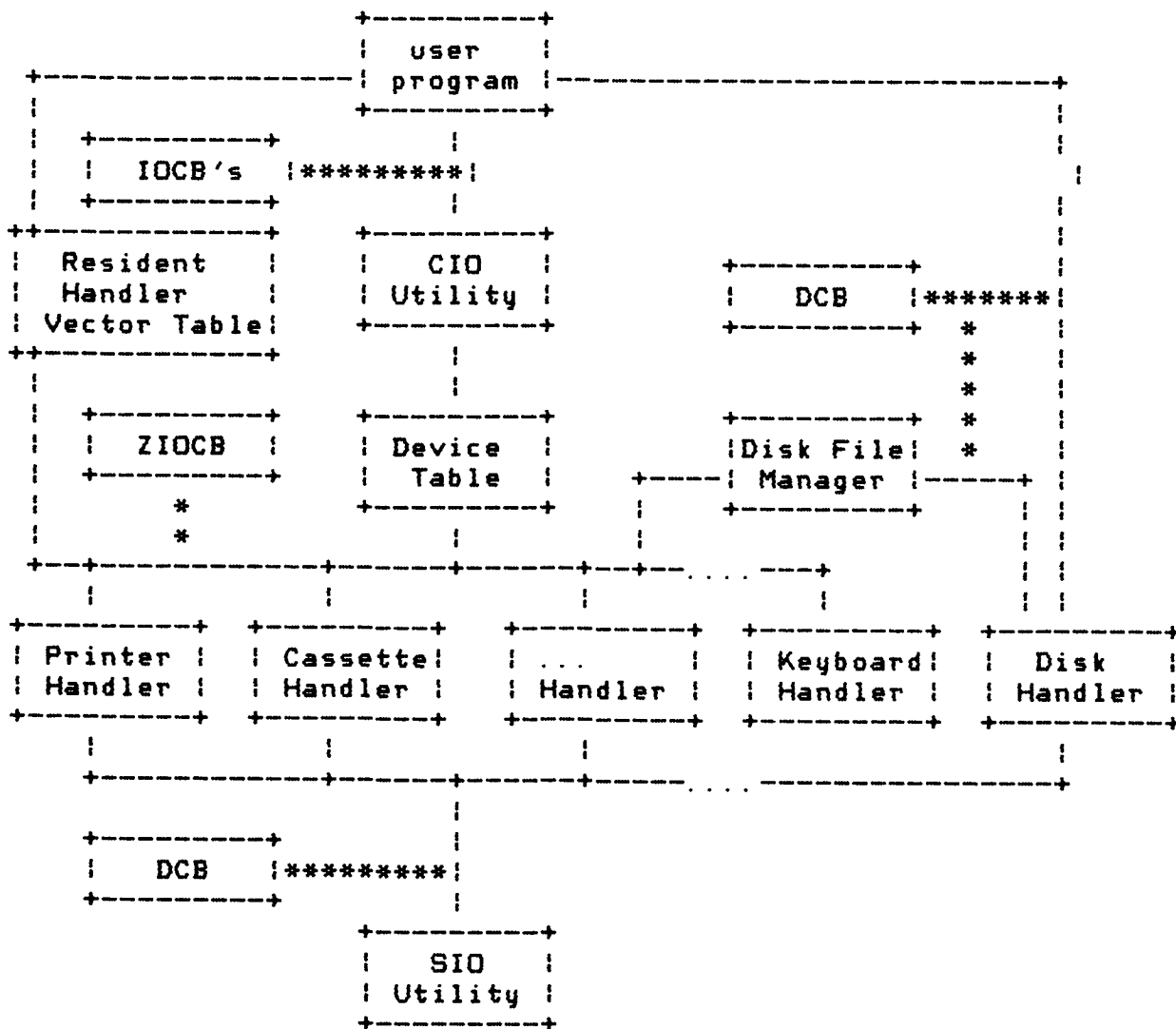
Where: ---- shows a control path. **** shows the data structure
               required for a path.

Note the following:

   o   The Keyboard/Display/Screen Editor Handlers don't use SIO.

   o   The Diskette handler cannot be called directly from CIO.

   o   The DCB is shown twice in the diagram.

Figure 5-1   I/O Subsystem Structure Flow Diagram

CENTRAL I/O UTILITY

The Central I/O Utility provides you with a single interface in which
to access all of the system peripheral devices in a device-independent
manner. The minimum unit of data transfer is the data byte. The CIO
also supports multiple byte transfers. All I/O operations are
performed on a "return-to-user-when-complete" basis; there is no way
to initiate concurrent "overlapped" I/O processes.

I/O is organized by "files," where a file is a sequential
collection of data bytes. A file can or may not contain textual
data and it can or may not be organized by "records," where a
record is a contiguous group of bytes terminated by an EOL (End of
Line) character. Some files are synonymous with a device (as with
the printer and the Screen Editor), while other devices can contain
multiple files, each with a unique name (as with the disk drive).

CIO allows you to access up to eight independent device/files
at one time, because there are eight I/O Control Blocks (IOCB's) in
the system. Each of the IOCB's can be assigned to control any
device/file because there are no preferred assignments, except that
IOCB #0 is assigned to the Screen Editor at power-up and
system reset.

To access a peripheral, you first set up an IOCB for the OPEN
command, that supplies the system name for the device to be
accessed (e.g. K:, for the keyboard, P:, for the printer, D:STARS
for a diskette file named 'STARS', etc). You then call the CIO,
telling it to examine the IOCB to find the OPEN information. CIO
attempts to find the specified device/file and returns a status
byte indicating the success of the search. If the specified
device/file can be found by CIO, then CIO stores control
information in the IOCB. The IOCB is now used for as long as that
file is open.

Once a file is open, it can then be accessed using data-read or
data-write types of commands; in general, reading can proceed until
there is no more data to read (End of File) and writing can proceed
until there is no more medium to store data on (End of Medium),
although neither reading nor writing need proceed to that point.
The reading and writing of data generally occurs into and out of
user-supplied data buffers (although a special case allowing single
byte transfers using the 6502 A register is provided).

When there are no more accesses to be performed on an open
device/file, you perform the close operation. This
accomplishes two functions:

o   It terminates and makes permanent an output file (essential
         for diskette and cassette).

o   It releases that IOCB to be used for another I/O operation.

CIO Design Philosophy

The CIO utility was designed specifically to meet the following design criteria.

o   The transfer of data is device independent.

o   Byte-at-a-time, multiple byte and record-aligned accesses are supported.

o   Multiple device/files can be accessed concurrently.

o   Error handling is largely device independent.

o   New device handlers can be added without altering the system ROM.

Device Independence

CIO provides device independence by having a single entry point for all devices (and for all operations) and by having a device-independent calling sequence. Once a device/file is opened, data transfers occur with no regard to the actual device involved. Uniform rules for handling byte- and record-oriented data transfers allow the actual device storage block sizes to be transparent to you.

Data Access Methods

The CIO supports two file access methods: byte-aligned and record-aligned.

Byte-aligned accesses allow you to treat the device/file as a sequential byte stream; any number of bytes can be read or written and the following operation will continue where the prior one left off. Records are of no consequence in this mode, and reads or writes can encompass multiple records if desired.

Record-aligned accesses allow you to deal with the data stream at a higher level, that of the data record or "line of text." Each and every write operation creates a single record (by definition). Each read operation assures that the following read operation will start at the beginning of a record. Record-aligned accesses cannot deal with portions of more than one record at a time. Record-aligned accesses are useful only with text data or with binary data guaranteed not to contain the EOL character ($9B) as data.

Note that any file can be accessed using the byte-aligned access method, regardless of how the file was created. But not all files can be successfully read using record-aligned accesses; the file

must contain EOL characters at the end of each record and at no
other place.

Multiple Device/File Concurrency

Up to eight device/files can be accessed concurrently using CIO,
each operating independently of the others.

Unified Error Handling

All error detection and recovery occurs within the CIO subsystem.
The status information that reaches you is in the form of a
status byte for each device/file. Error codes are device
independent as much as possible (see Appendix B).

Device Expansion

Devices are known by single character names such as K or P, and a
number of device handlers are part of the resident system ROM.
However, additional device handlers can be added to the system
using the RAM-resident device table; this is normally done at
power-up time as with the diskette boot process, but can be done at
any point in time.

CIO Calling Mechanism

The input/output control block (IOCB) is the primary parameter
passing structure between you and CIO. There are eight IOCB's
in the system, arranged linearly in RAM as shown below:

```
        +--------+  low address [0340]
        : IOCB 0 :
        +--------+
        : IOCB 1 :
        +--------+
        =        =
        +--------+
        : IOCB 6 :
        +--------+
        : IOCB 7 :
        +--------+  high address
```

Figure 5-2   CIO Calling Mechanism

One IOCB is required for each open device/file. Any IOCB can be used to control any device/file, although IOCB 0 is normally assigned to the Screen Editor (E:). You perform a typical I/O operation by:

o   Inserting appropriate parameters into an IOCB of your choosing

o   Putting the IOCB number times 16 into the 6502 X register

o   Performing a JSR to the CIO entry point CIOV [E456].

CIO returns to you when the operation is complete or if an error was encountered. The operation status is in the IOCB used, as well as in the 6502 Y register. The 6502 condition codes will also reflect the value in the Y register. In some cases a data byte will be in the 6502 A register. The X register will remain unchanged for all operations and conditions.   An example is shown below:

```
        IOCB2X = $20              ; INDEX FOR IOCB #2.

                LDX       #IOCB2X
                JSR       CIOV
                CPY       #0        ;(optional)
                BMI       ERROR
```

This sector describes each IOCB byte, with its file name and address. Each IOCB is 16 bytes long. Some bytes can be altered by you and some are reserved for use by CIO and/or the device handlers.

    Handler ID -- ICHID [0340]

The handler ID is an index into the system device table (see Section 9) and is not user-alterable. This byte is set by CIO as the result of an OPEN command and is left unchanged until the device/file is closed, at that time CIO will set the byte to $FF.

    Device Number -- ICDNO [0341]

The device number is provided by CIO as the result of an OPEN command and is not user-alterable. This byte is used to distinguish between multiple devices of the same type, such as D1: and D2:.

Command Byte -- ICCMD [0342]

You set the command byte. It specifies the command to be
performed by the CIO. This byte is not altered by CIO.

Status -- ICSTA [0343]

The CIO conveys operation status to you with the command
status byte as a result of each and every CIO call. Each and
every CIO call updates the command status byte. The most
significant (sign) bit is a one for error conditions and zero for
non-error conditions, and the remaining bits represent an error
number. See Appendix B for a list of status codes.

Buffer Address -- ICBAL [0344] and ICBAH [0345]

You set this 2-byte pointer; it is not altered by CIO. The
pointer contains the address of the beginning (low address) of a
buffer that:

o Contains data for read and write operations

o Contains the device/filename specification for the OPEN
command.

You can alter the pointer at any time.

PUT Address -- ICPTL [0346] and ICPTH [0347]

The CIO sets this 2-byte pointer at OPEN time to the handler's
PUT CHARACTER entry point (- 1). The pointer was provided to
accommodate the people writing the ATARI BASIC cartridge, and has
no legitimate use in the system. This variable is set to point to
CIO's "IOCB not OPEN" routine on CLOSE, Power-up and
[SYSTEM. RESET].

Buffer Length/Byte Count -- ICBLL [0348] and ICBLH [0349]

You set this 2-byte count to indicate the size of the data
buffer pointed to by ICBAL and ICBAH for read and write
operations. It is not required for OPEN. After each read or write
operation, CIO will set this parameter to the number of bytes
actually transferred into or out of the data buffer. For
record-aligned access, the record length can well be less than
the buffer length. Also an end of file condition or an error can
cause the byte count to be less than the buffer length.

Auxiliary Information -- ICAX1 [034A] and ICAX2 [034B]

You set these 2-bytes. They contain information that is
used by the OPEN command process and/or is device-dependent.

For OPEN, two bits of ICAX1 are always used to specify the OPEN
direction as shown below, where R is set to 1 for input (read)
enable and W is set to 1 for output (write) enable.

```
    7       3 2   0
   +-+-+-+-+-+-+-+-+
   ! ! ! ! !W!R! ! !
   +-+-+-+-+-+-+-+-+
```

ICAX1 is not altered by CIO. You should not alter ICAX1
once the device/file is open.

The remaining bits of ICAX1 and all of ICAX2 contain only
device-dependent data and are explained later in this section.

   Remaining Bytes (ICAX3-ICAX6)

The handler reserves the four remaining bytes for processing the
I/O command for CIO. There is no fixed use for these bytes. They
are not user-alterable except as specified by the particular
device descriptions. These bytes will be referred to as ICAX3,
ICAX4, ICAX5 and ICAX6, although there are no equates for those
names in the OS equate file.

CIO Functions

The CIO supports records and blocks and the handlers support
single bytes. All of the system handlers support one or more
of the eight basic functions subject to restrictions based
upon the direction of data transfer (e.g. one cannot read data
from the printer). The basic functions are: OPEN, CLOSE, GET
CHARACTERS, PUT CHARACTERS, GET RECORD, PUT RECORD, GET STATUS,
and SPECIAL.

   OPEN -- Assign Device/Filename to IOCB and Ready for Access

A device/file must be opened before it can be accessed. This
process links a specific IOCB to the appropriate device
handler, initializes the device/file, initializes all CIO
control variables, and passes device-specific options to the
device handler.

You set up the following IOCB parameters prior to calling CIO for an OPEN operation:

COMMAND BYTE = $03

BUFFER ADDRESS = pointer to a device/filename specification.

AUX1 = OPEN direction bits, plus device-dependent information.

AUX2 = device-dependent information.

After an OPEN operation, CIO will have altered the following IOCB parameters:

HANDLER ID = index to the system device table; this is used only by CIO and must not be altered.

DEVICE NUMBER = device number taken from the device/filename specification and must not be altered.

STATUS = result of OPEN operation; see Appendix B for a list of the possible status codes. In general, a negative status will indicate a failure to open properly.

PUT ADDRESS = pointer to the PUT CHARACTERS routine for the device handler just opened.

It is recommended that this pointer not be used.

CLOSE -- Terminate Access to Device/File and Release IOCB.

You issue a CLOSE command after you are through accessing a given device/file. The CLOSE process completes any pending data writes, goes to the device handler for any device-specific actions, and then releases the IOCB.

You set the following IOCB parameter prior to calling CIO:

COMMAND BYTE = $0C

The CIO alters the following IOCB parameters as a result of the CLOSE operation:

HANDLER ID = $FF

STATUS = Result of CLOSE operation.

PUT ADDRESS = pointer to "IOCB not OPEN" routine.

GET CHARACTERS -- Read n Characters (Byte-Aligned Access)

The specified number of characters are read from the device/file
to the user-supplied buffer. EOL characters have no termination
features when using this function; there can be no EOL, or many
EOL's, in the buffer after operation completion. There is a
special case provided that passes a single byte of data in the
6502 A register when the buffer length is set to zero.

You set the following IOCB parameters prior to calling CIO:

        COMMAND BYTE = $07

        BUFFER ADDRESS = pointer to data buffer.

        BUFFER LENGTH = number of bytes to read; if this is zero,
        the data will be returned in the 6502 A register only.

The CIO alters the following IOCB parameters as a result of the
GET CHARACTERS operation:

        STATUS = result of GET CHARACTERS operation.

        BYTE COUNT/BUFFER LENGTH = number of bytes read to the
        buffer. The BYTE COUNT will always equal the BUFFER LENGTH
        except when an error or an end-of-file condition occurs.

        PUT CHARACTERS -- Write n Characters (Byte-Aligned Access)

The specified number of characters are written from the user-supplied
buffer to the device/file. EOL characters have no buffer
terminating properties, although they have their standard meaning
to the device/file receiving them; no EOL's are generated by CIO.
There is a special case that allows a single character to be
passed to CIO in the 6502 A register if the buffer length is
zero.

You set the following IOCB parameters prior to initiating the PUT
CHARACTERS operation:

        COMMAND BYTE = $0B

        BUFFER ADDRESS = pointer to data buffer.

        BUFFER LENGTH = number of bytes of data in buffer.

The CIO alters the following IOCB parameter as a result of the
PUT CHARACTERS operation:

        STATUS = result of PUT CHARACTERS operation.

GET RECORD -- Read Up To n Characters (Record-Aligned Access)

Characters are read from the device/file to the user-supplied
buffer until either the buffer is full or an EOL character is
read and put into the buffer. If the buffer fills before an EOL
is read, then the CIO continues reading characters from the
device/file until an EOL is read,, and sets the status to
indicate that a truncated record was read. No EOL will be put at
the end of the buffer.

You set the following IOCB parameters prior to calling CIO:

    COMMAND BYTE = $05

    BUFFER ADDRESS = pointer to data buffer.

    BUFFER LENGTH = maximum number of bytes to read (including
    the EOL character).

The CIO alters the following IOCB parameters as a result of the
GET RECORD operation:

    STATUS = result of GET RECORD operation.

    BYTE COUNT/BUFFER LENGTH = number of bytes read to data
    buffer; this can be less than the maximum buffer length.

PUT RECORD -- Write Up To n Characters (Record-Aligned Access)

Characters are written from the user-supplied buffer to the
device/file until either the buffer is empty or an EOL character
is written. If the buffer is emptied without writing an EOL
character to the device/file, then CIO will send an EOL after the
last user-supplied character.

You set the following IOCB parameters prior to calling CIO:

    COMMAND BYTE = $09

    BUFFER ADDRESS = pointer to data buffer.

    BUFFER LENGTH = maximum number of bytes in buffer.

The CIO alters the following IOCB parameter as a result of the
PUT RECORD operation:

    STATUS = result of PUT RECORD operation.

GET STATUS -- Return Device-Dependent Status Bytes

The device controller is sent a STATUS command, and the
controller returns four bytes of status information that are
stored in DVSTAT [02EA].

You set the following IOCB parameters prior to calling CIO:

    COMMAND BYTE = $0D

    BUFFER ADDRESS = pointer to a device/filename specification
    if the IOCB is not already OPEN; see the discussion of the
    implied OPEN option below.

After a GET STATUS operation, CIO will have altered the following
parameters:

    STATUS = result of GET STATUS operation; see Appendix B for
    a list of the possible status codes.

    DVSTAT = the four-byte response from the device controller.

    SPECIAL -- Special Function

Any command byte value greater than $0D is treated by CIO as a
special case. Since CIO does not know what the function is, CIO
transfers control to the device handler for complete processing
of the operation.

The user sets the following IOCB parameters prior to
calling CIO:

    COMMAND BYTE > $0D

    BUFFER ADDRESS = pointer to a device/filename specification
    if the IOCB is not already open; see the discussion of the
    implied OPEN option below.

    Other IOCB bytes can be set up, depending upon the specific
    SPECIAL command being performed.

After a SPECIAL operation, CIO will have altered the following
parameters:

    STATUS = result of SPECIAL operation; see Appendix B for a
    list of the possible status codes.

    Other bytes can be altered, depending upon the specific
    SPECIAL command.

Implied OPEN Option

The GET STATUS and SPECIAL commands are treated specially by CIO;
they can use an already open IOCB to initiate the process or they
can use an unopened IOCB. If the IOCB is unopened, then the
buffer address must contain a pointer to a device/filename
specification, just as for the OPEN command; CIO will then open
that IOCB, perform the specified command and then close the IOCB
again.

Device/Filename Specification

As part of the OPEN command, the IOCB buffer address parameter
points to a device/filename specification, that is a string of
ATASCII characters in the following format:

        <specification> ::= <device>[<number>]:[<filename>]<eol>

        <device> ::= C|D|E|K|P|R|S
        <number> ::= 1|2|3|4|5|6|7|8
        <filename> has device-dependent characteristics.
        <eol> ::= $9B

The following devices are supported at this writing:

        C = Cassette drive
        D1 through D8 = Floppy diskette drives *
        E = Screen Editor
        K = Keyboard
        P = 40-column printer
        P2 = 80-column printer *
        R1 through R4 = RS-232-C interfaces *
        S = Screen display

Devices flagged by asterisks (*) are supported by nonresident
handlers.

If <number> is not specified, it is assumed to be 1.

The following examples show valid device/filename specifications:

        C: Cassette
        D2:BDAT   File "BDAT" on disk drive #2
        D:HOLD   File "HOLD" on disk drive #1
        K: Keyboard

I/O Example

The example provided in this section illustrates a simple example of
an I/O operation using the CIO routine.

```
; This code segment illustrates the simple example of reading
; text lines (records) from a diskette file named TESTER on disk
; drive #1. All symbols used are equated within the program
; although many of the symbols are in the OS equate file.

; The program performs the following steps:
;
;       1. Opens the file 'D1:TESTER' using IOCB #3.
;       2. Reads records until an error or EOF is reached.
;       3. Closes the file.

; I/O EQUATES

EOL=       $9B                        ; END OF LINE CHARACTER.
IOCB3=     $30                        ; IOCB #3 OFFSET (FROM IOCB #0).

ICHID=     $0340                      ; (HANDLER ID -- SET BY CIO).
ICDNO=     ICHID+1                    ; (DEVICE # -- SET BY CIO).
ICCOM=     ICDNO+1                    ; COMMAND BYTE.
ICSTA=     ICCOM+1                    ; STATUS BYTE -- SET BY CIO.
ICBAL=     ICSTA+1                    ; BUFFER ADDRESS (LOW).
ICBAH=     ICBAL+1                    ; BUFFER ADDRESS (HIGH).
ICPTL=     ICBAH+1
ICPTH=     ICPTL+1
ICBLL=     ICPTH+1                    ; BUFFER LENGTH (LOW).
ICBLH=     ICBLL+1                    ; BUFFER LENGTH (HIGH).
ICAX1=     ICBLH+1                    ; AUX 1.
ICAX2=     ICAX1+1                    ; AUX 2.

OPEN=      $03                        ; OPEN COMMAND.
GETREC=    $05                        ; GET RECORD COMMAND.
CLOSE=     $0C                        ; CLOSE COMMAND.

OREAD=     $04                        ; OPEN DIRECTION = READ.
OWRIT=     $08                        ; OPEN DIRECTION = WRITE.

EOF=       $88                        ; END OF FILE STATUS VALUE.

CIOV=      $E456                      ; CIO ENTRY VECTOR ADDRESS.


;
; FIRST INITIALIZE THE IOCB FOR FILE "OPEN".
;


           LDX       #IOCB3           ; SETUP TO ACCESS IOCB #3.
```

OPERATING SYSTEM CO16555 -- Section 5

```
        LDA     #OPEN           ; SETUP OPEN COMMAND.
        STA     ICCOM, X

        LDA     #NAME           ; SETUP BUFFER POINTER TO ...
        STA     ICBAL, X        ; ... POINT TO FILENAME.
        LDA     #NAME/256
        STA     ICBAH, X


        LDA     #OREAD          ; SETUP FOR OPEN READ.
        STA     ICAX1, X

        LDA     #0              ; CLEAR AUX 2.
        STA     ICAX2, X


;
;  "OPEN" THE FILE.
;

        JSR     CIOV            ; PERFORM "OPEN" OPERATION.
        BPL     TP10            ; STATUS WAS POSITIVE -- OK.

        JMP     ERROR           ; NO -- "OPEN" PROBLEM.


;
;  SETUP TO READ A RECORD.
;

TP10    LDA     #GETREC         ; SETUP "GET RECORD" COMMAND.
        STA     ICCOM, X

        LDA     #BUFF           ; SETUP DATA BUFFER POINTER.
        STA     ICBAL, X
        LDA     #BUFF/256
        STA     ICBAH, X


;
;  READ RECORDS.
;


LOOP    LDA     #BUFFSZ         ; SETUP MAX RECORD SIZE ...
        STA     ICBLL, X        ; ... PRIOR TO EVERY READ.
        LDA     #BUFFSZ/256
        STA     ICBLH, X

        JSR     CIOV            ; READ A RECORD.
        BMI     TP20            ; MAY BE END OF FILE.

;
;  A RECORD IS NOW IN THE DATA BUFFER "BUFF".  IT IS TERMINATED BY
```

; AN EOL CHARACTER, AND THE RECORD LENGTH IS IN "ICBLL" and "ICBLH".
; THIS EXAMPLE WILL DO NOTHING WITH THE RECORD JUST READ.
;

```
        JMP     LOOP                ; READ NEXT RECORD.


;
; NEGATIVE STATUS ON READ -- CHECK FOR END OF FILE.
;


TP20    CPY     #EOF                ; END OF FILE STATUS?
        BNE     ERROR               ; NO -- ERROR.

        LDA     #CLOSE              ; YES -- CLOSE FILE.
        STA     ICCOM, X

        JSR     CIOV                ; CLOSE THE FILE.

        JMP     *                   ; *** END OF PROGRAM ***


;
; DATA REGION OF EXAMPLE PROGRAM
;

NAME    . BYTE   "D1: TESTER", EOL

BUFFSZ= 80                          ; 80 CHARACTER RECORD MAX
                                                  (INCLUDES EOL).

BUFF=   *                           ; READ BUFFER.
*=      *+BUFFSZ
        . END
```

Figure 5-3   An I/O Example

Device-Specific Information

This section provides device-specific information regarding the
device handlers that interface to CIO.

   Keyboard Handler (K:)

The keyboard device is a read only device with a handler that
supports the following CIO functions:

     OPEN
     CLOSE
     GET CHARACTERS
     GET RECORD
     GET STATUS (null function)

The Keyboard Handler can produce the following error statuses:

     $80 -- [BREAK] key abort.
     $88 -- end-of-file (produced by pressing [CTRL] 3).

The Keyboard Handler is one of the resident handlers. It has a
set of device vectors starting at location E420.

The keyboard can produce any of the 256 codes in the ATASCII
character set (see Appendix F). Note that a few of the keyboard
keys do not generate data at the Keyboard Handler level. These
keys are described below:

  [/!\] - The ATARI key toggles a flag that enables/disables the
          inversion of bit 7 of each data character read. The
          Screen Editor editing keys are exempted from such
          inversion, however.

   CAPS - The [CAPS/LOWR] key provides three functions:

               [SHIFT][CAPS/LOWR] -- Alpha caps lock.
               [CNTRL][CAPS/LOWR] -- Alpha [CTRL] lock.
               [CAPS/LOWR]   -- Alpha unlock.

The system powers up and will system reset to the alpha
caps lock option.

Some key combinations are ignored by the handler, such as
[CTRL] 4 through [CTRL] 9, [CTRL] 0, [CTRL] 1, [CTRL] /, and
all key combinations in that the [SHIFT] and [CTRL] keys are
depressed simultaneously.

The [CTRL] 3 key generates an EOL character and returns EOF status.

The [BREAK] key generates an EOL character and returns BREAK status.

   CIO Function Descriptions

The device-specific characteristics of the standard CIO functions
(described earlier in this section) are detailed below:

   OPEN

The device name is K, and the handler ignores any device number
and filename specification, if included.

There are no device-dependent option bits in AUX1 or AUX2.

   CLOSE

No special handler actions.

   GET CHARACTERS and GET RECORD

The handler returns the ATASCII key codes to CIO as they are
entered, with no facility for editing.

   GET STATUS

The handler does nothing but set the status to $01.

Theory of Operation

Pressing a keyboard key generates an IRQ interrupt and vectors to
the Keyboard Handler's interrupt service routine (see Section 6).
The key code for the key pressed is then read and stored in data
base variable CH [02FC]. This occurs whether or not there is an
active read request to the Keyboard Handler, and effects a one-byte
FIFO for keyboard entry. See Appendix L (E8) for a discussion of
the auto repeat feature.

The Keyboard Handler monitors the CH variable for not containing
the value $FF (empty state) whenever there is an active read
request for the handler. When CH shows nonempty, the handler
takes the key code from CH and sets CH to $FF again. The key code
byte obtained from CH is not an ATASCII code and has the
following form:

```
        7               0
        +-+-+-+-+-+-+-+-+
        |C|S| key code  |
        +-+-+-+-+-+-+-+-+
```

Where: C = 1 if the [CTRL] key is pressed.
        S = 1 if the [SHIFT] key is pressed.

        The remaining six bits are the hardware key code.

The key code obtained is then converted to ATASCII using the
first of the following rules that applies:

   1. Ignore the code if the C and S bits are both set.
   2. If the C bit is set, process the key as a [CTRL] code.
   3. If the S bit is set, process the key as a [SHIFT] code.
   4. If [CTRL] lock is in effect, process alpha characters as CTRL
       codes, all others as lowercase.
   5. IF [SHIFT] lock is in effect, process alpha characters as SHIFT
       codes, all others as lowercase.
   6. Else, process as lowercase character.

Then: If the resultant code is not a Screen Editor control code,
and if the video inverse flag is set, then set bit 7 of the
ATASCII code (will cause inverse video when displayed).

KEY CODE TO ATASCII CONVERSION TABLE

Key Code	Key Cap	Lwr. Case	[SHIFT]	[CTRL]	Key Code	Key Cap	Lwr. Case	SHIFT	CTRL
00	L	6C	4C	0C	20	,	2C	5B	00
01	J	6A	4A	0A	21	SPACE	20	20	20
02	;	3B	3A	7B	22	.	2E	5D	60
03	--	--	--	--	23	N	6E	4E	0E
04	--	--	--	--	24	--	--	--	--
05	K	6B	4B	0B	25	M	6D	4D	0D
06	+	2B	5C	1E	26	/	2F	3F	--
07	*	2A	5E	1F	27	/|\	--	--	--
08	O	6F	4F	0F	28	R	72	52	12
09	--	--	--	--	29	--	--	--	--
0A	P	70	50	10	2A	E	65	45	05
0B	U	75	55	15	2B	Y	79	59	19
0C	RET	9B	9B	9B	2C	TAB	7F	9F	9E
0D	I	69	49	09	2D	T	74	54	14
0E	-	2D	5F	1C	2E	W	77	57	17
0F	=	3D	7C	1D	2F	Q	71	51	11
10	V	76	56	16	30	9	39	28	--
11	--	--	--	--	31	--	--	--	--
12	C	63	43	03	32	0	30	29	--
13	--	--	--	--	33	7	37	27	--
14	--	--	--	--	34	BACKS	7E	9C	FE
15	B	62	42	02	35	8	38	40	--
16	X	78	58	18	36	<	3C	7D	7D
17	Z	7A	5A	1A	37	>	3E	9D	FF
18	4	34	24	--	38	F	66	46	06
19	--	--	--	--	39	H	68	48	08
1A	3	33	23	9B*	3A	D	64	44	04
1B	6	36	26	--	3B	--	--	--	--
1C	[ESC]	1B	1B	1B	3C	CAPS	--	--	--
1D	5	35	25	--	3D	G	67	47	07
1E	2	32	22	FD	3E	S	73	53	13
1F	1	31	21	--	3F	A	61	41	01

* [CTRL] 3 returns EOF status.

A complement of this table (ATASCII to keystroke) is given in Appendix F.

Figure 5-4   Keycode to ATASCII Conversion Table

Display Handler (S:)

The display device is a read/write device with a handler
that supports the following CIO functions:

    OPEN
    CLOSE
    GET CHARACTERS
    GET RECORD
    PUT CHARACTERS
    PUT RECORD
    GET STATUS (null function)
    DRAW
    FILL

The Display Handler can produce the following error statuses:

    $84 -- Invalid special command.
    $8D -- Cursor out-of-range.
    $91 -- Screen mode > 11.
    $93 -- Not enough memory for screen mode selected.

The Display Handler is one of the resident handlers, and
therefore has a set of device vectors starting at location E410.

  Screen Modes

You can operate the display screen in any of 20
configurations (modes 1 through 8, with or without split
screen; plus mode 0, and modes 9 through 11 without split
screen). Mode 0 is the text displaying mode. Modes 1 through
11 are all graphics modes (although modes 2 and 3 do display a
subset of the ATASCII character set). Modes 9 through 11
require a GTIA chip to be installed in place of the standard
CTIA chip.

  TEXT MODE 0

In text mode 0 the screen is comprised of 24 lines of 40
characters per line. Program alterable left and right margins
limit the display area. They default to 2 and 39 (of a possible 0
and 39).

A program-controllable cursor shows the destination of the next character to be output onto the screen. The cursor is visible as the inverse video representation of the current character at the destination position.

The text screen data is internally organized as variable length logical lines. The internal representation is 24 lines when the screen is cleared. Each EOL marks the end of a logical line as text is sent to the screen. If more than 3 physical lines of text are sent, a logical line will be formed every 3 physical lines. The number of physical lines used to comprise a logical line (1 to 3) is always the minimum required to hold the data for that logical line.

The text screen "scrolls" upward whenever a text line at the bottom row of the screen extends past the right margin, or a text line at the bottom row is terminated by an EOL. Scrolling removes the entire logical line that starts at the top of the screen, and then moves all subsequent lines upward to fill in the void. The cursor also moves upward, if the logical line deleted exceeds one physical line.

All data going to or coming from the text screen is represented in 8-bit ATASCII code as shown in Appendix E.

   TEXT MODES 1 AND 2

In text modes 1 and 2 the screen comprises either 24 lines of 20 characters (mode 1), or 12 lines of 20 characters (mode 2). The left and right margins are of no consequence in these modes and there is no visible cursor. There are no logical lines associated with the data and in all regards these modes are treated as graphics modes by the handler.

Data going to or coming from the screen is in the form shown below:


```
        7                 0
        +-+-+-+-+-+-+-+-+
        ¦  C  ¦   D    ¦
        +-+-+-+-+-+-+-+-+
```

Where:C is the color/character-set select field

C Value	Color (default)	Color Register (see Appendix H)	Character Set CHBAS=$E0	Character Set CHBAS=$E2
0	green	(PF1)	! – ?	[HEART] [ARROW]
1	gold	(PF0)	! – ?	[HEART] [ARROW]
2	gold	(PF0)	@ – _	[DIAMOND][TRIANGLE]
3	green	(PF1)	@ – _	[DIAMOND][TRIANGLE]
4	red	(PF3)	! – ?	[HEART] [ARROW]
5	blue	(PF2)	! – ?	[HEART] [ARROW]
6	blue	(PF2)	@ – _	[DIAMOND][TRIANGLE]
7	red	(PF3)	@ – _	[DIAMOND][TRIANGLE]

D is a 5-bit truncated ATASCII code that selects the specific character within the set selected by the C field. See Appendix E for the graphics representations of the characters.

Data base variable CHBAS [02F4] allows for the selection of either of two data sets. The default value of $E0 provides the capital letters, numbers and punctuation characters; the alternate value of $E2 provides lowercase letters and the special character graphics set.

Figure 5-5   Text Modes 1 and 2 Data Form

GRAPHICS MODES (Modes 3 Through 11)

The screen has varying physical characteristics for each of the graphics modes as shown in Appendix H. Depending upon the mode, a 1 to 16 color selection is available for each pixel and the screen size varies from 20 by 12 (lowest resolution) to 320 by 192 (highest resolution) pixels.

There is no visible cursor for the graphics mode output.

Data going to or coming from the graphics screen is represented as 1 to 8-bit codes as shown in Appendix H and in the GET/PUT diagrams following.

SPLIT-SCREEN CONFIGURATIONS

In split-screen configurations, the bottom of the screen is reserved for four lines of mode 0 text. The text region is controlled by the Screen Editor, and the graphics region is controlled by the Display handler. Two cursors are maintained in this configuration so that the screen segments can be managed independently.

To operate in split-screen mode, the Screen Editor must first be
opened and then the Display Handler must be opened using a
separate IOCB (with the split-screen option bit set in AUX1).

CIO Function Descriptions

The device-specific characteristics of the standard CIO functions
(described earlier in this section) are detailed below:

   OPEN

The device name is S, and the handler ignores any device number an(
filename specification, if included.

The handler supports the following options:

```
                7               0
                +-+-+-+-+-+-+-+-+
        AUX1    |   |C|S|W|R|   |
                +-+-+-+-+-+-+-+-+
```

Where: C = 1 indicates to inhibit screen clear on OPEN.
        S = 1 indicates to set up a split-screen configuration (for
              modes 1 through 8 only).
        R and W are the direction bits (read and write).

```
                7               0
                +-+-+-+-+-+-+-+-+
        AUX2    |       | mode  |
                +-+-+-+-+-+-+-+-+
```

Where: mode is the screen mode (0 through 11).

Note: If the screen mode selected is 0, then the AUX1 C and
S options are assumed to be 0.

You share memory utilization with the Display Handler
information. Sharing is necessary because the Display Handler
dynamically allocates high address memory for use in generating
the screen display, and because different amounts of memory are
needed for the different screen modes. Prior to initiating an
OPEN command the variable APPMHI [000E] should contain the
highest address of RAM you need. The Screen handler
will open the screen only if no RAM is needed at or below that
address.

Upon return from a screen OPEN, the variable MEMTOP [02E5] will
contain the address of the last free byte at the end of RAM
memory prior to the screen-required memory.

As a result of every OPEN command, the following screen variables
are altered:

   The text cursor is enabled (CRSINH = 0). The tabs are set to
   the default settings (2 and 39). The color registers are set
   to the default values (shown in Appendix H).

   Tabs are set at positions 7,15,23,31,39,
   47,55,63,71,79,87,95,103,111,119.

   CLOSE

No special handler actions.

   GET CHARACTERS and GET RECORD

Returns data in the following screen mode dependent forms, where
each byte contains the data for one cursor position (pixel); there
is no facility for having the handler return packed graphics data.

```
       7               0
       +-+-+-+-+-+-+-+-+
       !    ATASCII    !        Mode 0
       +-+-+-+-+-+-+-+-+


       +-+-+-+-+-+-+-+-+
       ! C !    D      !        Modes 1,2 -- C = color/data
       +-+-+-+-+-+-+-+-+                               set.

                                D = truncated ATASCII.

       +-+-+-+-+-+-+-+-+
       !    zero    ! D !        Modes 3,5,7 -- D = color.
       +-+-+-+-+-+-+-+-+


       +-+-+-+-+-+-+-+-+
       !     zero    !D!        Modes 4,6,8 -- D = color.
       +-+-+-+-+-+-+-+-+


       +-+-+-+-+-+-+-+-+
       ! zero !    D   !        Modes 9,10,11 -- D = data.
       +-+-+-+-+-+-+-+-+
```

Figure 5-6   Graphics Mode 3-11 GET Data Form

The cursor moves to the next position as each data byte is
returned.   For mode 0, the cursor will stay within the specified
margins; for all other modes, the cursor ignores the margins.

PUT CHARACTERS and PUT RECORD

The handler accepts display data in the following screen mode
dependent forms; there is no facility for the handler to receive
graphics data in packed form.

```
      7                 0
      +-+-+-+-+-+-+-+-+
      !    ATASCII    !        Mode 0
      +-+-+-+-+-+-+-+-+


      +-+-+-+-+-+-+-+-+
      ! C !    D      !        Modes 1,2 -- C = color/data
      +-+-+-+-+-+-+-+-+                         set,
                               D = truncated ATASCII.


      +-+-+-+-+-+-+-+-+
      !      ?      ! D !      Modes 3,5,7 -- D = color.
      +-+-+-+-+-+-+-+-+


      +-+-+-+-+-+-+-+-+
      !      ?       !D!       Modes 4,6,8 -- D = color.
      +-+-+-+-+-+-+-+-+


      +-+-+-+-+-+-+-+-+
      !   ?   !   D   !        Modes 9,10,11 -- D = data.
      +-+-+-+-+-+-+-+-+
```

Figure 5-7   Graphics Mode 3-11 PUT Data Form

NOTE: For all modes, if the output data byte equals $9B (EOL), that
byte will be treated as an EOL character; and if the output
data byte equals $7D (CLEAR) that byte will be treated as a
screen-clear character.

The cursor moves to the next cursor position as each data byte is
written. For mode 0, the cursor will stay within the specified
margins; for all other modes, the cursor ignores the margins.

While outputting, the Display Handler monitors the keyboard to
detect the pressing of the [CTRL] 1 key combination. When this
occurs, the handler loops internally until that key combination
is pressed again: This effects a stop/start function that
freezes the screen display. Note that there is no ATASCII code
associated with either the [CTRL] 1 key combination or the
start/stop function. The stop/start function can be controlled
only from the keyboard (or by altering database variable CH as
discussed in Appendix L, E4).

GET STATUS

No handler action except to set the status to $01.

DRAW

This special command draws a simulated "straight" line from the
current cursor position to the location specified in ROWCRS
[0054] and COLCRS [0055]. The color of the line is taken from the
last character processed by the Display Handler or Screen Editor.
To force the color, store the desired value in ATACHR [02FB]. At
the completion of the command, the cursor will be at the location
specified by ROWCRS and COLCRS.

The value for the command byte for DRAW is $11.

FILL

This special command fills an area of the screen defined by two
lines with a specified color. The command is set up the same as
in DRAW, but as each point of the line is drawn, the routine
scans to the right performing the procedure shown below (in
PASCAL notation):

```
    WHILE PIXEL [ROW,COL] = 0 DO
        BEGIN
            PIXEL [ROW,COL] := FILDAT;
            COL := COL + 1;
            IF COL > Screen right edge THEN COL := 0
        END;
```

An example of a FILL operation is shown below:

```
                                                           + 1
                                                          +
        +----------------------------------------+
        +----------------------------------------+
    4 +----------------------------------------+
                                               +
                                            + 2
```

Where: '-' represents the fill operation.
       '+' are the line points, with '+' for the endpoints.

       1 -- set cursor and plot point.
       2 -- set cursor and DRAW line.
       3 -- set cursor and plot point.
       4 -- set fill data value, set cursor, and FILL.

FILDAT [02FD] contains the fill data, and ROWCRS and COLCRS
contain the cursor coordinates of the line endpoint. The value
in ATACHR [02FB] will be used to draw the line; ATACHR always
contains the last data read or written, so if the steps above
are followed exactly, ATACHR will not have to be modified.

The value for the command byte for FILL is $12.

User-Alterable Data Base Variables

Certain functions of the Display Handler require you to
examine and/or alter variables in the OS database. The following
describes some of the more commonly used handler variables. (see
Appendix L, B1-55, for additional descriptions).

Cursor Position

Two variables maintain the cursor position for the graphics
screen or mode 0 text screen. ROWCRS [0054] maintains the display
row number; and COLCRS [0055] maintains the display column
number. Both numbers range from 0 to the maximum number of
rows/columns, − 1. The cursor can be set outside of the defined
text margins with no ill effect. You can read and write this
region. The home position (0,0) for both text and graphics is the
upper left corner of the screen.

ROWCRS is a single byte. COLCRS is maintained at 2-bytes, with
the least significant byte being at the lower address.

When you alter these variables, the screen representation
of the cursor will not move until the next I/O operation
involving the display is performed.

Inhibit/Enable Visible Cursor Display

You can inhibit the display of the text cursor on the screen
by setting the variable CRSINH [02F0] to any nonzero value.
Subsequent I/O will not generate a visible cursor.

You can enable the display of the text cursor by setting
CRSINH to zero. Subsequent I/O will then generate a visible
cursor.

Text Margins

The text screen has user-alterable left and right margins. The OS
sets these margins to 2 and 39. The variable LMARGN [0052]
defines the left margin, and the variable RMARGN [0053] defines
the right margin. The leftmost margin value is 0 and the

rightmost margin value is 39.

The margin values inclusively define the useable portion of the
screen for all operations in that you do not explicitly
alter the cursor location variables as described prior to this
paragraph.

   Color Control

The OS updates hardware color registers using data from the OS
data base as part of normal Stage 2 VBLANK processing (see Section
6). Shown below are the data base variable names, the hardware
register names, and the function of each register. See Appendix H
for the mode dependent uses for the registers.

          Data Base   Hardware   Function

          COLOR0   COLPF0   PF0 -- Playfield 0.
          COLOR1   COLPF1   PF1 -- Playfield 1.
          COLOR2   COLPF2   PF2 -- Playfield 2.
          COLOR3   COLPF3   PF3 -- Playfield 3.
          COLOR4   COLBK   BAK -- Playfield background.

          PCOLR0   COLPM0   PM0 -- Player/missile 0.
          PCOLR1   COLPM1   PM1 -- Player/missile 1.
          PCOLR2   COLPM2   PM2 -- Player/missile 2.
          PCOLR3   COLPM3   PM3 -- Player/missile 3.

Theory of Operation

The Display Handler automatically sets up all memory resources
required to create and maintain the screen display at OPEN time.
The screen generation hardware requires that two distinct data
areas exist for graphics modes: 1) a display list and 2) a
screen data region. A third data area must exist for text modes.
This data area defines the screen representation for each of the
text characters. Consult the ATARI Home Computer
Hardware Manual for a complete understanding of the material that
is to follow.

The simplified block diagram below shows the relationships
between the memory and hardware registers used to set up a screen
display (without player/missile graphics) by the OS Note that
the hardware registers allow for many other possibilities.

```
                        DATA BASE              HARDWARE
                        VARIABLE               REGISTER
                                               (Updated every
                                                  VBLANK)

                        +-----------+
                        : MEMTOP    :
                        +           +
                        :           :
                        +---+-------+
         +----------------+
         :    +-----------------------------------------+
         :  : :                                         :
  +---+-+--------+ +-----------+       +----------+   :
  :     Display  : :  SDLSTL   :       :  DLISTL  : :
  :     List     : +           +------>+          +-+
  =            = :  SDLSTH   :       :  DLISTH  :
  :              : : :        :       :          :
  +--------------+ +-----------+       +----------+
         :              :
         :              :   +-----------+
         : Screen Data :<-- SAVMSC     :
         =            = +           +
         : Graphics    :  :        :
         : and/or      : +-----------+
         :    Text     :
  +--------------+
End of RAM memory
         +--------------------------------------------+
         :          +-----------+     +----------+    :
         :          : CHBAS=E0 :--->: CHBASE   +------+
         :          +-----------+     +----------+
  +------+------+
  : Specials and: E000
  : Numbers     :
  +-------------+
  : Capital     : E100
  : Letters     :
  +-------------+
  : Special     : E200
  : Graphics    :
  +-------------+
  : Lowercase   : E300
  : Letters     :
  +-------------+
```

```
        +---------+        +---------+
        ! COLOR 0 !        ! COLPF0  !
        =                  =-->! COLPF1  !
        ! COLOR 1 !        ! COLPF2  !
        ! COLOR 2 !        ! COLPF3  !
        ! COLOR 3 !        ! COLBK   !
        ! COLOR 4 !        +---------+
        +---------+
```
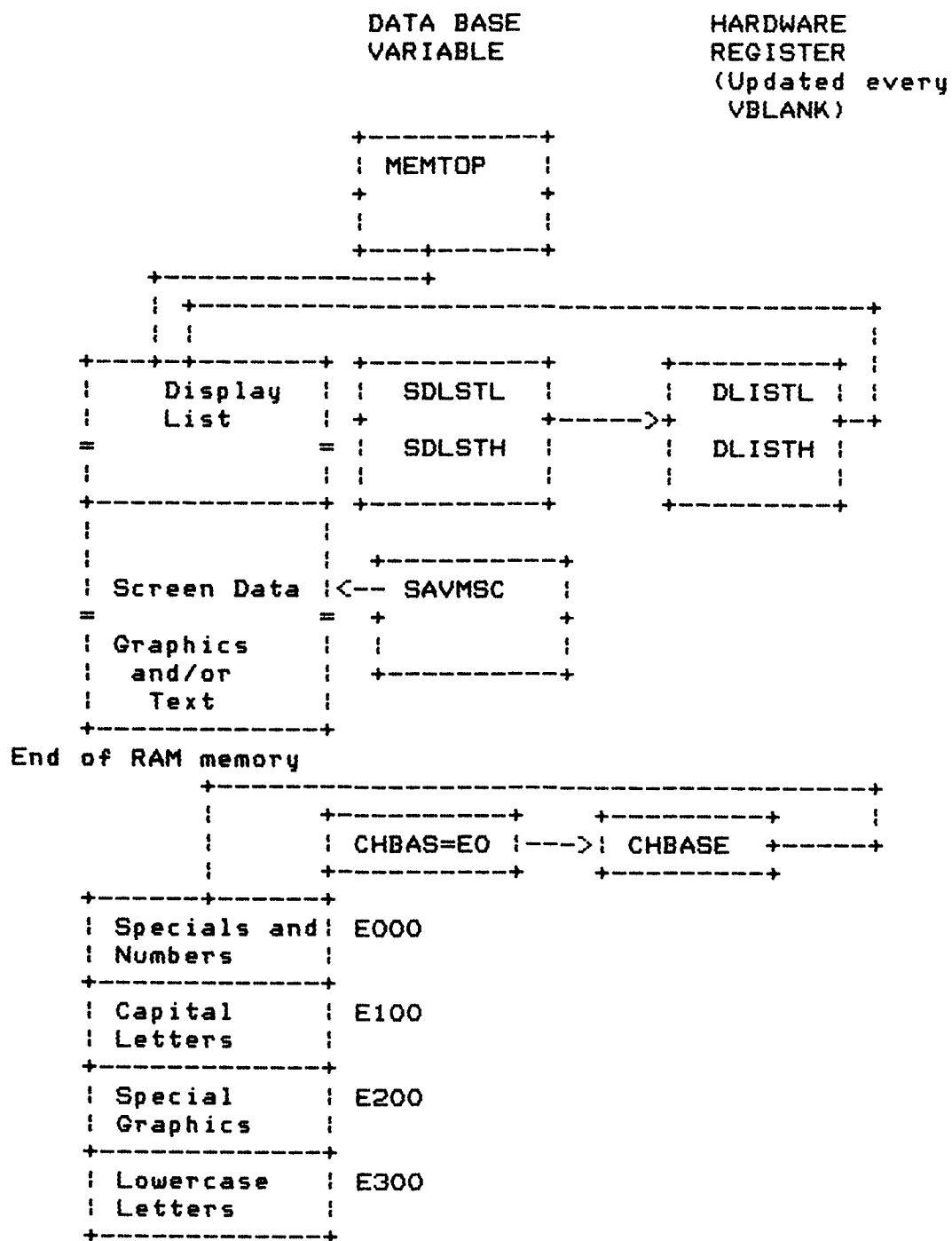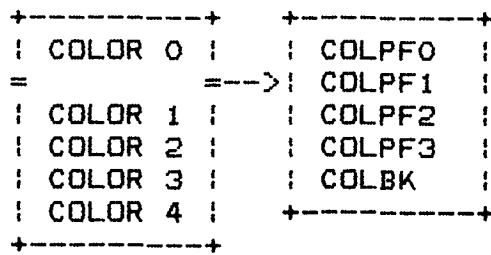
Figure 5-8 Screen Display Block Diagram

The following relationships are present in the preceding diagram:

1. Data base variables SDLSTL/SDLSTH contain the address of
    the current display list. This address is stored in the
    hardware display list address registers DLISTL and DLISTH
    as part of the VBLANK process.

2. The display list itself defines the characteristics of the
    screen to be displayed and points to the memory containing
    the data to be displayed.

3. Data base variable CHBAS contains the MSB of the base address
    of the character representations for the character data (text
    modes only).

    The default value for this variable is $E0. This variable
    declares that the character representations start at memory
    address E000 (the character set provided by the OS in ROM).
    Each character is defined as an 8X8 bit matrix, requiring 8
    bytes per character. 1024 bytes are required to define the
    largest set, since a character code contains up to 7
    significant bits (set of 128 characters). The OS ROM contains
    the default set in the region from E000 to E3FF.

    All character codes are converted by the handler from ATASCII
    to an internal code (and vice versa), as shown below:

        ATASCII   INTERNAL
         CODE   CODE

         00-1F   40-5F
         20-3F   00-1F
         40-5F   20-3F
         60-7F   60-7F
         80-9F   C0-DF
         A0-BF   80-9F
         C0-DF   A0-BF
         E0-FF   E0-FF

```

The character set in ROM is ordered by internal code order. Three considerations differentiate the internal code from the external (ATASCII) code:

ATASCII codes for all but the special graphics characters were to be similar to ASCII. The alphabetic, numeric, and punctuation character codes are identical to ASCII.

In text modes 1 and 2 it was desired that one character subset include capital letters, numbers, and punctuation and the other character subset include lowercase letters and special graphics characters.

The codes for the capital and lowercase letters were to be identical in text modes 1 and 2.

Database variables COLOR0 through COLOR4 contain the current color register assignments. Hardware color registers receive these values as part of the stage 1 VBLANK process, thus providing synchronized color changes (see Appendix H).

Database variable SAVMSC points to the lowest memory address of the screen data region. It corresponds to the data displayed at the upper left corner of the display.

When the Display Handler receives an open command, it first determines the screen mode from the OPEN IOCB. Then it allocates memory from the end of RAM downward (as specified by data base variable RAMTOP), first for the screen data and then for the display list. The screen data region is cleared and the display list is created if sufficient memory is available. The display list address is stored to the database.

Screen Editor (E:)

The Screen Editor is a read/write handler that uses the Keyboard
Handler and the Display Handler to provide "line-at-a-time" input
with interactive editing functions, as well as formatted output.


The Screen Editor supports the following CIO functions:

    OPEN
    CLOSE
    GET CHARACTERS
    GET RECORD
    PUT CHARACTERS
    PUT RECORD
    GET STATUS (null function)


See Keyboard Handler and Display Handler Sections for a
discussion of Screen Editor error statuses.

The Screen Editor is one of the resident handlers, and
therefore has a set of device vectors starting at location
E400.

The Screen Editor is a program that reads key data from the
Keyboard Handler and sends each character to the Display Handler
for immediate display. The Screen Editor also accepts data from
you to send to the Display Handler, and reads data from the
Display Handler (not the Keyboard Handler) for you. In fact,
the Keyboard Handler, Display Handler, and the Screen Editor are
all contained in one monolithic hunk of code.

Most of the behaviors already defined for the Keyboard Handler
and the Display Handler apply as well to the Screen Editor: The
discussions in this Section will be limited to deviations from
those behaviors, or to additional features that are part of the
Screen Editor only. The Screen Editor deals only with text data
(screen mode 0). This Section also explains the split-screen
configuration feature.

The Screen Editor uses the Display Handler to read data from
graphics and text screens on demand. You use the Screen
Editor to determine when the program will read Screen data, and
where upon the screen the data will be read from. You
first locates the cursor on the screen to determine the screen
area to be read; you then press the [RETURN] key to determine
when the program will begin to read the data indicated.

When the [RETURN] key is pressed, the entire logical line within
that the cursor resides is then made available to the calling
program: Trailing blanks in a logical line are never returned as
data, however. After all of the data in the line has been sent to
the caller (this can entail multiple READ CHARACTERS functions if
desired), an EOL character is returned and the cursor is
positioned to the beginning of the logical line following the one
just read.

## CIO Function Descriptions

The device-specific characteristics of the standard CIO
functions are detailed below:

### OPEN

The device name is E, and the Screen Editor ignores any
device number and filename specification, if included.

The Screen Editor supports the following option:

```
          7   O
         +-+-+-+-+-+-+-+-+
AUX1   ! | |W|R| |F|
         +-+-+-+-+-+-+-+-+
```

Where: R and W are the direction bits (read and write).
       F = 1 indicates that a "forced read" is desired (see GET
           CHARACTER and GET RECORD for more information).

### CLOSE

No special handler actions.

### GET CHARACTER and GET RECORD

Normally the Screen Editor will return data only when you press the
[RETURN] key at the keyboard. However, the "forced read" OPEN option
allows you to read text data without intervention. When you command a
READ operation, the Screen Editor will return data from the start of
the logical line in which the text cursor is located, and then
move the cursor to the beginning of the following logical line. A
read of the last logical line on the screen will cause the screen
data to scroll.

A special case occurs when characters are output without a
terminating EOL, and then additional characters are appended to

that logical line from the keyboard. When the [RETURN] key is
pressed, only the keyboard entered characters are sent to the
caller, unless the cursor has been moved out of and then back
into the logical line, in that case all of the logical line will
be sent.


PUT CHARACTER and PUT RECORD

The Handler accepts ATASCII characters as one character per byte.
Sixteen of the 256 ATASCII characters are control codes; the EOL
code has universal meaning, but most of the other control codes
have special meaning only to a display or print device. The
Screen Editor processing of the ATASCII control codes is
explained below:


CLEAR ($7D) -- The Screen Editor clears the current display of
all data and the cursor is placed at the home position (upper
left corner of the screen).


CURSOR UP ($1C) -- The cursor moves up by one physical line. The
cursor will wrap from the top line of the display to the bottom
line.


CURSOR DOWN ($1D) -- The cursor moves down by one physical line.
The cursor will wrap from the bottom line of the display to the
top line.


CURSOR LEFT ($1E) -- The cursor moves left by one column. The
cursor will wrap from the left margin of a line to the right
margin of the same line.


CURSOR RIGHT ($1F) -- The cursor moves right by one column. The
cursor will wrap from the right margin of a line to the left
margin of the same line.


BACKSPACE ($7E) -- The cursor moves left by one column (but never
past the beginning of a logical line), and the character at that
new position is changed to a blank ($20).

SET TAB ($9F) -- The Screen Editor establishes a tab point at the logical line position at that the cursor is residing. The logical line tab position is not synonymous with the physical line column position since the logical line can be up to 3 physical lines in length. For example, tabs can be set at the 15th, 30th, 45th, 60th and 75th character positions of a logical line as shown below:

```
0 2         9           19          29          39   Screen column #.
--L-------+-----------+-----------+---------R        L/R = margins.

x x---------------T---------------T--------           A logical line.
x x-------T---------------T---------------T-          x = inaccesible
x x--------------------------------------------        columns.
```

Note the effect of the left margin in defining the limits of the logical line.

The Handler default tab settings are shown below:

```
0 2         9           19          29          39   Screen column #.
--L-------+-----------+-----------+---------R        L/R = margins.

x xT-----T--------T--------T--------T-------T        A logical line.
x x-----T--------T--------T--------T-------T         x = inaccesible
x x-----T--------T--------T--------T-------T          columns.
```


CLEAR TAB ($9E) -- The Screen Editor clears the current cursor position within the logical line from being a tab point. There is no "clear all tab points" facility provided by the Handler.

TAB ($7F) -- The cursor moves to the next tab point in the current logical line, or to the beginning of the next line if no tab point is found. This function will not increase the logical line length to accommodate a tab point outside the current length (e.g. the logical line length is 38 characters and there is a tab point at position 50).

INSERT LINE ($9D) -- All physical lines at and below the physical line in that the cursor resides, are moved down by one physical line. The last logical line on the display can be truncated as a result. The blank physical line at the insert point becomes the beginning of a new logical line. A logical line can be split into two logical lines by this process, the last half of the original logical line being concatenated with the blank physical line formed at the insert point.

DELETE LINE ($9C) -- The logical line in that the cursor resides is deleted and all data below that line is moved upward to fill the void. Empty logical lines are created at the bottom of the display.

INSERT CHARACTER ($FF) -- All physical characters at and behind the cursor position on a logical line are moved one position to the right. The character at the cursor position is set to blank. The last character of the logical line will be lost when the logical line is full and a character is inserted. The number of physical lines comprising a logical line can increase as a result of this function.

DELETE CHARACTER ($FE) -- The character on which the cursor resides is removed, and the remainder of the logical line to the right of the deleted character is moved to the left by one position. The number of physical lines composing a logical line can decrease as a result of this function.

ESCAPE ($1B) -- The next non-EOL character following this code is displayed as data, even if it would normally be treated as a control code. The sequence [ESC][ESC] will cause the second [ESC] character to be displayed.

BELL ($FD) -- An audible tone is generated; the display is not modified.

END OF LINE ($9B) -- In addition to its record termination function, the EOL causes the cursor to advance to the beginning of the next logical line. When the cursor reaches the bottom line of the screen, the receipt of an EOL will cause the screen data to scroll upward by one logical line.

GET STATUS

The Handler takes no action other than to set the status to $01.

User-Alterable Data Base Variables

Also see the Display Handler data base variable discussion.

Cursor Position

When in a split-screen configuration, ROWCRS and COLCRS are associated
with the graphics portion of the display and two other variables,
TXTROW [0290] and TXTCOL [0291], are associated with the text window.
TXTROW is a single byte, and TXTCOL is 2-bytes with the least
significant byte being at the lower address. Note that the most
significant byte of TXTCOL should always be zero.

The home position (0,0) for the text window is the upper left corner
of the window.

Enable/Inhibit of Control Codes in Text

Normally all text mode control codes are operated upon as received,
but sometimes it is desirable to have the control codes displayed as
if they were data characters. This is done by setting the variable
DSPFLG [02FE] to any nonzero value before outputting the data
containing control codes. Setting DSPFLG to zero restores normal
processing of text control codes.

Cassette Handler (C:)

The Cassette device is a read or write device with a Handler
that supports the following CIO functions:

    OPEN
    CLOSE
    GET CHARACTERS
    GET RECORD
    PUT CHARACTERS
    PUT RECORD
    GET STATUS (null function)

The Cassette Handler can produce the following error statuses:

    $80 -- [BREAK] key abort.
    $84 -- Invalid AUX1 byte on OPEN.
    $88 -- end-of-file.
    $8A-90 -- SIO error set (see Appendix C).

The Cassette Handler is one of the resident handlers, and therefore
has a set of device vectors starting at location E440.

CIO Function Descriptions

The device-specific characteristics of the standard CIO functions are
detailed below:

    OPEN

The device name is C, and the Handler ignores any device number and
filename specification, if included.

The Handler supports the following option:

```
              7                    0
              +-+-+-+-+-+-+-+-+-+
     AUX2     |C|                 |
              +-+-+-+-+-+-+-+-+-+
```

Where: C = 1 indicates that the cassette is to be read/written without
       stop/start between records (continuous mode).

Opening the cassette for input generates a single audible tone, as a
prompt for you to verify that the cassette player is set up
for reading (power on; Serial Bus cable connected; tape cued to start
of file; and PLAY button depressed). When the cassette is ready,
you can press any keyboard key (except [BREAK]) to initiate tape
reading.

Opening the cassette for output generates two closely spaced audible
tones, as a prompt for you to verify that the cassette player
is set up for writing (as above, plus RECORD button depressed). When
the cassette is ready, you can press any keyboard key (except
[BREAK]) to begin tape writing. There is no way for the computer to
verify that the RECORD or PLAY button is depressed. It is possible for
the file not to be written, with no immediate indication of this fact.

There is a potential problem with the cassette in that when the
cassette is opened for writing, the motor keeps running until the
first record (128 data bytes) is written. If 128 data bytes are
written or the cassette is closed within about 30 seconds of the OPEN,
and no other serial bus I/O is performed, then there is no problem.
However, if those conditions are not met, some noise will be written
to the tape prior to the first record and an error will occur when
that tape file is read later. If lengthy delays are anticipated
between the time the cassette file is opened and the time that the
first cassette record (128 data bytes) is written, then a dummy record
should be written as part of the file; typically 128 bytes of some
innocuous data would be written, such as all zeros, all $FFs, or all
blanks ($20).

The system sometimes emits whistling noises after cassette I/O has
occurred. The sound can be eliminated by storing $03 to SKCTL [D20F],
thus bring POKEY out of the two-tone (FSK) mode.

   CLOSE

The CLOSE of a tape read stops the cassette motor.

The CLOSE of a tape write does the following:

     Writes any remaining user data in the buffer to tape.
     Writes an end-of-file record.
     Stops the cassette motor.

GET CHARACTERS and GET RECORD

The Handler returns data in the following format:

```
      7                   0
      +-+-+-+-+-+-+-+-+
      !    data byte    !
      +-+-+-+-+-+-+-+-+
```

PUT CHARACTERS and PUT RECORD

The Handler accepts data in the following format:

```
      7                   0
      +-+-+-+-+-+-+-+-+
      !    data byte    !
      +-+-+-+-+-+-+-+-+
```

The Handler attaches no significance to the data bytes written, a value of $9B (EOL) causes no special action.

GET STATUS

The Handler does no more than set the status to $01.

Theory of Operation

The Cassette Handler writes and reads all data in fixed-length records of the format shown below:

```
      +-+-+-+-+-+-+-+-+
      !0 1 0 1 0 1 0 1!        Speed measurement bytes.
      +-+-+-+-+-+-+-+-+
      !0 1 0 1 0 1 0 1!
      +-+-+-+-+-+-+-+-+
      !  control byte  !
      +-+-+-+-+-+-+-+-+
      !      128       !
      =      data      =
      !      bytes     !
      +-+-+-+-+-+-+-+-+
      !    checksum    !        (Managed by SIO, not the
      +-+-+-+-+-+-+-+-+          Handler.)
```

Figure 5-9   Cassette Handler Record Format

The control byte contains one of three values:

o $FC indicates the record is a full data record (128 bytes).

o $FA indicates the record is a partially full data record; you
       supplied fewer than 128 bytes to the record. This case can
       occur only in the record prior to the end-of-file. The number
       of user-supplied data bytes in the record is contained in the
       byte prior to the checksum.

o $FE indicates the record is an End-of file record; the data
       portion is all zeroes for an end-of-file record.

The SIO routine generates and checks the checksum. It is part of the
tape record, but it is not contained in the Handler's record buffer
CASBUF [03FD].

The processing of the speed-measurement bytes during cassette reading
is discussed in Appendix L, D1-D7.

File Structure

The Cassette Handler writes a file to the cassette device with a file
structure that is totally imposed by the Handler (soft format). A file
consists of the following three elements:

o   A 20-second leader of mark tone.

o   Any number of data-record frames.

o   An end-of-file frame.

The cassette-data record frames are formatted as shown below:

        frame = pre-record write tone (PRWT),
                + data record,
                + post record gap (PRG)

The nondata portions of a frame have characteristics that are
dependent upon the write OPEN mode, i.e. continuous or
start/stop.

        Stop/start PRWT = 3 seconds of mark tone.
        Continuous PRWT = .25 second of mark tone.

        Stop/start PRG = up to 1 second of unknown tones.
        Continuous PRG = from 0 to n seconds of unknown tones, where
                         n is dependent upon your program timing.

The inter-record gap (IRG) between any two records consists of
the PRG of the first record followed by the PRWT of the second
record.

Printer Handler (P:)

The Printer device is a write-only device with a Handler that
supports the following CIO functions:

    OPEN
    CLOSE
    PUT CHARACTERS
    PUT RECORD
    GET STATUS

The Printer Handler can produce the following error statuses:

    $8A-90 -- SIO error set (see Appendix C).

The Printer Handler is one of the resident handlers, and
therefore has a set of device vectors starting at location E430.

CIO Function Descriptions

The device-specific characteristics of the standard CIO functions
are detailed below:

  OPEN

The device name is P. The Handler ignores any device number and
filename specification, if included.

  CLOSE

The Handler writes any data remaining in its buffer to the
printer device, with trailing blanks to fill out the line.

  PUT CHARACTERS and PUT RECORD

The Handler accepts print data in the following format:

```
        7                 0
        +-+-+-+-+-+-+-+-+
        !     ATASCII    !
        +-+-+-+-+-+-+-+-+
```

The only ATASCII control code of any significance to the Handler
is the EOL character. The printer device ignores bit 7 of every
data byte and prints a sub set of the remaining 128 codes. (see
Appendix G for the printer character set).

The Handler supports the following print option:

```
                  7                0
                  +-+-+-+-+-+-+-+-+
        AUX2      !  print mode   !
                  +-+-+-+-+-+-+-+-+
```

Where: $4E (N) selects normal printing (40 characters per line).
       $53 (S) selects sideways printing (29 characters per line).
       $57 (W) selects wide printing (not supported by printer
               device).

       Any other value (including 00) is treated as a normal (N)
       print select, without producing an error status.

   GET STATUS

The Handler obtains a 4-byte status from the printer
controller and puts it in system location DVSTAT [02EA]. The
format of the status bytes is shown below:

```
              +-+-+-+-+-+-+-+-+
              ! command stat. !      DVSTAT + 0
              +-+-+-+-+-+-+-+-+
              ! AUX2 of prev. !           + 1
              +-+-+-+-+-+-+-+-+
              !    timeout    !           + 2
              +-+-+-+-+-+-+-+-+
              !    (unused)   !           + 3
              +-+-+-+-+-+-+-+-+
```

The command status contains the following status bits and
condition indications:

    bit 0:   an invalid command frame was received.
    bit 1:   an invalid data frame was received.
    bit 7:   an intelligent controller (normally = 0).

The next byte contains the AUX2 value from the previous operation.

The timeout byte contains a controller provided maximum timeout
value (in seconds).

Theory of Operation

The ATARI 820[TM] 40-Column Printer is a line-at-a-time printer rather
than a character-at-a-time printer, so your data must be buffered by
the Handler and sent to the device in records corresponding to one
print line (40 characters for normal, 29 characters for sideways).

```

The printer device does not attach any significance to the EOL
character, so the Handler does the appropriate blank fill
whenever it sees an EOL.


### Disk File Manager (D:)

The OS supports four unique File Management Subsystems at the
time of this writing. Version IA is the original version.
Version IB is a slightly modified version of IA and is the one
described in this document. Most of this discussion applies as
well to Version II, that handles a double-density diskette (720
256-byte sectors) in addition to the single-density diskette (720
128-byte sectors). Version III has all new file/directory/map
structures and can possibly contain changes to your interface
as well.

The File Management Subsystem includes a disk-bootable
(RAM-resident) Disk File Manager (DFM) that maintains a
collection of named files on diskettes. Up to 4 disk drives
(D1: through D4:) can be accessed, and up to 64 files per
diskette can be accessed. The system diskettes supplied by ATARI
allow a single disk drive (D1) and up to 3 OPEN files, but
you can alter these numbers as described later in
this section.


The Disk File Manager supports the following CIO functions:

    OPEN FILE
    OPEN DIRECTORY
    CLOSE
    GET CHARACTERS
    GET RECORD
    PUT CHARACTERS
    PUT RECORD
    GET STATUS


    NOTE
    POINT
    LOCK
    UNLOCK
    DELETE
    RENAME
    FORMAT

The Disk File Manager can produce the following error statuses:

    $03 -- Last data from file (EOF on next read).
    $88 -- end-of-file.
    $8A-90 -- SIO error set (see Appendix C).
    $A0 -- Drive number specification error.
    $A1 -- No sector buffer available (too many open files).
    $A2 -- Disk full.
    $A3 -- Fatal I/O error in directory or bitmap.
    $A4 -- Internal file # mismatch (structural problem).
    $A5 -- File name specification error.
    $A6 -- Point information in error.
    $A7 -- File locked to this operation.
    $A8 -- Special command invalid.
    $A9 -- Directory full (64 files).
    $AA -- File not found.
    $AB -- Point invalid (file not OPENed for update).

CIO Function Descriptions

The device-specific characteristics of the standard CIO functions
are detailed below:

   OPEN FILE

The device name is D. Up to four disk drives can be accessed (D1
through D4). The disk filename can be from 1 to 8 characters in
length with an optional 1- to 3-character extension.

The OPEN FILE command supports the following options:


                 7               0
               +-+-+-+-+-+-+-+-+
       AUX1    |         |W|R| |A|
               +-+-+-+-+-+-+-+-+

Where: W and R are the direction bits.
       WR = 00 is invalid
            01 indicates OPEN for read only.
            10 indicates OPEN for write only.
            11 indicates OPEN for read/write (update).


       A = 1 indicates appended output when W = 1.

You may use these following valid AUX1 options:

OPEN Input (AUX1 = $04)

The indicated file is opened for input. Any wild-card characters are used to search for the first match. If the file is not found, an error status is returned, and no file will be opened.


OPEN Output (AUX1 = $08)

The indicated file is opened for output starting with the first byte of the file, if the file is not locked. Any wild-card characters are used to search for the first match. If the file already exists, the existing file will be deleted before opening the named file as a new file. If the file does not already exist, it will be created.

A file opened for output will not appear in the directory until it has been closed. If an output file is not properly closed, some or all of the sectors that were acquired for it can be lost until the disk is reformatted.

A file that is opened for output can not be opened concurrently for any other access.


OPEN Append (AUX1 = $09)

The indicated file is opened for output starting with the byte after the last byte of the existing file (that must already exist), if the file is not locked. Any wild-card characters are used to search for the first match.

If a file opened for append is not properly closed, the appended data will be lost. The existing file will remain unmodified and some or all of the sectors that were acquired for the appended portion can be lost until the diskette is reformatted.


OPEN Update (AUX1 = $0C)

The indicated file (that must already exist) will be opened for update provided it is not locked. Any wild-card characters are used to search for the first match.

The GET, PUT, NOTE and POINT operations are all valid, and can be intermixed as desired.

If a file opened for update is not properly closed, a sector's worth of information can be lost to the file. A file opened for update can not be extended.

Device/Filename Specification

The Handler expects to find a device/filename specification of the following form:


    D[<number>]:<filename><EOL>

where:

    <number> ::= 1|2|3|4
    <filename> ::= [<primary>][.[<extension>]]<terminator>
    <primary> ::= an uppercase alpha character followed by 0 to 7
                  alphanumeric characters. If the primary name is
                  less than 8 characters, it will be padded with
                  blanks; if it is greater than 8 characters, the
                  extra characters will be ignored.

    <extension> ::= Zero to 3 alphanumeric characters. If the
                  extension name is missing or less than 3
                  characters, it will be padded with blanks; if
                  it is greater than 3 characters, the extra
                  characters will be ignored.

    <terminator> ::= <EOL>|<blank>


Figure 5-10    Device/Filename Syntax



The following are all valid device/filenames for the diskette:

    D1:GAME.SRC
    D:MANUAL6
    D:.WHY
    D3:FILE.
    D4:BRIDGE.002




Filename Wildcarding

The filename specification can be further generalized to include
the use of the "wild-card" characters * and ?. These wildcard
characters allow portions of the primary and/or extension to be
abbreviated as follows:

The ? character in the specification allows any filename
character at that position to produce a "match." For example, WH?
will match files named WHO, WHY, WH4, etc., but not a file named
WHAT.

The * character causes the remainder of the primary or extension field in that it is used to be effectively padded with ? characters. For example, WH* will match WHO, WHEN, WHATEVER, etc.

Some valid uses of wild-card specifications are shown below:

    *.SRC              Files having an extension of SRC.
    BASIC.*            Files named BASIC with any extension.
    *.*                All files.
    H*.?               Files beginning with H and having a 0 or 1
                       character extension.

If wildcarding is used with an OPEN FILE command, the first file found (if any) that meets the specification will be the one (and only one) opened.


   OPEN DIRECTORY

The OPEN DIRECTORY command allows you read directory information for the selected filename(s), using normal GET CHARACTERS or GET RECORD commands. The information read will be formatted as ATASCII records, suitable for printing, as shown below. Wildcarding can be used to obtain information for multiple files or the entire diskette.

The OPEN DIRECTORY command uses the same CIO parameters as a standard OPEN FILE command:

    COMMAND BYTE = $03

    BUFFER ADDRESS = pointer to device/filename specification.

    AUX1 = $06

After the directory is opened, a record will be returned to the caller for each file that matches the OPEN specification. The record, that contains only ATASCII characters, is formatted as shown below:

```
                    1
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |s|b| primary name | ext |b|count|e|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where: s = * or ' ', with * indicating file locked.
       b = blank.
       primary name = left-justified name with blank fill.
       ext = left-justified extension with blank fill.
       b = blank.
       count = number of sectors comprising the file.
       e = EOL ($9B).

After the last filename match record is returned, an additional
record is returned. This record indicates the number of unused
sectors available on the diskette. The format for this record is shown
below:

```
                            1
      1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      !count!  F R E E   S E C T O R S!e!
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where: count = the number of unused sectors on the diskette.
       e = EOL ($9B).

The EOF statuses ($03 and $88) are returned as in a normal data
file when the last directory record is read.

The opening of another diskette file while the directory read is
open will cause subsequent directory reads to malfunction, so
care must be taken to avoid this situation.

   CLOSE

Upon closing a file read, the Handler releases all internal
resources being used to support that file.

Upon closing a file write, the Handler:

o   writes any residual data from its file buffer for that file
       to the diskette.

o   updates the directory and allocation map for the associated
       diskette.

o   releases all internal resources being utilized to support
       that file

   GET CHARACTERS and GET RECORD

Characters are read from the diskette and passed to CIO as a raw
data stream. None of the ATASCII control characters have any
special significance. A status of $88 is returned if an attempt
is made to read past the last byte of a file.

PUT CHARACTERS and PUT RECORD

Characters are obtained from CIO and written to the diskette as a raw
data stream. None of the ATASCII control characters have any special
significance.

GET STATUS

The indicated file is checked and one of the following status
byte values is returned in ICSTA and register Y:

    $01 -- File found and unlocked.
    $A7 -- File locked.
    $AA -- File not found.

Special CIO Functions

The DFM supports a number of SPECIAL commands, that are device
specific. These are explained in the paragraphs that follow:

NOTE (COMMAND BYTE = $25)

This command returns to the caller the exact diskette location of
the next byte to be read or written, in the variables shown
below:

    ICAX3 = LSB of the diskette sector number.
    ICAX4 = MSB of the diskette sector number.
    ICAX5 = relative sector displacement to byte (0-124).

POINT (COMMAND BYTE = $26)

This command allows you to specify the exact diskette location of
the next byte to be read or written. In order to use this commmand,
the file must have been opened with the "update" option.

    ICAX3 = LSB of the diskette sector number.
    ICAX4 = MSB of the diskette sector number.
    ICAX5 = relative sector displacement to byte (0-124).

LOCK

This command allows you to prevent write access to any
number of named files. Locked files can not be deleted, renamed,
nor opened for output unless they are first unlocked. Locking a
file that is already locked is a valid operation. The Handler
expects a device/filename specification; then all occurrences of
the filename specified will be locked, using the wild-card rules.

You set up these following IOCB parameters prior to
calling CIO:

    COMMAND BYTE = $23

    BUFFER ADDRESS = pointer to device/filename specification.

After a LOCK operation, the following IOCB parameter will have
been altered:

    STATUS = result of LOCK operation; see Appendix B for a list
             of possible status codes.

   UNLOCK

This command allows you to remove the lock status of any
number of named files. Unlocking a file that is not locked is a
valid operation. The Handler expects a device/filename
specification; then all occurrences of the filename specified
will be unlocked, using the wild-card rules.

You set up these following IOCB parameters prior to
calling CIO:

    COMMAND BYTE = $24

    BUFFER ADDRESS = pointer to device/filename specification.

After an UNLOCK operation, the following IOCB parameter will have been
altered:

    STATUS = result of UNLOCK operation; see Appendix B for a
             list of possible status codes.

   DELETE

This command allows you to delete any number of unlocked
named files from the directory of the selected diskette and to
deallocate the diskette space used by the files involved. The
Handler expects a device/filename specification; then all
occurences of the filename specified will be deleted, using the
wild-card rules.

You set up these following IOCB parameters prior to
calling CIO:

    COMMAND BYTE = $21

    BUFFER ADDRESS = pointer to device/filename specification.

After a DELETE operation, the following IOCB parameter will have
been altered:

    STATUS = result of DELETE operation; see Appendix B for a list of
    possible status codes.

RENAME

This command allows you to change the filenames of any
number of unlocked files on a single diskette. The Handler
expects to find a device/filename specification that follows:

    <device spec>:<filename spec>,<filename spec><EOL>

All occurrences of the first filename will be replaced with the
second filename, using the wild-card rules. No protection is
provided against forming duplicate names. Once formed, duplicate
names cannot be separately renamed or deleted; however, an OPEN
FILE command will always select the first file found that matches
the filename specification, so that file will always be
accessible. The RENAME command does not alter the content of the
files involved, merely the name in the directory.

Examples of some valid RENAME name specifications are shown
below:

    D1:*.SRC,*.TXT
    D:TEMP,FDATA
    D2:F*,F*.OLD

You set up these following IOCB parameters prior to
calling CIO:

    COMMAND BYTE = $20

    BUFFER ADDRESS = pointer to device/filename specification.

After a RENAME operation, the following IOCB parameter will have
been altered:

    STATUS = result of RENAME operation; see Appendix B for a
    list of possible status codes.

FORMAT

Soft-sector diskettes must be formatted before they can store
data. The FORMAT command allows you to physically format a
diskette. The physical formatting process writes a new copy of
every sector on the soft-sectored diskette, with the data portion
of each sector containing all zeros. The FORMAT process creates
an "empty" non system diskette. When the formatting process is
complete, the FMS creates an initial Volume Table of Contents
(VTOC) and an initial File Directory. The boot sector (#1) is
permanently reserved as part of this process.

You set up these following IOCB parameters prior to
calling CIO:

    COMMAND BYTE = $FE

    BUFFER ADDRESS = pointer to device specification.

After a FORMAT operation, the following IOCB parameter will have
been altered:

    STATUS = result of FORMAT operation; see Appendix B for a
    list of possible status codes.

To create a system diskette, a copy of the boot file must then be
written to sectors #2-n. This is accomplished by writing the file
named DOS.SYS. This is a name that is recognized by the FMS even
though it is not in the directory initially.

Theory of Operation

The resident OS initiates the disk-boot process (see Section 10).
The OS reads diskette sector #1 to memory and then transfers
control to the "boot continuation address" (boot address + 6).
The boot-continuation program contained in sector #1 then
continues to load the remainder of the File Management Subsystem
to memory using additional information contained in sector #1.
The File Management Subsystem loaded, will contain a Disk File
Manager ,and optionally, a Disk Utilities (DOS) package.

When the boot process is complete, the Disk File Manager will
allocate additional RAM for the creation of sector buffers.
Sector buffers are allocated based upon information in the boot
record as shown below:

    Byte 9 = maximum number of open files; one buffer per (the
             maximum value is 8).

    Byte 10 = drive select bits; one buffer per (1-4 only).

The Disk File Manager will then insert the name D and the Handler
vector table address in the device table.

NOTE: There is a discrepancy between the Disk File Manager's
numbering of diskette sectors (0-719) and the disk controller's
numbering of diskette sectors (1-720); as a result, only sectors
1- 719 are used by the Disk File Manager.

The Disk File Manager uses the Disk Handler to perform all
diskette reads and writes; the DFM's function is to support and
maintain the directory/file/bitmap structures as described in the
following pages:

FMS Diskette Utilization

The map below shows the diskette sector utilization for a
standard 720 sector diskette.

```
+----------------+
|   BOOT record  |        Sector 1
+----------------+
|    FMS BOOT    |        Sector 2     -+
=     file       =                      |
|    DOS.SYS     |        Sector n      +- Note 1
+----------------+                      |
|      User      |        Sector n+1  -+
=      File      =
|      Area      |        Sector 359 ($167)
+----------------+
|  VTOC(note 2)  |        Sector 360 ($168)
+----------------+
|      File      |        Sector 361 ($169)
=   Directory    =
|                |        Sector 368 ($170)
+----------------+
|      User      |
=      File      =
|      Area      |        Sector 719 ($2CF)
+----------------+
|     unused     |        Sector 720 ($2D0)
+----------------+
```

Figure 5-11   File Management Subsystem Diskette Sector Utilization
Map

NOTE 1 - If the diskette is not a system diskette, then your
File Area starts at sector 2 and no space is reserved for the FMS
BOOT file. However, "DOS" (DOS.SYS and DUP.SYS) may still be
written to a diskette that has already used sectors "2-N."

NOTE 2 -- VTOC stands for Volume Table of Contents.

FMS Boot Record Format

The FMS BOOT record (sector #1) is a special case of diskette-booted software (see Section 10). The format for the FMS BOOT record is shown below:

```
          +----------------+
          !  boot flag = 0 !          Byte 0
          +----------------+
          ! # sectors = 1  !               1
          +----------------+
          !  boot address  !               2
          +                +
          !     = 0700     !
          +----------------+
          !  init address  !               4
          +                +
          !                !
          +----------------+
          !   JMP = $4B    !               6
          +                +
          !   boot read    !
          +  continuation  +
          !    address     !
          +----------------+ ---+
          ! max files = 3  !    !      9   Note 1
          +----------------+    !
          ! drive bits = 1 !    !     10   Note 2
          +----------------+    !
          ! alloc dirc = 0 !    !     11   Note 3
          +----------------+    !
          ! boot image end !    !
          +                +    !                   FMS
          !  address + 1   !    +-------------- configuration
          +----------------+    !                   data
          ! boot flag <> 0 !    !     14   Note 4
          +----------------+    !
          !  sector count  !    !     15   Note 5
          +----------------+    !
          !    DOS.SYS      !    !
          +    starting    +    !
          ! sector number  !    !
          +----------------+ ---+
          ! code for second!
          ! phase of boot  !
          +----------------+
```

Figure 5-12 File Management Subsystem Boot Record Format

NOTE 1 - Byte 9 specifies the maximum number of concurrently open
         files to be supported. This value can range from 1 to 8.

NOTE 2 - Byte 10 specifies the specific disk drive numbers to be
         supported using a bit encoding scheme as shown below:

```
   7 6 5 4 3 2 1 0
   +-+-+-+-+-+-+-+-+
   !         !4!3!2!1!  where a 1 indicates a selected drive.
   +-+-+-+-+-+-+-+-+
```

NOTE 3 - Byte 11 specifies the buffer allocation direction, this
         byte should equal 0.

NOTE 4 - Byte 14 must be nonzero for the second phase of the boot
         process to initiate. This flag indicates that the file
         DOS.SYS has been written to the diskette.

NOTE 5 - This byte is assigned as being the sector count for the
         DOS.SYS file. It is actually an unused byte.

Boot Process Memory Map

The diagram below shows how the boot sector (part of file
DOS.SYS) and following sectors are loaded to memory as part of
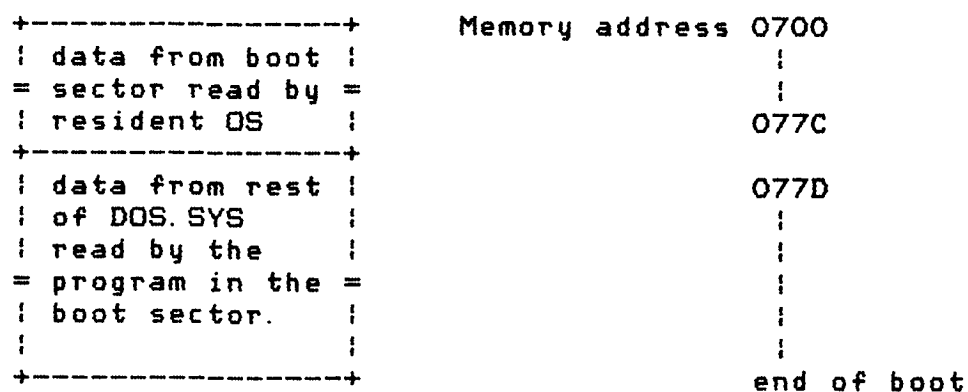the boot process.

```
         +------------------+      Memory address 0700
         ! data from boot !                        !
         = sector read by =                        !
         ! resident OS    !                     077C
         +------------------+
         ! data from rest !                     077D
         ! of DOS.SYS     !                        !
         ! read by the    !                        !
         = program in the =                        !
         ! boot sector.   !                        !
         !                !                        !
         +------------------+                   end of boot
```

Figure 5-13   File Management Subsystem Boot Process Memory Map

OPERATING SYSTEM CO16555 -- Section 5

Volume Table of Contents

The format for the FMS volume table of contents (VTOC, sector 360) is shown in the diagram below:

```
        +-----------------+
        ! directory type !      Byte 0   Note 1
        +-----------------+
        ! maximum    (lo) !         1   Note 2
        + sector #        +
        ! = 02C5     (hi) !
        +-----------------+
        ! number of  (lo) !         3   Note 3
        + sectors         +
        ! available  (hi) !
        +-----------------+
        !                 !
        =                 =
        !                 !
        +-----------------+
        !                 !          10
        = volume bit map  =
        !                 !
        +-----------------+
        !                 !
        =                 =
        !                 !
        +-----------------+
```

Figure 5-14   File Management Subsystem Volume Table of Contents

The volume bit map organization location follows:

```
        7                0
        +-+-+-+-+-+-+-+-+
        ! !1 2 3 4 5 6 7!     Byte 10 of VTOC
        +-+-+-+-+-+-+-+-+
        !8 9 . . . . . .!         11
        =               =
        !               !         99
        +-+-+-+-+-+-+-+-+
```

Figure 5-15   File Management Subsystem Volume Bit Map

At each map bit position, a 0 indicates the corresponding sector is in use and a 1 indicates that the sector is available.

NOTE 1 - The directory type byte must equal 0.

NOTE 2 - The maximum sector number is not used because it is incorrectly set to 709 decimal. The true maximum sector number is actually 719 for the DFM.

NOTE 3 - The number of sectors available is initially set to 709
        after a diskette is freshly formatted; this number is
        adjusted as files are created and deleted to show the
        number of sectors available. The sectors that are
        initially reserved are 1 and 360-368.

    File Directory Format

The FMS reserves eight sectors (361-368) for a file directory.
Each sector containing directory information for up to eight
files, thus providing for a maximum of 64 files for any volume.
The format of a single 16-byte file entry is shown below:

```
        +----------------+
        !   flag byte    !      Byte 0
        +----------------+
        ! sector    (lo) !        1
        + count          +
        !           (hi) !
        +----------------+
        ! starting  (lo) !        3
        + sector         +
        ! number    (hi) !
        +----------------+
        !           (1)  !        5
        +                +
        !           (2)  !
        +                +
        !           (3)  !
        +                +
        !    file   (4)  !
        +                +
        !    name   (5)  !
        +                +
        ! primary   (6)  !
        +                +
        !           (7)  !
        +                +
        !           (8)  !
        +----------------+
        !    file   (1)  !        13
        +                +
        !    name   (2)  !
        +                +
        ! extension (3)  !
        +----------------+
```

Figure 5-16   File Directory Format

Where the flag byte has the following bits assigned:

OPERATING SYSTEM C016555 -- Section 5

```
bit 7 = 1 if the file has been deleted.
bit 6 = 1 if the file is in use.
bit 5 = 1 if the file is locked.
bit 0 = 1 if OPEN output.
```

The flag byte can take on the following values:

```
$00 = entry not yet used (no file).
$40 = entry in use (normal CLOSEd file).
$41 = entry in use (OPEN output file).
$60 = entry in use (locked file).
$80 = entry available (prior file deleted).
```

Sector count is the number of sectors comprising the file.

FMS File Sector Format

The format of a sector in your data file is shown below:

```
     7               0
     +-+-+-+-+-+-+-+-+
     !     data      !   +0
     =               =
     !               !
     +-+-+-+-+-+-+-+-+
     ! file #    !hi !   +125
     +-+-+-+-+-+-+   +
     !forward pointer!   +126
     +-+-+-+-+-+-+-+-+
     !S! byte count  !   +127
     +-+-+-+-+-+-+-+-+
```
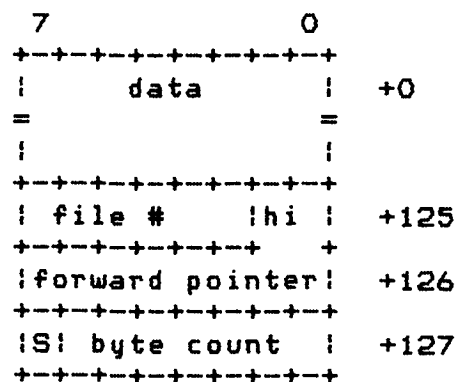
Figure 5-17   File Management Subsystem File Sector Format

The FMS uses the file # to verify file integrity. The file #
is a redundant piece of information. The file number field
contains the value of the directory position of that file. If a
mismatch occurs between the file's directory position, and the
file number as contained in each sector, then the DFM will
generate the error $A4.

The forward pointer field contains the 10-bit value for the
diskette sector number of the next sector of the file. The
pointer equals zero for the last sector of a file.

The S bit indicates whether or not the sector is a "short sector"
(a sector containing fewer than 125 data bytes). S is equal to
1 when the sector is short.

The byte-count field contains the number of data bytes in the sector.

Non-CIO I/O

Some portions of the I/O subsystem are accessed independently of the Central I/O Utility (CIO); this section discusses those areas.

Resident Device Handler Vectors

All of the OS ROM resident device handlers can be accessed via sets of vectors that are part of the OS ROM. These vectors increase the speed of I/O operations that utilize fixed device assignments, such as output to the Display Handler. For each resident Handler there is a set of vectors ordered as shown below:

```
+----------------+
+-    OPEN     -+        +0
+----------------+
+-    CLOSE    -+        +2
+----------------+
+-  GET BYTE   -+        +4
+----------------+
+-  PUT BYTE   -+        +6
+----------------+
+- GET STATUS  -+        +8
+----------------+
+-   SPECIAL   -+        +10
+----------------+
+-    JMP      -+        +12
+-    INIT     -+
+----------------+
+-    SPARE    -+
+-    BYTE     -+
+----------------+
```

Figure 5-18   Resident Device Handler Vectors

See Section 9 for a detailed description of the data interface for each of these Handler entry points.

Each of the vectors contains the address (lo,hi) of the Handler entry point minus 1. A technique similar to the one shown below is required to access the desired routines:

```
        VTBASE=$E400                          ; BASE OF VECTOR TABLE.

            .
            LDX     #xx                      ; OFFSET TO DESIRED ROUTINE.
            LDA     data
            JSR     GOVEC                    ; SEND DATA TO ROUTINE.

            .
            LDX     #yy                      ; OFFSET TO DIFFERENT ROUTINE.
            JSR     GOVEC                    ; GET DATA FROM ROUTINE.
            STA     data
            .

    GOVEC   TAY                              ; SAVE REGISTER A.
            LDA     VTBASE+1,X               ; ADDRESS MSB TO STACK.
            PHA
            LDA     VTBASE,X                 ; ADDRESS LSB TO STACK.
            PHA
            TYA                              ; RESTORE REGISTER A.
            RTS                              ; JUMP TO ROUTINE.
```

The JMP INIT slot in each set of vectors jumps to the Handler
initialization entry (not minus 1).

The base address of the vector set for each of the resident
handlers is shown below:

```
        Screen Editor (E:)        E400.
        Display Handler (S:)      E410.
        Keyboard Handler (K:)     E420.
        Printer Handler (P:)      E430.
        Cassette Handler (C:)     E440.
```

The resident diskette Handler is not CIO-compatible, so its
interface does not use a vector set.

    Resident Diskette Handler

The resident Diskette Handler (not to be confused with the Disk
File Manager) is responsible for all physical accesses to the
diskette. The unit of data transfer for this Handler is a single
diskette sector containing 128 data bytes.

Communication between you and the Diskette Handler is
effected using the system's Device Control Block (DCB), that is
also used for Handler/SIO communication (see Section 9). The DCB
is 12 bytes long. Some bytes are user-alterable and some are for
use by the Diskette Handler and/or the Serial I/O Utility (SIO).
You supply the required DCB parameters and then do a JSR
DSKINV [E453].

Each of the DCB bytes will now be described, and the
system-equate file name for each will be given.

SERIAL BUS ID -- DDEVIC [0300]

The Diskette Handler sets up this byte to contain the Serial Bus ID
for the drive to be accessed. It is not user-alterable.

DEVICE NUMBER -- DUNIT [0301]

You set up this byte to contain the disk drive number to be
accessed (1 - 4).

COMMAND BYTE -- DCOMND [0302]

You set up this byte to contain the disk device command to
be performed.

STATUS BYTE -- DSTATS [0303]

This byte contains the status of the command upon return to the
caller. See Appendix C for a list of the possible status codes.

BUFFER ADDRESS -- DBUFLO [0304] and DBUFHI [0305]

This 2-byte pointer contains the address of the source or
destination of the diskette sector data. You need not supply
an address for the disk status command. The Disk Handler will
obtain the status and insert the address of the status buffer
into this field.

DISK TIMEOUT VALUE -- DTIMLO [0306]

The Handler supplies this timeout value (in whole seconds) for
use by SIO.

BYTE COUNT -- DBYTLO [0308] and DBYTHI [0309]

This 2-byte counter indicates the number of bytes transferred to
or from the disk as a result of the most recent command, and is
set up by the Handler.

SECTOR NUMBER -- DAUX1 [030A] and DAUX2 [030B]

This 2-byte number specifies the diskette sector number (1 - 720)
to read or write. DAUX1 contains the least significant byte, and

DAUX2 contains the most significant byte.

Diskette Handler Commands

There are five commands supported by the Diskette Handler:

    GET SECTOR (PUT SECTOR --*** not supported by current handler ***)
    PUT SECTOR WITH VERIFY
    STATUS REQUEST
    FORMAT DISK

GET SECTOR (Command byte = $52)

The Handler reads the specified sector to your buffer and returns the
operation status. You set the following DCB parameters prior to
calling the Diskette Handler:

    COMMAND BYTE = $52.

    DEVICE NUMBER = disk drive number (1-4).

    BUFFER ADDRESS = pointer to your 128-byte buffer.

    SECTOR NUMBER = sector number to read.

Upon return from the sector, several of the other DCB parameters
will have been altered. The STATUS BYTE will be the only
parameter of interest to you, however.

PUT SECTOR (Command byte = $50)

*** Not supported by current Handler ***
    (But can be accessed through SIO directly.)

The Handler writes the specified sector from your buffer and returns
the operation status. You set the following DCB parameters prior to
calling the Diskette Handler:

    COMMAND BYTE = $50.

    DEVICE NUMBER = disk drive number (1-4).

    BUFFER ADDRESS = pointer to your 128 byte buffer.

    SECTOR NUMBER = sector number to write.

Upon return from the operation, several of the other DCB parameters
will have been altered. The STATUS BYTE will be the only one of
interest you, however.

PUT SECTOR WITH VERIFY (Command Byte = $57)

The Handler writes the specified sector from your buffer
and returns the operation status. This command differs from PUT
SECTOR in that the diskette controller reads the sector data after
writing to verify the write operation. Aside from the COMMAND
BYTE value, the calling sequence is identical to PUT SECTOR.

STATUS REQUEST (Command byte = $53)

The Handler obtains a 4-byte status from the diskette controller and
puts it in system location DVSTAT [02EA]. The operation status
format is shown below:

```
        7               0
        +-+-+-+-+-+-+-+-+
        : command stat. :     DVSTAT + 0
        +-+-+-+-+-+-+-+-+
        : hardware stat.:          + 1
        +-+-+-+-+-+-+-+-+
        :    timeout    :          + 2
        +-+-+-+-+-+-+-+-+
        :   (unused)    :          + 3
        +-+-+-+-+-+-+-+-+
```
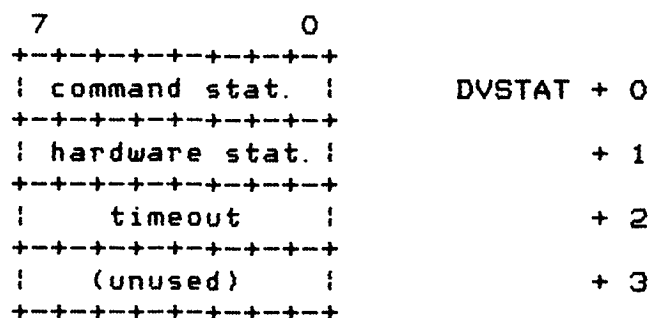
Figure 5-19.   DVSTAT 40-Byte Operation Status Format

The command status contains the following status bits:

    Bit 0 = 1 indicates an invalid command frame was received.
    Bit 1 = 1 indicates an invalid data frame was received.
    Bit 2 = 1 indicates that a PUT operation was unsuccessful.
    Bit 3 = 1 indicates that the diskette is write protected.
    Bit 4 = 1 indicates active/standby.

The hardware status byte contains the status register of the
INS1771-1 Floppy Diskette Controller chip used in the diskette
controller. See the documentation for that chip to obtain
information relating to the meaning of each bit in the byte.

The timeout byte contains a controller-provided maximum timeout
value (in seconds) to be used by the Handler.

You set the following DCB parameters prior to calling
the Diskette Handler:

    COMMAND BYTE = $53.

    DEVICE NUMBER = disk drive number (1-4).

Upon return from the operation, several of the other DCB parameters
will have been altered. The STATUS BYTE will be the only one of

interest to you, however.

FORMAT DISK (Command Byte = $21)

The Handler commands the diskette controller to format the entire
diskette and then to verify it. All bad sector numbers (up to a
maximum of 63) are returned and put in the supplied buffer,
followed by two bytes of all 1's ($FFFF). You set up the
following DCB parameters prior to calling the Diskette Handler:

    COMMAND BYTE = $21.

    DEVICE NUMBER = disk drive number (1-4).

    BUFFER ADDRESS = pointer to your 128-byte buffer.

Upon return, you might be interested in the following DCB parameters:

    STATUS BYTE = status of operation.

    BYTE COUNT = number of bytes of bad sector information in
    your buffer, not including the $FFFF terminator. If there
    are no bad sectors, the count will equal zero.

Serial Bus I/O

Input/Output to devices other than the keyboard, the screen, and
the ATARI Computer controller port devices, must utilize the
Serial I/O bus. This bus contains data, control, and clock lines
to be used to allow the computer to communicate with external
devices on this "daisychained" bus. Every device on the bus has
a unique identifier and will respond only when directly
addressed.

The resident system provides a Serial I/O Utility (SIO), that
provides a standardized high-level program interface to the bus.
SIO is utilized by the resident Diskette, Printer, and Cassette
handlers, and is intended to be used by nonresident handlers (see
Section 9), or by applications, as well. For a detailed
description of the program/SIO interface and for a detailed bus
specification refer to Section 9.

6   INTERRUPT PROCESSING

Section 6 describes system actions for the various interrupt
causing events, defines the many RAM vectors and provides
recommended procedures for dealing with interrupts.

The 6502 microcomputer processes three general interrupt types:
chip-reset, nonmaskable interrupts (NMI) and maskable interrupts
(IRQ). The IRQ interrupt type can be enabled and disabled using
the 6502 CLI and SEI instructions. The NMI type cannot be
disabled at the processor level; but the NMI interrupts other
than [SYSTEM.RESET] key can be disabled at the ANTIC chip.

The system events that can cause interrupts are listed below:

     chip-reset - power-up

     NMI - Display list interrupt (unused by OS)
           vertical-blank (50/60 Hz)
           [SYSTEM.RESET] key

     IRQ - Serial bus output ready
           Serial bus output complete
           Serial bus input ready
           Serial bus proceed line (unused by system)
           Serial bus interrupt line (unused by system)
           POKEY timers 1, 2 and 4
           Keyboard key
           [BREAK] key
           6502 BRK instruction (unused by OS)

Figure 6-1 List of System-Interrupt Events

The chip-reset interrupt is vectored via location FFFC to E477,
where a JMP vector to the power-up routine is located. All NMI
interrupts are vectored via location FFFA to the NMI interrupt
service routine at E7B4, and all IRQ interrupts are vectored
via location FFFE to the IRQ interrupt service routine at E6F3;
at that point the cause of the interrupt must be determined by
a series of tests. For some of the events there are built in
monitor actions and for other events the corresponding
interrupts are disabled or ignored. The system provides RAM
vectors so that you can intercept interrupts when
necessary.

CHIP-RESET

The OS generates chip-reset in response to a power-up condition.
The system is completely initialized (see Section 7).

NONMASKABLE INTERRUPTS

When an NMI interrupt occurs, control is transferred through
the ROM vector directly to the system NMI interrupt service
routine. A cause for the interrupt is determined by examining
hardware register NMIST [D40F]. The NMI makes a jump through the
global RAM vector VDSLST [0200] if a display list interrupt is
pending. The OS does not use display list interrupts, so VDSLST
is initialized to point to an RTI instruction, and you must not
change it before VDSLST generates a display interrupt.

If the interrupt is not a display-list interrupt, then a test is
made to see if it is a [SYSTEM.RESET] key interrupt. If so, then a
jump is made to the system reset initialization routine (see Section
7 for details of system reset initialization).

If the interrupt is neither a display list interrupt nor a
[SYSTEM.RESET] key interrupt; then it is assumed to be a
vertical-blank (VBLANK) interrupt, and the following actions
occur:

    Registers A, X and Y are pushed to the stack.

    The interrupt request is cleared (NMIRES [D40F]).

    A jump is made through the "immediate" vertical-blank global
    RAM vector VVBLKI [0222] that normally points to the Stage 1
    VBLANK processor.

The following actions occur assuming that you have not changed VVBLKI.

The stage 1 VBLANK processor is executed.

The OS tests to see if a critical code section has been interrupted. If so, then all registers are restored, and an RTI instruction returns from the interrupt to the critical section. A critical section is determined by examining the CRITIC flag [0042], and the processor I bit. If either are set, then the interrupted section is assumed to be critical.

If the interrupt was not from a critical section, then the stage 2 VBLANK processor is executed.

The OS then jumps through the "deferred" vertical-blank global RAM vector VBLKD [0224], that normally points to the VBLANK exit routine.

The following actions occur assuming that you have not changed VVBLKD.

o   The 6502 A, X and Y registers are restored.

o   An RTI instruction is executed.

NOTE: You can alter the deferred and immediate
VBLANK RAM vectors, but still enable normal system processes; or restore original vectors without having to save them. The instruction at E45F is a JMP to the stage 1 VBLANK processor; the address at [E460,2] is the value normally found in VVBLKI. The instruction at E462 is a JMP to the VBLANK exit routine; the address at [E463,2] is the value normally found in VVBLKD. These ROM vectors to stage 1 VBLANK processor and to the VBLANK exit routine will accomplish your goal.

NOTE: Every VBLANK interrupt jumps through vector VVBLKI. Only VBLANK interrupts from noncritical code sections jump through vector VVBLKD.

Stage 1 VBLANK Process

The following stage 1 VBLANK processing is performed at every VBLANK interrupt:

The stage 1 VBLANK process increments the 3-byte frame counter RTCLOK [0012-0014]; RTCLOK+0 is the MSB and RTCLOK+2 is the LSB. This counter wraps to zero when it overflows (every 77 hours or so), and continues counting.

The Attract mode variables are processed (see Appendix L, B10-12).

The stage 1 VBLANK process decrements the System Timer 1 CDTMV1 [0218,2] if it is nonzero; if the timer goes from

nonzero to zero then an indirect JSR is performed via CDTMA1
[0226,2].

Stage 2 VBLANK Process

The stage 2 VBLANK processing performs the following for those
VBLANK interrupts that do not interrupt critical sections:

The stage 2 VBLANK process clears the 6502 processor I bit.
This enables the IRQ interrupts.

The stage 2 VBLANK process updates various hardware
registers with data from the OS data base, as shown below:

Data Base Item	Hardware Register	Reason for Update
SDLSTH [0231]	DLISTH [D403]	Display list start
SDLSTL [0230]	DLISTL [D402]	
SDMCTL [022F]	DMACTL [D400]	
CHBAS [02F4]	CHBASE [D409]	
CHACT [02F3]	CHACTL [D401]	
GPRIOR [026F]	PRIOR [D01B]	
COLOR0 [02C4]	COLPF0 [D016]	Attract mode.
COLOR1 [02C5]	COLPF1 [D017]	
COLOR2 [02C6]	COLPF2 [D018]	
COLOR3 [02C7]	COLPF3 [D019]	
COLOR4 [02C8]	COLBK [D01A]	
PCOLR0 [02C0]	COLPM0 [D012]	
PCOLR1 [02C1]	COLPM1 [D013]	
PCOLR2 [02C2]	COLPM2 [D014]	
PCOLR3 [02C3]	COLPM3 [D015]	
Constant = 8	CONSOL [D01F]	Console speaker off.

The stage 2 VBLANK process decrements the System Timer 2
CDTMV2 [021A,2] if it is nonzero; if the timer goes from
nonzero to zero, then an indirect JSR is performed
through CDTMA2 [0228,2].

The stage 2 VBLAMK process decrements System Timers 3, 4 and
5 if they are nonzero; the corresponding flags are set to
zero for each timer that changes from nonzero to zero.

Timer	Timer Value	Timer Flag
3	CDTMV3 [021C,2]	CDTMF3 [022A,1]
4	CDTMV4 [021E,2]	CDTMF4 [022C,1]
5	CDTMV5 [0220,2]	CDTMF5 [022E,1]

A character is read from the POKEY keyboard register and stored in CH [02FC], if auto repeat is active.

The stage 2 VBLANK process decrements the keyboard debounce counter if it is not equal to zero, and if no key is pressed.

The stage 2 VBLANK process processes the keyboard auto repeat (see Appendix L, E8).

The stage 2 VBLANK process reads game controller data from the hardware to the RAM data base, as shown below:

Hardware Register		Data Base Item		Function
PORTA	[D300]	STICK0	[0278]	Joysticks and
		STICK1	[0279]	
		PTRIG0	[027C]	Paddle Controllers
		PTRIG1	[027D]	
		PTRIG2	[027E]	
		PTRIG3	[027F]	
PORTB	[D301]	STICK2	[027A]	
		STICK3	[027B]	
		PTRIG4	[0280]	
		PTRIG5	[0281]	
		PTRIG6	[0282]	
		PTRIG7	[0283]	
POT 0	[D200]	PADDL0	[0270]	Paddle Controllers
POT 1	[D201]	PADDL1	[0271]	
POT 2	[D202]	PADDL2	[0272]	
POT 3	[D203]	PADDL3	[0273]	
POT 4	[D204]	PADDL4	[0274]	
POT 5	[D205]	PADDL5	[0275]	
POT 6	[D206]	PADDL6	[0276]	
POT 7	[D207]	PADDL7	[0277]	
TRIG0	[D001]	STRIG0	[0284]	Joystick triggers.
TRIG1	[D002]	STRIG1	[0285]	
TRIG2	[D003]	STRIG2	[0286]	
TRIG3	[D004]	STRIG3	[0287]	

MASKABLE INTERRUPTS

An IRQ interrupt causes control to be transferred through the
immediate IRQ global RAM vector VIMIRQ [0216]. Ordinarily this
vector points to the system IRQ Handler. The Handler performs
these following actions:

The IRQ Handler determines a cause for the interrupt by
examining the IRQST [D20E] register and the PIA status
registers PACTL [D302] and PBCTL [D303]. The interrupt status bit
is cleared when it is found. One interrupt event is cleared and
processed for each interrupt-service entry. If multiple IRQs are
pending, then a separate interrupt will be generated for each
pending IRQ, until all are serviced.

The system IRQ interrupt service routine deals with each of the
possible IRQ causing events, in the following ways:

o   The 6502 A register is pushed to the stack.

o   If the interrupt is due to serial I/O bus output ready,
     then clear the interrupt and jump through global RAM
     vector VSEROR [020C].

o   If the interrupt is due to serial I/O bus input ready,
     then clear the interrupt and jump through global RAM
     vector VSERIN [020A].

o   If the interrupt is due to serial I/O bus output
     complete, then clear the interrupt and jump through
     global RAM vector VSEROC [020E].

o   If the interrupt is due to POKEY timer #1, then clear the
     interrupt and jump through global RAM vector VTIMR1 [0210].

o   If the interrupt is due to POKEY timer #2, then clear the
     interrupt and jump through global RAM vector VTIMR2 [0212].

o   If the interrupt is due to POKEY timer #4, then clear the
     interrupt. The service routine contains a bug, and falls
     into the following test.

o   If pressing a keyboard key caused the interrupt (other
     than [BREAK], [START], [OPTION], or [SELECT]); then clear the
     interrupt and jump through global RAM vector VKEYBD [0208].

o   If pressing the [BREAK] key caused the interrupt; then
     clear the interrupt. Set the BREAK flag BRKKEY [0011] to
     zero, proceed to clear the following:

          Start/stop flag SSFLAG [02FF]
          Cursor inhibit flag CRSINH [02F0]
          Attract mode flag ATRACT [004D]

Return from the interrupt after restoring the 6502 A
register from the stack.

o   If the interrupt is due to the serial I/O bus proceed line;
     then clear the interrupt, and jump through global RAM vector
     VPRCED [0202].

o   If the interrupt is due to the serial I/O bus interrupt
     line, then clear the interrupt and jump through global RAM
     vector VINTER [0204].

o   If the interrupt is due to a 6502 BRK instruction, then jump
     through global RAM vector VBREAK [0206].

o   If none of the above, restore the 6502 A register and return
     from the interrupt (RTI).

INTERRUPT INITIALIZATION

The interrupt subsystem completely reinitializes itself whenever
the system is powered up or the [SYSTEM.RESET] key is pressed.
The OS clears the hardware registers, and sets the interrupt
global RAM vectors to the following configurations:

Vector	Type	Function
VDSLST [0200]	NMI	RTI -- ignore interrupt.
VVBLKI [0222]	"	System stage 1 VBLANK.
CDTMA1 [0226]	"	SIO timeout timer.
CDTMA2 [0228]	"	No system function.
VVBLKD [0224]	"	System return from interrupt.
VIMIRQ [0216]	IRQ	System IRQ processor.
VSEROR [020C]	"	SIO.
VSERIN [020A]	"	SIO.
VSEROC [020E]	"	SIO.
VTIMR1 [0210]	"	PLA,RTI -- ignore interrupt.
VTIMR2 [0212]	"	PLA,RTI -- ignore interrupt.
VTIMR4 [0214]	"	*** doesn't matter ***
VKEYBD [0208]	"	System keyboard interrupt handler.
VPRCED [0202]	"	PLA,RTI -- ignore interrupt.
VINTER [0204]	"	PLA,RTI -- ignore interrupt.
VBREAK [0206]	BRK	PLA,RTI -- ignore interrupt.

Figure 6-2   Interrupt RAM Vector Initialization

System initialization sets the interrupt enable status
as follows:

        NMI   VBLANK enabled, display list disabled.

        IRQ   [BREAK] key and data key interrupts enabled, all others
                 disabled.

SYSTEM TIMERS

The OS contains five general purpose software timers, plus an
OS-supported frame counter. The timers are 2 bytes in length
(lo,hi) and the frame counter RTCLOK [0012] is three bytes in
length (hi,mid,lo). The timers count downward from any
nonzero value to zero. Upon reaching zero, they either clear
an associated flag, or JSR through a RAM vector. The frame
counter counts upward, wrapping to zero when it overflows.

The following table shows the timers and the frame counter
characteristics:

Timer Name	Flag/Vector	Use
* CDTMV1 [0218]	CDTMA1 [0226]	2-byte vector -- SIO timeout.
CDTMV2 [021A]	CDTMA2 [0228]	2-byte vector
CDTMV3 [021C]	CDTMF3 [022A]	1-byte flag
CDTMV4 [021E]	CDTMF4 [022C]	1-byte flag
CDTMV5 [0220]	CDTMF5 [022E]	1-byte flag
* RTCLOK [0012]		3-byte frame counter.

* These two timers are maintained as part of every VBLANK
  interrupt (stage 1 process). The other timers are subject to
  the critical section test (stage-2 process), that can defer
  their updating to a later VBLANK interrupt.

USAGE NOTES

This subsection describes the techniques you need to know in
order to utilize interrupts in conjunction with the operating
system.

POKEY Interrupt Mask

ANTIC (display-list and vertical-blank) and PIA (interrupt and
proceed lines) interrupts can be masked directly (see the
Hardware Manual). However, eight bits of a single byte IRQEN
[D20E] mask the POKEY interrupts ([BREAK] key, data key,
serial input ready, serial output ready, serial output done
and timers 1,2 and 4).

IRQEN is a write-only register. Thus, we must maintain a
current value of that register in RAM in order to update
individual mask bits selectively, while not changing other bits.
The name of the variable used is POKMSK [0010], and it is used
as shown in the examples below:


```
        ; EXAMPLE OF INTERRUPT ENABLE

                SEI                     ; TO AVOID CONFLICT WITH IRQ ...
                LDA     POKMSK  ; ... PROCESSOR WHICH ALTERS VAR.
                ORA     #$xx    ; ENABLE BIT(S).
                STA     POKMSK
                STA     IRQEN   ; TO HARDWARE REG TOO.
                CLI

        ; EXAMPLE OF INTERRUPT DISABLE

                SEI                     ; TO AVOID CONFLICT WITH IRQ ...
                LDA     POKMSK  ; ... PROCESSOR WHICH ALTERS VAR.
                AND     #$FF-xx ; DISABLE BIT(S).
                STA     POKMSK
                STA     IRQEN   ; TO HARDWARE REGISTER TOO.
                CLI
```

Figure 6-3   POKEY Interrupt Mask Example

Note that the OS IRQ service routine uses and alters POKMSK, so
alterations to the variable must be done with interrupts
inhibited. If done at the interrupt level there is no problem, as
the I bit is already set; if done at a background level then the
SEI and CLI instructions should be used as shown in the examples.

Setting Interrupt and Timer Vectors

Because vertical-blank interrupts are generally kept enabled so that
the frame counter RTCLOK is maintained accurately, there is a
problem with setting the VBLANK vectors (VVBLKI and VVBLKD) or
the timer values (CDTMV1 through CDTMV5) directly. A VBLANK
interrupt could occur when only one byte of the two-byte value had
been updated, leading to undesired consequences. For this reason,

the SETVBV [E45F] routine is provided to perform the desired
update in safe manner. The calling sequence is shown below:

        A = update item indicator
            1 - 5 for timers 1 - 5.
            6 for immediate VBLANK vector VVBLKI.
            7 for deferred VBLANK vector VVBLKD.
        X = MSB of value to store.
        Y = LSB of value to store.

        JSR   SETVBV

        The A, X and Y registers can be altered.
        The display list interrupt will always be disabled on
        return, even if enabled upon entry.

It is possible to fully process a vertical-blank interrupt
during a call to this routine.

When working with the System Timers, the vectors for timers 1 and
2 and the flags for timers 3, 4 and 5 should be set while the
associated timer is equal to zero, then the timer should be set
to its (nonzero) value.

Stack Content at Interrupt Vector Points

The following table shows the stack content at every one of the
RAM interrupt vector points:

INTERRUPT VECTOR	DESCRIPTION	OS RETURN CONTROL
VDSLST [0200]	Display list	return, P
VVBLKI [0222] *	VBLANK immediate	return, P, A, X, Y
CDTMA1 [0226]	System Timer 1	return, P, A, X, Y, return
CDTMA2 [0228]	System Timer 2	return, P, A, X, Y, return
VVBLKD [0224] *	VBLANK defer.	return, P, A, X, Y
VIMIRQ [0216] *	IRQ immediate	return, P, A
VSEROR [020C] *	Serial out ready	return, P, A
VSERIN [020A] *	Serial in ready	return, P, A
VSEROC [020E] *	Serial out compare	return, P, A
VTIMR1 [0210]	POKEY timer 1	return, P, A
VTIMR2 [0212]	POKEY timer 2	return, P, A
VTIMR4 [0214]	POKEY timer 4	return, P, A
VKEYBD [0208] *	Keyboard data	return, P, A
VPRSED [0202]	Serial proceed	return, P, A
VINTER [0204]	Serial interrupt	return, P, A
VBREAK [0206]	BRK instruction	return, P, A

Figure 6-4   Interrupt and Timer Vector RAM Stack Content Table

* The OS initializes these entries at power-up. Improperly
   changing these vectors will alter system performance.

Miscellaneous Considerations

The following paragraphs list a set of miscellaneous
considerations for the writer of an interrupt service routine.

   Restrictions on Clearing of "I" Bit

Display list, immediate vertical-blank and System Timer #1
routines should not clear the 6502 I bit. If the NMI leading to
one of these routines occurred while an IRQ was being processed,
then clearing the I bit will cause the IRQ to re-interrupt with
an unknown result.

The OS VBLANK processor carefully checks this condition after the
stage 1 process and before the stage 2 process.

   Interrupt Process Time Restrictions

You should not write an interrupt routine that exceeds 400 msec.
when added to the stage 1 VBLANK, if the serial I/O is being
used. The SIO sets the CRITIC flag while serial bus I/O is in
progress.

Interrupt Delay Due to "WAIT FOR SYNC"

Whenever a key is read from the keyboard, the Keyboard Handler
sets WSYNC [D40A] repeatedly while generating the audible click
on the console speaker. A problem occurs when interrupts are
generated during the wait-for-sync period; the processing of such
interrupts will be delayed by one horizontal scan line. This
condition cannot be prevented. You can work around the condition
by examining the line count VCOUNT [D40B] and delaying interrupt
processing by one line when no WSYNC delay has occurred.

FLOWCHARTS

The following pages contain process flowcharts showing the main
events that occur in the NMI and IRQ interrupt processes.

IRQ INTERRUPT PROCESS

NMI INTERRUPT PROCESS

```
                    ENTER
                      │
                      ▼
                  ┌─────────┐   Y   ┌─────────┐      ┌─────┐
                  │ DISPLAY │──────▶│ VDSLST  │─────▶│ RTI │
                  │  LIST?  │       └─────────┘      └─────┘
                  └─────────┘
                      │ N
                      ▼
                ┌────────────┐
                │ PUSH REG A │
                │  TO STACK  │
                └────────────┘
                      │
                      ▼
                 ┌──────────┐  N   ┌───────┐
                 │ VERTICAL │─────▶│ RESET │
                 │  BLANK?  │      └───────┘
                 └──────────┘
                      │
                      ▼
              ┌──────────────┐    ┌────────┐    ┌─────────┐
              │  PUSH X & Y, │───▶│ VVBLKI │───▶│ STAGE 1 │──┐
              │ CLEAR STATUS │    └────────┘    └─────────┘  │
              └──────────────┘                               │
                                                             │
                      ┌──────────────────────────────────────┘
                      ▼
          Y      ┌──────────┐
      ┌──────────│ CRITICAL │
      │          │ SECTION? │
      │          └──────────┘
      │               │ N
      │               ▼
      │          ┌───────┐    ┌─────────┐    ┌────────┐
      │          │ CLEAR │───▶│ STAGE 2 │───▶│ VVBLKD │──┐
      │          │ I BIT │    └─────────┘    └────────┘  │
      │          └───────┘                               │
   ┌────────┐        │          ┌───────────────────────┘
   │ XITVBL │────────┼──────────┘
   └────────┘        ▼
              ┌──────────┐
              │ RESTORE  │
              │ REGISTERS│
              └──────────┘
                   │
                   ▼
                ┌─────┐
                │ RTI │
                └─────┘
```
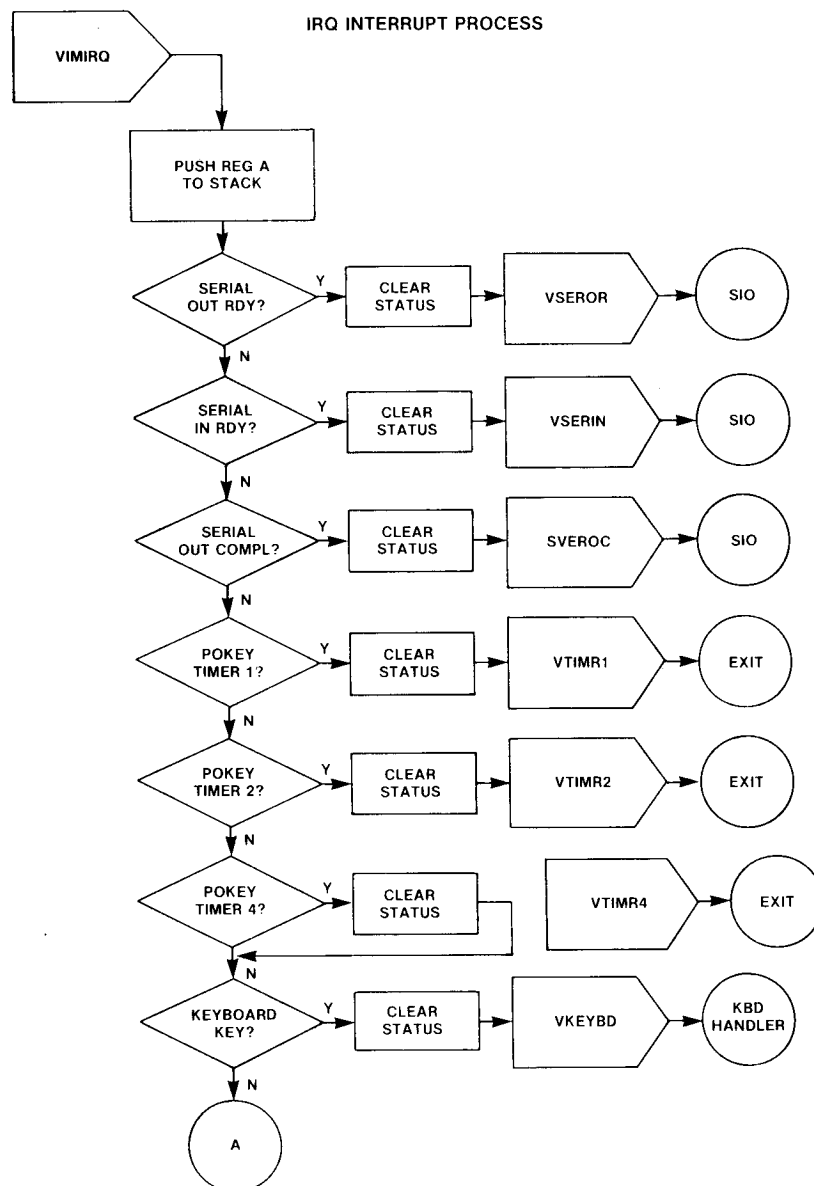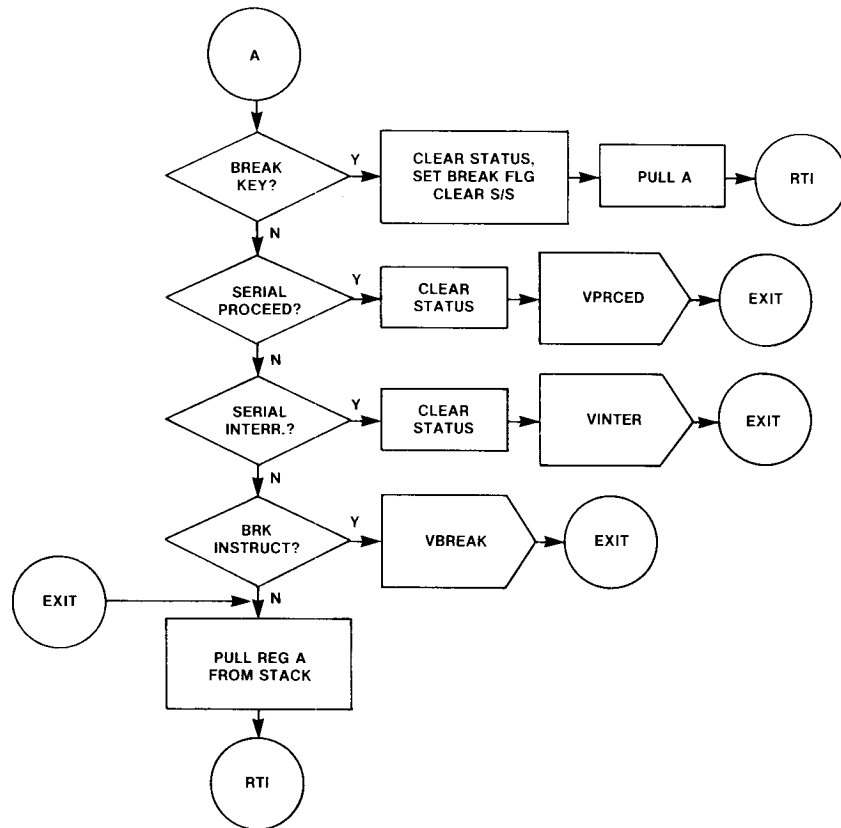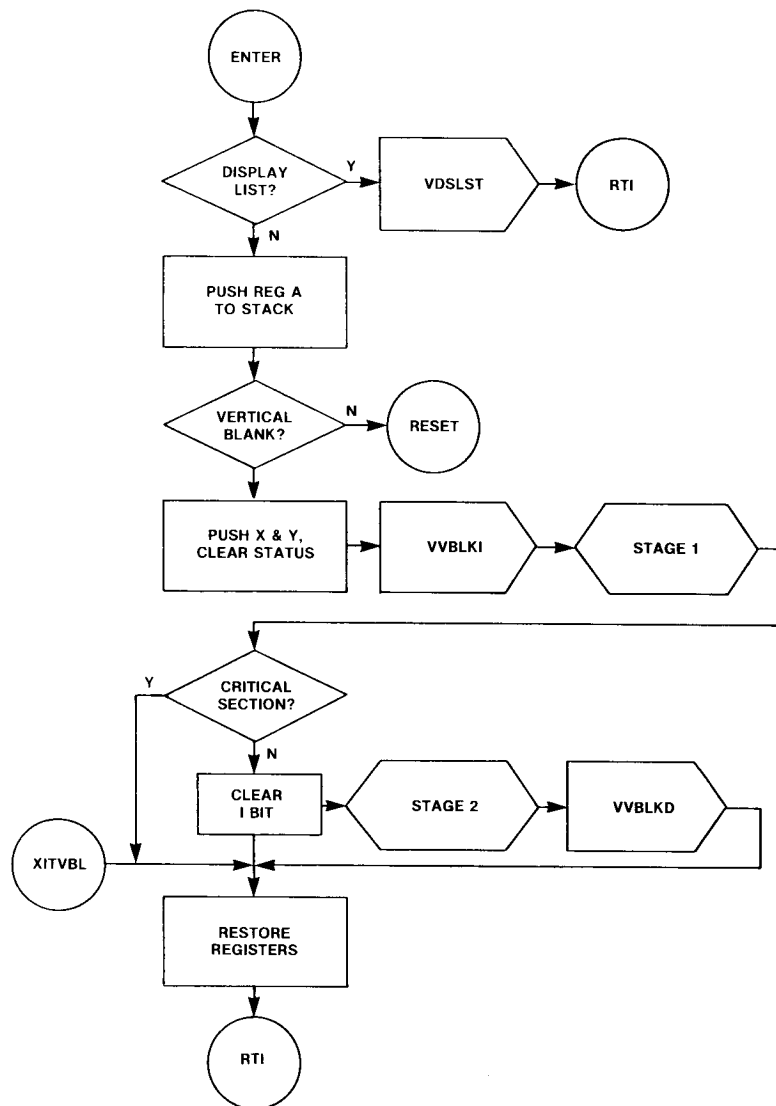
7   SYSTEM INITIALIZATION

Section 7 discusses the details of the power-up and
system reset processes. The power-up process will be explained
first, and then the system reset process will be explained in
terms of its differences from the power-up process.

Both power-up (also called coldstart) and pressing [SYSTEM.RESET]
(warmstart) will cause system initialization: In addition, there
are vectors for these processes at E474 (system reset) and E477
(power-up) so that they can be user-initiated.

The power-up initialization process is a superset of the
system reset initialization process. Power-up initializes both
the OS and user RAM regions, whereas system reset initializes
only the OS RAM region. In both cases, the OS calls the outer
level software initialization entry points allow the application
to initialize its own variables.

Pressing the [SYSTEM.RESET] key produces an NMI interrupt. It
does not perform a 6502 chip-reset. If the processor is locked
up, the [SYSTEM.RESET] key cannot be sufficient to unlock it, and the
system must have power cycled to clear the problem.

POWER-UP INITIALIZATION (COLDSTART) PROCEDURE

The OS performs the following functions in the order shown, as
part of the power-up initialization process:

1. The following 6502 processor states are set:

o   IRQ interrupts are disabled using the SEI instruction.

o   The decimal flag is cleared using the CLD instruction.

o   The stack pointer is set to $FF.

2. The OS sets the warmstart flag WARMST [0008] to 0 (false).

3. The OS tests to see if a diagnostic cartridge is in the A slot:

o   Cartridge address BFFC = 00?

o   The memory at BFFC is not RAM?

o   Bit 7 of the byte at BFFD = 1?

    If all of the above tests are true, then control is passed to
    the diagnostic cartridge via the vector at BFFE. No return is
    expected.

4. The OS determines the lowest memory address containing
   non-RAM, by testing the first byte of every 4K "block" to see
   if the content can be complemented. If it can be complemented,
   then the original value is restored and testing continues. If
   it can't be complemented; then the content is assumed to be
   the first non-RAM address in the system. The MSB of the
   address is stored temporarily in TRAMSZ [0006].

5. Zero is stored to all of the hardware register addresses shown
   below (most of that aren't decoded by the hardware):

        D000 through D0FF
        D200 through D2FF
        D300 through D3FF
        D400 through D4FF

6. The OS clears RAM from location 0008, to the address
   determined in step 4, above.

7. The default value for the "noncartridge" control vector
   DOSVEC [000A] is set to point to the blackboard routine. At
   the end of initialization, control is passed through this
   vector if a cartridge does not take control.

8. The coldstart flag COLDST [0244] is set to -1 (local use).

9. The screen margins are set: left margin = 2, right margin =
   39, for a 38 character physical line. The maximum line size of
   40 characters can be obtained by setting the margins to 0 and
   39. The OS insets the left margin because the two leftmost
   columns of the video picture on many television sets are not
   entirely visible on the screen.

10. The interrupt RAM vectors VDSLST [0200] through VVBLKD [0224]
    are initialized. See Section 6 for the initialization values.

11. Portions of the OS RAM are set to their required nonzero values
    as shown below:

o   The [BREAK] key flag BRKKEY [0011] = -1 (false).

o   The top of memory pointer MEMTOP [02E5] = the lowest
         non-RAM address (from step 4); MEMTOP will be altered
         later when the Screen Editor is opened in step 15.

o   The bottom of memory pointer MEMLO [02E7] = 0700; MEMLO
         can be changed later if there is either a diskette- or
         cassette-boot operation.

o   The following resident routines are called for initialization:

         Screen Editor
         Display Handler
         Keyboard Handler
         Printer Handler
         Cassette Handler
         Central I/O Monitor (CIO)
         Serial I/O Monitor (SIO)
         Interrupt processor

o   The [START] key is checked, and if pressed, the cassette-boot
         request flag CKEY [004A] is set.

12. 6502 IRQ interrupts are enabled using the CLI instruction.

13. The device table HATABS [031A] is initialized to point to the
    resident handlers. See Section 9 for information relating to
    the Device Handler table.

14. The cartridge slot addresses for cartridges B and A are
    examined to determine if cartridges are inserted, if RAM does
    not extend into the cartridge address space.

    If the content of location 9FFC is zero, then a JSR is
    executed through the vector at 9FFE, thus initializing
    cartridge "B". The cartridge is expected to return.

    If the content of location BFFC is zero, then a JSR is
    executed through the vector at BFFE, thus initializing
    cartridge "A". The cartridge is expected to return.

15. IOCB #0 is set up for an OPEN of the Screen Editor (E) and
    the OPEN is performed. The Screen Editor will use the highest
    portion of RAM for the screen and will adjust MEMTOP
    accordingly. If this operation should fail, the entire
    initialization process is repeated.

16. A delay is effected to assure that a VBLANK interrupt has
    occurred. This is done so that the screen will be established
    before continuing.

17. If the cassette-boot request flag is set (see step 11 above),
    then a cassette-boot operation is attempted. See Section 10

for details of the cassette-boot operation.

18. If any of the three conditions stated below exists, an attempt is made to boot from the disk.

    There are no cartridges in the slots.

    Cartridge B is inserted and bit 0 of 9FFD is 1.

    Cartridge A is inserted and bit 0 of BFFD is 1.

    See Section 10 for details of the diskette-boot operation.

19. The coldstart flag COLDST is reset to indicate that the coldstart process went to completion.

20. The initialization process is now complete, and the controlling application is now determined via the remaining steps.

    If there is an A cartridge inserted and bit-2 of BFFD is 1, then a JMP is executed through the vector at BFFA.

    Or, if there is a B cartridge inserted and bit-2 of 9FFD is 1, then a JMP is executed through the vector at 9FFA.

    Or, a jump is executed through the vector DOSVEC that can point to the blackboard routine (default case), cassette booted software or diskette booted software. DOSVEC can be altered by the booted software as explained in Section 10.

SYSTEM RESET INITIALIZATION (WARMSTART) PROCEDURE

The functions listed below are performed, in the order shown, as part of the system reset initialization process:

A. Same as power-up step 1.

B. The warmstart flag WARMST [0008] is set to -1 (true).

C. Same as power-up steps 3 through 5.

D. OS RAM is zeroed from locations 0200-03FF and 0010-007F.

E. Same as power-up steps 9 through 16.

F. If a cassette-boot was successfully completed during the power-up initialization, then a JSR is executed through the vector CASINI [0002]. See Section 10 for details of the cassette-boot process.

G. Same as power-up step 18, except instead of booting the diskette software, a JSR is executed through the vector DOSINI [000C] if the diskette-boot was successfully completed during the Power-up initialization. See Section 10 for details of the diskette-boot process.

H. Same as power-up steps 19 and 20.

Note that the initialization procedures and main entries for all software entities are executed at every system reset as well as at power up (see steps 14, 17, 18, 20, F and G). If the user-supplied initialization/startup code must behave differently in response to system reset than it does to power-up, then the warmstart flag WARMST [0008] should be interrogated; WARMST = 0 means power-up entry, else system reset entry.

8 FLOATING POINT ARITHMETIC PACKAGE

This section describes the BCD floating point (FP) package that
is resident in the OS ROM in both the models 400 and 800.

The floating point package maintains numbers internally as 6-byte
quantities: a 5-byte (10 BCD digit) mantissa with a 1-byte
exponent. BCD internal representation was chosen so that decimal
division would not lead to the rounding errors typically found in
binary representation implementations.

The package provides the following operations:

        ASCII to FP conversion.
        FP to ASCII conversion.
        Integer to FP conversion.
        FP to integer conversion.
        FP add, subtract, multiply, and divide.
        FP logarithm, exponentiation, and polynomial evaluation.
        FP zero, load, store, and move.

A floating point operation is performed by calling one of the
provided routines (each at a fixed address in ROM) after having
set one or more floating point pseudo registers in RAM. The
result of the desired operation will also involve floating point
pseudo registers. The primary pseudo registers are described
below and their addresses given within the square brackets:

```
        FRO [00D4] =      6-byte internal form of FP number.
        FR1 [00E0] =      6-byte internal form of FP number.
        FLPTR [00FC] =    2-byte pointer (lo,hi) to a FP.
                          number.
        INBUFF [00F3] =   2-byte pointer (lo,hi) to an ASCII text
                          buffer.
        CIX [00F2] =      1-byte index, used as offset to buffer
                          pointed to by INBUFF.
        LBUFF [0580] =    result buffer for the FASC routine.
```

FUNCTIONS/CALLING SEQUENCES

Descriptions of these floating point routines assume that
a pseudo register is not altered by a given routine. The
numbers in square brackets [xxxx] are the ROM addresses of the
routines.

ASCII to Floating Point Conversion (AFP)

Function: This routine takes an ASCII string as input and
produces a floating point number in internal form.

Calling sequence:

```
    INBUFF = pointer to buffer containing the ASCII
             representation of the number.
    CIX = the buffer offset to the first byte of the ASCII
          number.

    JSR       AFP [D800]
    BCS       first byte of ASCII number is invalid

    FRO = floating point number.
    CIX = the buffer offset to the first byte after the ASCII
          number.
```

Algorithm: The routine takes bytes from the buffer until it
encounters a byte that cannot be part of the number. The bytes
scanned to that point are then converted to a floating point
number. If the first byte encountered is invalid, the carry bit
is set as a flag.

Floating Point to ASCII Conversion (FASC)

Function: This routine converts a floating point number from
internal form to its ASCII representation.

Calling sequence:

    FR0 = floating point number.

    JSR   FASC [D8E6]

    INBUFF = pointer to the first byte of the ASCII number.
    The last byte of the ASCII representation has the most
    significant bit (sign bit) set; no EOL follows.

Algorithm: The routine converts the number from its internal
floating point representation to a printable form (ATASCII). The
pointer INBUFF will point to part of LBUFF, where the result is
stored.

Integer to Floating Point Conversion (IFP)

Function: This routine converts a 2-byte unsigned integer (0 to
65535) to floating point internal representation.

Calling sequence:

    FR0 = integer (FR0+0 = LSB, FR0+1 = MSB).

    JSR   IFP [D9AA]

    FR0 = floating point representation of integer.

Floating Point to Integer Conversion (FPI)

Function: This routine converts a positive floating point number
from its internal representation to the nearest 2-byte integer.

Calling sequence:

    FR0 = floating point number.

    JSR   FPI [D9D2]
    BCS   FP number is negative or >= 65535.5

    FR0 = 2-byte integer (FR0+0 = LSB, FR0+1 = MSB).

Algorithm: The routine performs true rounding, not truncation,
during the conversion process.

Floating Point Addition (FADD)

Function: This routine adds two floating point numbers and checks
the result for out-of-range.

Calling sequence:

        FRO = floating point number.
        FR1 = floating point number.

        JSR   FADD [DA66]
        BCS   out-of-range result.

        FRO = result of FRO + FR1.
        FR1 is altered.

Floating Point Subtraction (FSUB)

Function: This routine subtracts two floating point numbers and
checks the result for out-of-range.

Calling sequence:

        FRO = floating point minuend.
        FR1 = floating point subtrahend.

        JSR   FSUB [DA60]
        BCS   out-of-range result.

        FRO = result of FRO - FR1.
        FR1 is altered.

Floating Point Multiplication (FMUL)

Function: This routine multiplies two floating point numbers and
checks the result for out-of-range.

Calling sequence:

        FRO = floating point multiplier.
        FR1 = floating point multiplicand.

        JSR   FMUL [DADB]
        BCS   out-of-range result.

        FRO = result of FRO * FR1.
        FR1 is altered.

Floating Point Division (FDIV)

Function: This routine divides two floating point numbers and checks for division by zero and for result out-of-range.

Calling sequence:

```
    FR0 = floating point dividend.
    FR1 = floating point divisor.

    JSR        FDIV [DB28]
    BCS        out-of-range result or divisor is zero.

    FR0 = result of FR0 / FR1.
    FR1 is altered.
```

Floating Point Logarithms (LOG and LOG10)

Function: These routines take the natural or base 10 logarithms of a floating point number.

Calling sequence:

```
    FR0 = floating point number.

    JSR        LOG [DECD] for natural logarithm
or
    JSR        LOG10 [DED1] for base 10 logarithm
    BCS        negative number or overflow.

    FR0 = floating point logarithm.
    FR1 is altered.
```

Algorithm: Both logarithms are first computed as base 10 logarithms using a 10 term polynomial approximation; the natural logarithm is computed by dividing the base 10 result by the constant LOG10(e).

The logarithm of a number Z is computed as follows:

```
    F * (10 ** Y) = Z where 1 <= F < 10 (normalization).
    L = LOG10(F) by 10 term polynomial approximation.
    LOG10(Z) = Y + L.   LOG(Z) = LOG10(Z) / LOG10(e).
```

NOTE: This routine does not return an error if the number input is zero; the LOG10 result in this case is approximately -129.5, which is not useful.

Floating Point Exponentiation (EXP and EXP10)

Function: This routine exponentiates.

Calling sequence:

    FRO = floating point exponent (Z).

    JSR   EXP [DDCO] for e ** Z
or
    JSR   EXP10 [DDCC] for 10 ** Z
    BCS   overflow.

    FRO = floating point result.
    FR1 is altered.

Algorithm: Both exponentials are computed internally as base 10,
with the base e exponential using the identity:
e ** X = 10 ** (X * LOG10(e)).

The base 10 exponential is evaluated in two parts using the identity:

10 ** X = 10 ** (I + F) = (10 ** I) * (10 ** F) -- where I is the
integer portion of X and F is the fraction.

The term 10 ** F is evaluated using a polynomial approximation,
and 10 ** I is a straightforward modification to the floating
point exponent.

Floating Point Polynomial Evaluation (PLYEVL)

Function: This routine performs an n degree polynomial
evaluation.

Calling sequence:

    X,Y = pointer (X = LSB) to list of FP coefficients (A(i))
          ordered from high order to low order (six bytes per
          coefficient).
    A = number of coefficients in list.
    FRO = floating point independent variable (Z).

    JSR   PLYEVL [DD40]
    BCS   overflow or other error.

    FRO = result of A(n)*Z**n + A(n-1)*Z**n-1 ... + A(1)*Z +
          A(0).
    FR1 is altered.

Algorithm: The polynomial P(Z) = SUM(i=0 to n) (A(i)*Z**i) is
computed using the standard method shown below:

    P(Z) = (...(A(n)*Z + A(n-1))*Z + ... + A(1))*Z + A(0)

Clear FRO (ZFRO)

Function: This routine sets the contents of pseudo register FRO
to all zeros.

Calling sequence:

        JSR   ZFRO [DA44]

        FRO = zero.

Clear Page Zero Floating Point Number (ZF1)

Function: This routine sets the contents of a zero-page floating
point number to all zeroes.

Calling sequence:

        X = Zero-page address of FP number to clear.

        JSR   ZF1 [DA46]

        Zero-page FP number(X) = zero.

Load Floating Point Number to FRO (FLDOR and FLDOP)

Function: These routines load pseudo register FRO with the
floating point number specified by the calling sequence.

Calling sequences:

        X,Y = pointer (X = LSB) to FP number.

        JSR   FLDOR [DD89]

or

        FLPTR = pointer to FP number.

        JSR   FLDOP [DD8D]

        FRO = floating point number (in either case).
        FLPTR = pointer to FP number (in either case).

Load Floating Point Number to FR1 (FLD1R and FLD1P)

Function: These routines load pseudo register FR1 with the
floating point number specified by the calling sequence.

Calling sequences:

   As in prior description, except the result goes to FR1
   instead of FR0. FLD1R [DD98] and FLD1P [DD9C].

Store Floating Point Number From FR0 (FSTOR and FSTOP)

Function: These routines store the contents of pseudo register
FR0 to the address specified by the calling sequence:

Calling sequence:

   As in prior descriptions, except the floating point number
   is stored from FR0 rather than loaded to FR0. FSTOR [DDA7]
   and FSTOP [DDAB].

Move Floating Point Number From FR0 to FR1 (FMOVE)

Function: This routine moves the floating point number in FR0 to
pseudo register FR1.

Calling sequence:

   JSR   FMOVE [DDB6]

   FR1 = FR0   (FR0 remains unchanged).

RESOURCE UTILIZATION

The floating point package uses the following RAM locations in
the course of performing the functions described in this section:

   00D4 through 00FF
   057E through 05FF

All of these locations are available for program coding
if your program does not call the floating point package.

IMPLEMENTATION DETAILS

Floating point numbers are maintained internally as 6-byte quantities, with 5 bytes (10 BCD digits) of mantissa and 1 byte of exponent. The mantissa is always normalized such that the most significant byte is nonzero (note "byte" and not "BCD digit").

The most significant bit of the exponent byte provides the sign for the mantissa; 0 for positive and 1 for negative. The remaining 7 bits of the exponent byte provide the exponent in excess 64 notation. The resulting number represents powers of 100 decimal (not powers of 10). This storage format allows the mantissa to hold 10 BCD digits when the value of the exponent is an even power of 10, and 9 BCD digits when the value of the exponent is an odd power of 10.

The implied decimal point is always to the immediate right of the first byte. An exponent less than 64 indicates a number less than 1. An exponent equal to or greater than 64 represents a number equal to or greater than 1.

Zero is represented by a zero mantissa and a zero exponent. To test for a result from any of the standard routines; test either the exponent or the first mantissa byte for zero.

The absolute value of floating point numbers must be greater than $10**-98$, and less than $10**+98$, or be equal to zero. There is perfect symmetry between positive and negative numbers with the exception that negative zero is never generated.

The precision of all computations is maintained at 9 or 10 decimal digits, but accuracy is somewhat less for those functions involving polynomial approximations (logarithm and exponentiation). Also, the problems inherent in all floating point systems are present here; for example: subtracting two very nearly equal numbers, adding numbers of disparate magnitude, or successions of any operation, will all result in a loss of significant digits. An analysis of the data range and the order of evaluation of expressions may be required for some types of applications.

The examples below compare floating point numbers with their internal representations, as an aid to understanding storage format. All numbers prior to this point have been expressed in decimal notation, but these examples will use hexadecimal notation. Note that 64 decimal (the excess number of the exponent) is 40 when expressed in hexadecimal:

```
Number:  +0.02 = 2 * 10**-2 = 2 * 100**-1
Stored:  3F 02 00 00 00 00   (FP exponent = 40 - 1)

Number:  -0.02 = -2 * 10**-2 = -2 * 100**-1
Stored:  BF 02 00 00 00 00   (FP exponent = 80 + 40 - 1)
```

```
Number: +37.0 = 3.7 * 10**1 = 37 * 100**0
Stored: 40 37 00 00 00 00   (FP exponent = 40 + 0)

Number: -4.60312486 * 10**11 = -46.03... * 100**5
Stored: C5 46 03 01 24 86   (FP exponent = 80 + 40 + 5)

Number: 0.0
Stored: 00 00 00 00 00 00   (special case)
```

9   ADDING NEW DEVICE HANDLERS/PERIPHERALS

This section describes the interface requirements for a nonresident Device Handler that is to be accessed via the Central I/O utility (CIO). The Serial bus I/O utility (SIO) interface is defined for those handlers that utilize the Serial I/O bus.

The I/O subsystem is organized with three levels of software between you and your hardware: The CIO, the individual device handlers, and the SIO.

The CIO performs the following functions:

Logical device name to Device Handler mapping (on OPEN).

I/O Control Block (IOCB) maintenance.

Logical record handling.

User buffer handling.

The device handlers are below CIO. They perform the following functions:

Device initialization on power-up and system reset.

Device-dependent support of OPEN and CLOSE commands.

Byte-at-a-time data input and output.

Device-dependent special operations.

Device-dependent command support.

Device data buffer management.

The SIO is at the bottom level (for Serial I/O bus peripheral handlers). It performs the following functions:

Control of all Serial bus I/O, conforming to the bus protocol.

Bus operation retries on errors.

Return of unified error statuses on error conditions.

OPERATING SYSTEM CO16555 -- Section 9

A separate control structure is used for communication at each
interface, as follows:

      User/CIO   I/O Control Block (IOCB)

      CIO/Handler   Zero-page IOCB (ZIOCB)

      Handler/SIO   Device Control Block (DCB)

```
                        +----------+
                        :  user    :
                        :  program :------------------------------+
                        +----------+                              :
+----------+            :                                         :
:  IOCB's  :**********:                                           :
+----------+            :                                         :
                        +----------+                              :
                        :   CIO    :        +----------+          :
                        : utility  :        :   DCB    :*******:
                        +----------+        +----------+   *     :
                        :                                  *     :
                        :                                  *     :
+----------+            +----------+        +----------+   *     :
:  ZIOCB   :            :  Device  :        :Disk File:   *     :
+----------+            :  Table   :    +----: Manager :-----+   :
    *                   +----------+    :   +----------+     :   :
    *                   :               :                   :   :
+-------------+---------+-------+--+--.....---+             :   :
:             :                :            :              :   :
   +----------+   +----------+   +----------+   +----------+   +----------+
   : Printer  :   : Cassette:   :   ...    :   : Keyboard:   :  Disk    :
   : Handler  :   : Handler  :   : Handler  :   : Handler :   : Handler  :
   +----------+   +----------+   +----------+   +----------+   +----------+
   :             :                :                          :
+-------------+---------+-------+--.....--------------+
                 :
+----------+     :
:   DCB    :**********:
+----------+     :
                +----------+
                :   SIO    :
                : Utility  :
                +----------+
```
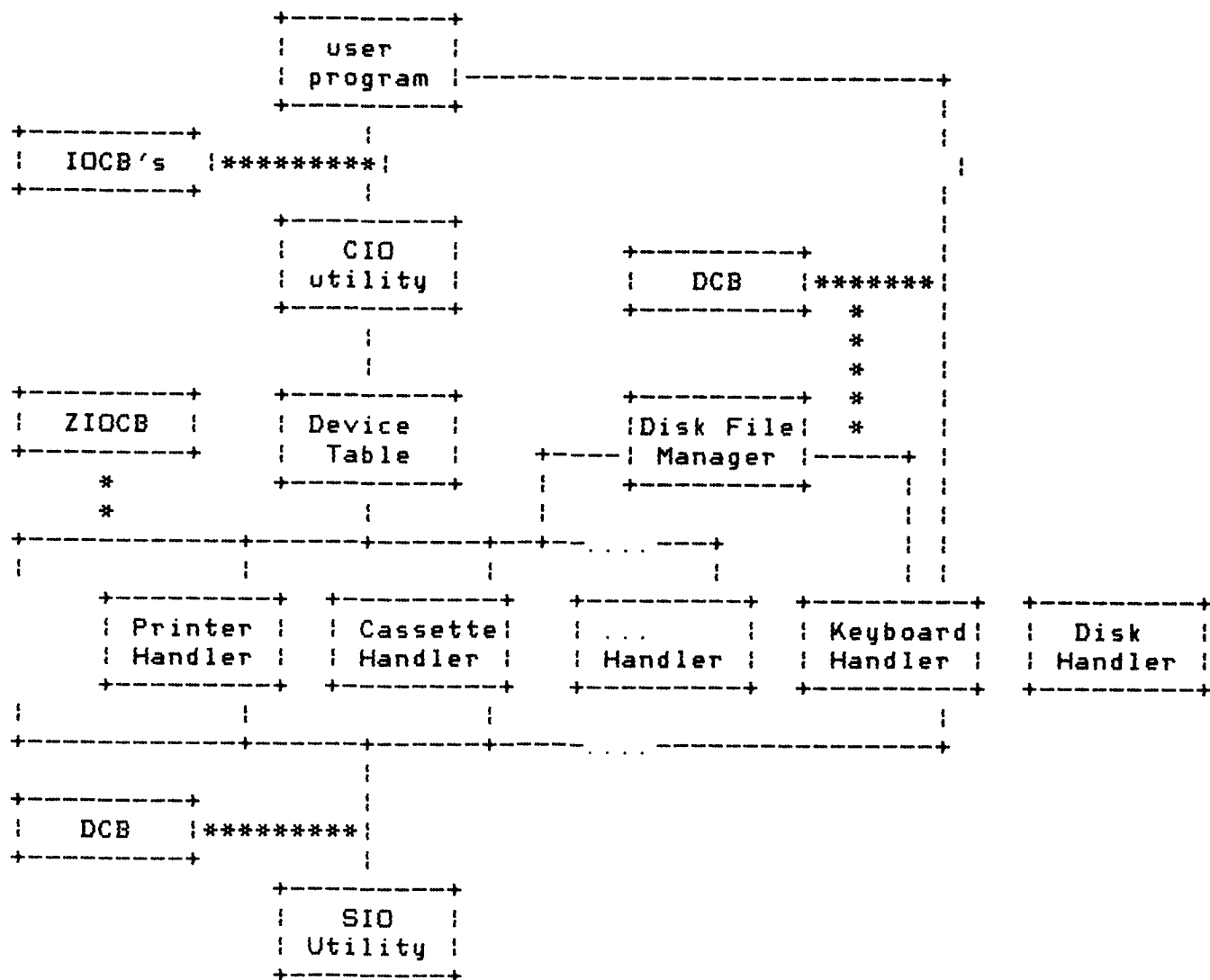
Where: ---- shows a control path.
        **** shows the data structure required for a path.

Note the following:

  1. The Keyboard/Display/Screen Editor handlers don't use
     SIO.
  2. The Diskette Handler cannot be called directly from CIO.
  3. The DCB is shown twice in the diagram.

Figure 9-1   I/O Subsystem Flow Diagram

DEVICE TABLE

The device table is a RAM-resident table that contains the
single-character device name (e.g. K, D, C, etc). and the
handler address for each of the handlers known to CIO. The
table is initialized to contain entries for the following
resident handlers: Keyboard (K), Display (S), Screen Editor
(E), Cassette (C), and Printer (P) at power-up and system reset. To
install a new handler, some procedure must insert a device table entry
after the table is initialized.

The table format is shown below:

```
                    +------------------+   -+
HATABS [031A]       !   device name    !    !
                    +------------------+    !
                    !  handler vector  !    +- one entry
                    +                  +    !
                    !  table address   !    !
                    +------------------+   -+
                    !       more       !
                    =                  =
                    !     entries      !
                    +------------------+
                    !   zero fill to   !
                    =                  =
                    !   end of table   !
                    +------------------+
```

Figure 9-2   Device Table Format

This 38-byte table will hold a maximum of 12 entries, with the
last 2 bytes being zero. CIO scans the table from the end to
the beginning (high to low address); so the entry nearest the
end of the table will take precedence in case of multiple
occurrences of a device name..

The device name for each entry is a single ATASCII character, and
the handler address points to the handler's vector table, that
will be described in the following section.

CIO/HANDLER INTERFACE

This section describes the interface between the Central I/O
utility and the individual device handlers that are represented
in the Device Table (as described in the preceding section).

Calling Mechanism

Each handler has a vector table as shown below:

```
        +----------------+
        + OPEN vector    +       (low address)
        +----------------+
        + CLOSE vector   +
        +----------------+
        + GETBYTE vector +
        +----------------+
        + PUTBYTE vector +
        +----------------+
        + GETSTAT vector +
        +----------------+
        + SPECIAL vector +
        +----------------+
        + JMP init code  +
        +                +       (high address)
        +----------------+
```

Figure 9-3 Handler Vector Table

The device table entry for the handler points to the first
byte of the vector table.

The first six entries in the table are vectors (lo,hi) that
contain the address - 1 of the handler routine that handles
the indicated function. The seventh entry is a 6502 JMP
instruction to the handler initialization routine. CIO uses
only the addresses contained in this table for handler entry.
Each user/CIO command translates to one or more calls to one
of the handler entries defined in the vector table.

The vector table provides the handler addresses for certain
fixed functions to be performed to CIO. In addition, operation
parameters also must be passed for most functions. Parameter
passing is accomplished using the 6502 A, X, and Y registers
and an IOCB in page 0 named ZIOCB [0020]. In general, register
A is used to pass data, register X contains the index to the
originating IOCB, and register Y is used to pass status
information to CIO. The zero-page IOCB, is a copy of the
originating IOCB; but in the course of processing some
commands, CIO can alter the buffer address and buffer length
parameters in ZIOCB, but not in the originating IOCB (see
Section 5 for information relating to the originating IOCB).

See Appendix B for the standard status byte values to be
returned to CIO in register Y.

The following sections describe the CIO/handler interface for each of the vectors in the handler vector table.

Handler Initialization

NOTE: This entry doesn't appear to have any function for nonresident handlers due to a bug in the current OS -- the device table is cleared in response to system reset as well as power-up. This prevents this entry point from ever being called. The rest of this section discusses the intended use of this entry point. Conformation would be in order to allow compatibility with possible corrected versions of the OS in the future.

The entry was to have been called on all occurrences of power-up and system reset; the handler is to perform initialization of its hardware and RAM data using a routine that assures proper processing of all CIO commands that follow.

Functions Supported

This section describes the functions associated with the first six vectors from the handler vector table. This section also presents a brief, device-independent description of the CIO/handler interface and recommended actions for each function vector.

OPEN

This entry is called in response to an OPEN command to CIO. The handler is expected to validate the OPEN parameters and perform any required device initialization associated with a device OPEN.

At handler entry, the following parameters can be of interest:

    X = index to originating IOCB.
    Y = $92 (status = function not implemented by handler).

    ICDNOZ [0021] = device number (1-4, for multiple device
                        handlers).
    ICBALZ/ICBAHZ [0024/0025] = address of device/filename
                                    specification.
    ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler attempts to perform the indicated OPEN and indicates the status of the operation by the value of the Y register. The responsibility for checking for multiple OPENs to

the same device or file, where it is illegal, lies with the
handler.

   CLOSE

This vector table entry is called in response to a CLOSE command
to CIO. The handler is expected to release any held resources
that relate specifically to that device/filename, and for output
files to:

      1) send any data remaining in handler buffers to the device,

      2) mark the end of file

      3) update any associated directories, allocation maps, etc.

At handler entry, the following parameters can be of interest:

      X = index to originating IOCB.
      Y = $92 (status = function not implemented by handler).

      ICDNOZ [0021] = device number (1-4, for multiple device
                          handlers).
      ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler attempts to perform the indicated CLOSE and
indicates the status of the operation by the value of the Y
register.

CIO releases the associated IOCB after the handler returns,
regardless of the operation status value.

   GETBYTE

This vector table entry is called in response to a GET
CHARACTERS or GET RECORD command to CIO. The handler is
expected to return a single byte in the A register, or return an
error status in the Y register.

At handler entry, the following parameters can be of interest:

      X = index to originating IOCB.
      Y = $92 (status = function not implemented by handler).

      ICDNOZ [0021] = device number (1-4, for multiple device handlers).
      ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler will obtain a data byte directly from the device or from a
handler-maintained buffer and return to CIO with the byte in the
A register and the operation status in the Y register.

Handlers that do not have short timeouts associated with the
reading of data (such as the Keyboard and Cassette Handlers),
must monitor the [BREAK] key flag BRKKEY [0011] and return with a
status of $80 when a [BREAK] condition occurs. See Appendix L,
E5; and Section 12 for a discussion of [BREAK] key monitoring.

CIO checks for reads from device/files that have not been opened
or have been opened for output only; the handler will not be called in
those cases.

   PUTBYTE

This entry is called in response to a PUT CHARACTERS or PUT
RECORD command to CIO. The handler is expected to accept a single
byte in the A register or return an error status in the Y
register.

At handler entry, the following parameters can be of interest:

    X = index to originating IOCB.
    Y = $92 (status = function not implemented by handler).
    A = data byte.

    ICDNOZ [0021] = device number (1-4, for multiple device
                    handlers).
    ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler sends the data byte directly to the device, or to a
handler-maintained buffer, and returns to CIO with the operation
status in the Y register. If a handler-maintained buffer fills,
the handler will send the buffered data to the device before
returning to CIO.

CIO checks for WRITEs to device/files that have not been opened,
or have been opened for input only. The handler will not be called in
those cases.

Now that the normal operation of PUTBYTE has been defined, a
special case must be added. Any handler that will operate within
the environment of the ATARI 8K BASIC language interpreter has a
different set of rules. Because BASIC can call the handler
PUTBYTE entry directly, without going through CIO, the zero-page
IOCB (ZIOCB) can or may not have a relation to the PUTBYTE call.
Therefore, the handler must use the outer level IOCB to obtain
any information that would normally be obtained from ZIOCB. Note
also that the OPEN protection normally provided by CIO is
bypassed (i.e. PUTBYTE to a non-OPEN device/file and PUTBYTE to a
read-only OPEN).

GETSTAT

This entry is called in response to a GET STATUS command to CIO.
The handler is expected to return four bytes of status to memory
or return an error status in the Y register.

At handler entry, the following parameters can be of interest:

    X = index to originating IOCB. Y = $92 (status = function not
    implemented by handler).

    ICDNOZ [0021] = device number (1-4, for multiple device handlers).
                    ICBALZ/ICBAHZ [0024/0025] = address of
                            device/filename specification.
                    ICAX1Z/ICAX2Z
                    [002A/002B] = device-specific information.

The handler gets device status information from the device
controller and puts the status bytes in DVSTAT [02EA] through
DVSTAT+3, and finally returns to CIO with the operation status
in register Y.

The IOCB need not be opened nor closed in order for you
to request CIO to perform a GET STATUS operation; the handler
must check where there are restrictions. See Section 5 for a
discussion of the CIO actions involved with a GET STATUS
operation using both open and closed IOCB's, and note the impact
of this operation on the use of the buffer address parameter.

    SPECIAL

This handler entry is used to support all functions not handled
by the other entry points, such as diskette file RENAME, display
DRAW, etc. Specifically, if the IOCB command byte value is
greater than $0D, then CIO will use the SPECIAL entry point. The
handler must interrogate the command byte to determine if the
requested operation is supported.

At handler entry, the following parameters can be of interest:

    X = index to originating IOCB.
    Y = $92 (status = function not implemented by handler).

    ICDNOZ [0021] = device number (1-4, for multiple device
                        handlers).
    ICCOMZ [0022] = command byte.
    ICBALZ/ICBALH [0024/0025] = buffer address.
    ICBLLZ/ICBLHZ [0028/0029] = buffer length.
    ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler will perform the indicated operation, if possible,
and return to CIO with the operation status in register Y.

The IOCB need not be opened nor closed in order for you
to request CIO to perform a SPECIAL operation; the handler
must check where there are restrictions. See Section 5 for a
discussion of the CIO actions involved with a SPECIAL
operation using both open and closed IOCB's, and note the
impact of this on the use of the buffer address parameter.

Error Handling

Error handling has been simplified somewhat by having CIO handle
outer level errors and having SIO handle Serial bus errors,
leaving the handler to process the remaining errors.   These
errors include:

     out-of-range parameters.
     [BREAK] key abort.
     Invalid command.
     Read after end of file.

The current handlers respond to errors using the following
guidelines:

     They keep the recovery simple (and therefore predictable and
     repeatable).

     They Do not interact directly with you for recovery
     instructions.

     They lose as little data as possible.

     They make all attempts to maintain the integrity of file
     oriented device storage -- this involves the initial design
     of the structural elements as well as error recovery
     techniques.

Resource Allocation

Nonresident handlers needing code and/or data space in RAM should
use the techniques listed below, to assure nonconflict with other
parts of the OS, including other nonresident handlers.

Zero-Page RAM

Zero-page RAM has no spare bytes, and even if there were, there
is no allocation scheme to support multiple program assignment of
the spares. Therefore, the nonresident handler must save and
restore the bytes of zero-page RAM it is going to use. The bytes
to use must be chosen carefully, according to the following
criteria:

The bytes cannot be accessed by an interrupt routine.

The bytes cannot be accessed by any noninterrupt code
between the time the handler modifies the bytes and then
restores the original values.

A simple save/restore technique would utilize the stack in a
manner similar to that shown below:

```
            LDA     COLCRS          ; (for example)
            PHA                     ; SAVE ON STACK.
            LDA     COLCRS+1
            PHA


            LDA     HPOINT          ; HANDLER'S POINTER.
            STA     COLCRS
            LDA     HPOINT+1
            STA     COLCRS+1


            XXX     (COLCRS),Y      ; DO YOUR POINTER THING.


            PLA                     ; RESTORE OLD DATA.
            STA     COLCRS+1
            PLA
            STA     COLCRS
```

Note that the Display Handler or Screen Editor should not be
called before restoring the original value of COLCRS, because
COLCRS is a variable used by those routines.

Nonzero-Page RAM

There is no allocation scheme to support the assignment of
fixed regions of nonzero-page RAM to any specific process, so the
handler has three choices:

1. Make a dynamic allocation at initialization time by
   altering MEMLO [02E7].

2. Include the variables with the handler for RAM-resident
   handlers. This still involves altering MEMLO at the time
   the handler is booted.

3. If the handler replaces one of the resident handlers (by
   removing the resident handler's entry in the device
   table), then the new handler can use any RAM that the

OPERATING SYSTEM C016555 -- Section 9

formerly resident handler would have used.

Stack Space

In most cases, there are no restrictions on the use of the stack
by a handler. However, if the handler plans to push more than a
couple dozen bytes to the stack; then it should do a stack
overflow test, and always leave stack space for interrupt
processing.

HANDLER/SIO INTERFACE

This section describes the interface between serial bus device
handlers and the serial bus I/O utility (SIO). SIO completely
handles all bus transactions following the device-independent bus
protocol. SIO is responsible for the following functions:

   Bus data format and timing from computer end.

   Error detection, retries and statuses.

   Bus timeout.

   Transfer of data between the bus and the caller's buffer.

Calling Mechanism

SIO has a single entry point SIOV [E459] for all operations. The
device control block (DCB) [0300] contains all parameters passed
to SIO. The DCB contains the following bytes:

   DEVICE BUS ID -- DDEVIC [0300]

The bus ID of the device is set by the handler prior to calling
SIO (see Appendix I).

   DEVICE UNIT # -- DUNIT [0301]

This byte indicates that of n units of a given device type to
access, and is set by the handler prior to calling SIO. This
value usually comes from ICDNOZ. SIO accesses the bus device
whose address is equal to the value of DDEVIC plus DUNIT minus 1
(the lowest unit number is normally equal to 1).

   DEVICE COMMAND -- DCOMND [0302]

The handler sets this byte prior to calling SIO. It will be sent
to the bus device as part of the command frame. See Appendix I
for device command byte values.

DEVICE STATUS -- DSTATS [0303]

This byte is bidirectional. The handler will use DSTATS to
indicate to SIO what to do after the command frame is sent and
acknowledged. SIO will use it to indicate to the handler the
status of the requested operation.

Prior to an SIO call:

```
      7           0
     +-+-+-+-+-+-+-+-+
     |W|R|  unused   |
     +-+-+-+-+-+-+-+-+
```

Where: W,R = 0,0 indicates no data transfer is associated with the
              operation.
        0,1 indicates a data frame is expected from the device.
        1,0 indicates a data frame is to be sent to the device.
        1,1 is invalid.

After an SIO call:

```
      7           0
     +-+-+-+-+-+-+-+-+
     |  status code  |
     +-+-+-+-+-+-+-+-+
```

See Appendix C for the possible SIO operation status codes.

HANDLER BUFFER ADDRESS -- DBUFLO/DBUFHI [0304/0305]

The handler sets this 2-byte pointer. It indicates the source
or destination buffer for device data or status information.

DEVICE TIMEOUT -- DTIMLO [0306]

The handler sets this byte. It specifies the device timeout time
in units of 64/60 of a second. For example, a count of 6
specifies a timeout of 6.4 seconds.

BUFFER LENGTH/BYTE COUNT -- DBYTLO/DBYTHI [0308/0309]

The handler sets this 2-byte count for the current
operation, and indicates the number of data bytes to be
transferred into or out of the buffer. This parameter is not
required if the STATUS byte W and R bits are both zero. These
values indicate that no data transfer is to take place.

    WARNING: There is a bug in SIO that causes incorrect
    actions when the last byte of a buffer is in a memory
    address ending in $FF, such as 13FF, 42FF, etc.

The handler sets these 2-bytes. The SIO includes them in the bus
command frame; they have device-specific meanings.

Functions Supported

SIO does not examine the COMMAND byte it sends to the device,
because all bus transactions are expected to conform to a
universal protocol. The protocol includes three forms, stated
below (as seen from the computer):

> Send command frame.

> Send command frame and send data frame.

> Send command frame and receive data frame.

The values of the W and R bits in the status byte select the
command form.

Error Handling

SIO handles most of the serial bus errors for the handler,
as indicated below:

Bus timeout -- SIO provides a uniform command frame and data
frame ACK byte timeout of 1/60 of a second - 0 / + 1/60.
The handler specifies the maximum COMPLETE byte timeout
value in DTIMLO.

Bus errors -- SIO detects and reports UART overrun and
framing errors. The sensing of these errors in any received
byte will cause the entire associated frame to be considered
bad.

Data frame checksum error -- SIO validates the checksum on
all received data frames and generates a checksum for all
transmitted frames.

Invalid response from device -- In addition to the error
conditions stated above, SIO checks that the ACK and
COMPLETE responses are proper (ACK = $41 and COMPLETE =
$43). ACK stands for acknowledge.

Bus operation retries -- SIO will attempt one complete command
retry if the first attempt is not error free, where a complete
command try consists of up to 14 attempts to send (and
acknowledge) a command frame, followed by a single attempt to

receive the COMPLETE code and possibly a data frame.

> NOTE: There is a bug in the retry logic for data writes,
> such that if the command frame is acknowledged by the
> controller, but the data frame is not acknowledged, then SIO
> will retry indefinitely.

Unified error status codes -- SIO provides device-independent error
codes (see Appendix C).

SERIAL I/O BUS CHARACTERISTICS AND PROTOCOL

This section describes:

o   The electrical characteristics of the ATARI 400
     and ATARI 800 Home Computers serial bus

o   The use of the bus to send bytes of data,

o   The organization of the bytes as "frames" (records),

o   The overall command sequences that utilize frames
     and response bytes to provide computer/peripheral communication.

Hardware/Electrical Characteristics

The ATARI 400 and the ATARI 800 Home Computers
communicate with peripheral devices over a 19,200 baud
asynchronous serial port. The serial port consists of a serial
DATA OUT (transmission) line, a serial DATA IN (receiver) line
and other miscellaneous control lines.

Data is transmitted and received as 8 bits of serial data (LSB
sent first) preceded by a logic zero start bit and succeeded
by a logic one stop bit. The serial DATA OUT is transmitted as
positive logic (+4v = one/true/high, Ov = zero/false/low). The
serial DATA OUT line always assumes its new state when the
serial CLOCK OUT line goes high; CLOCK OUT then goes low in
the center of the DATA OUT bit time.

An end view of the Serial bus connector at the computer or
peripheral is shown below (the cable connectors would of
course be a mirror image):

```
    2   4   6   8  10  12
    o   o   o   o   o   o

  o   o   o   o   o   o   o
  1   3   5   7   9  11  13
```

where: 1 = computer CLOCK IN.
         2 = computer CLOCK OUT.
         3 = computer DATA IN.
         4 = GND.
         5 = computer DATA OUT.
         6 = GND.
         7 = COMMAND-.
         8 = MOTOR CONTROL.
         9 = PROCEED-.
        10 = +5v/READY.
        11 = computer AUDIO IN.
        12 = +12v.
        13 = INTERRUPT-.

Figure 9-4   Serial Bus Connector Pin Descriptions

CLOCK IN is not used by the present OS and peripherals. This
line can be used in future synchronous communications schemes.

CLOCK OUT is the serial bus clock. CLOCK OUT goes high at the
start of each DATA OUT bit and returns to low in the middle of
each bit.

DATA IN is the serial bus data line to the computer.

Pin 4 GND is the signal/shield ground line.

DATA OUT is the serial bus data line from the computer.

Pin 6 GND is the signal/shield ground line.

COMMAND- is normally high and goes low when a command frame is
being sent from the computer.

MOTOR CONTROL is the cassette motor control line (high=on,
low= off).

PROCEED- is not used by the present OS and peripherals; this line
is pulled high.

+5v/READY indicates that the computer is turned on and ready. This
line can also be used as a +5 volt supply of 50ma current rating
for ATARI peripherals only.

AUDIO IN accepts an audio signal from the cassette.

+12V is a +12 volt supply of unknown current rating for ATARI
peripherals only.

INTERRUPT- is not used by the present OS and peripherals; this
line is pulled high.

There are no pin reassignments made in the Serial bus cable,
so pin 3, the computer's DATA IN line, is the peripheral's
data output line; and similarly for pin 5.

Serial Port Electrical Specifications

Peripheral input:

    V1H = 2.0v min.
    V1L = 0.4v max.

    I1H = 20ua. max. @ V1H = 2.0v
    I1L = 5ua. max. @ V1L = .4v

Peripheral output (open collector bipolar):

    VOL = 0.4v max. @ 1.6 ma.
    VOH = 4.5v min. with external 100Kohm pull-up.

Vcc/READY input:

    V1H = 2.0v min. @ I1H = 1ma. max.
    V1L = 0.4v max.
    Input goes to logic zero when open.

Bus Commands

The bus protocol specifies that all commands must originate from the
computer, and that peripherals will present data on the bus only when
commanded to. Every bus operation will go to completion before
another bus operation is initiated (no overlap). An error detected at
any point in the command sequence will abort the entire sequence.

A bus operation consists of the following elements:

    Command frame from the computer.

    Acknowledgement (ACK) from the peripheral.

    Optional data frame to or from the computer.

    Operation complete (COMPLETE) from the peripheral.

Command Frame

The serial bus protocol provides for three types of commands: 1) data
send, 2) data receive and 3) immediate (no data -- command only).
There is a common element in all three types, a command frame
consisting of five bytes of information sent from the computer
while the COMMAND- line is held low. The format of the command
frame is
shown below:

```
+------------------+
|    device ID     |
+------------------+
|     command      |
+------------------+
|  auxiliary #1    |
+------------------+
|  auxiliary #2    |
+------------------+
|     checksum     |
+------------------+
```
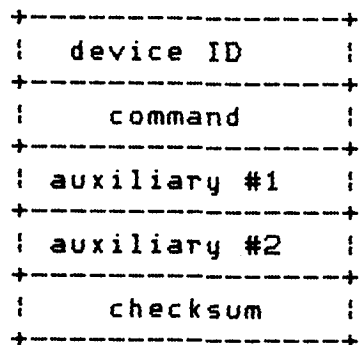
Figure 9-5   Serial Bus Command Frame Format

The device ID specifies that of the serial bus devices is being
addressed (see Appendix I for a list of device IDs).

The command byte contains a device-dependent command (see
Appendix I for a list of device commands).

The auxiliary bytes contain more device-dependent information.

The checksum byte contains the arithmetic sum of the first four
bytes (with the carry added back after every addition).

   Command Frame Acknowledge

The peripheral being addressed would normally respond to a
command frame by sending an ACK byte ($41) to the computer; if
there is a problem with the command frame, the peripheral should
not respond.

   Data Frame

Following the command frame (and ACK) can be an optional data
frame that is formatted as shown below:

```
          +----------------+
          !                !
          !                !
          !     data       !
          =                =
          !     bytes      !
          !                !
          !                !
          +----------------+
          !    checksum    !
          +----------------+
```
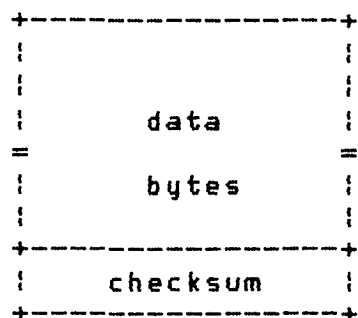
This data frame can originate at the computer or at the device
controller, depending upon the command. Current device
controllers expect fixed-length data frames as does the computer,
where the data frame length is a fixed function of the device
type and command.

The checksum value in the data frame is the arithmetic sum of all
of the frame data preceding the checksum, with the carry from
each addition being added back (the same as for the command
frame).

In the case of the computer sending a data frame to a peripheral,
the peripheral is expected to send an ACK if the data frame is
acceptable, and send a NAK ($4E), or do nothing if the data frame
is unacceptable. See the first flowchart in Section 9.

   Operation Complete

A peripheral is also expected to send an operation-COMPLETE byte
($43) at the time the commanded operation is complete. The
location of this byte in the command sequence for each command
type is shown in the timing diagrams in Section 9. If the
operation cannot go to normal, error-free completion, the
peripheral should respond with an ERROR byte ($45) instead of
COMPLETE.

Bus Timing

This section provides timing diagrams for the three types of
command sequences: data send, data receive, and immediate.

DATA SEND sequence:

```
            ---+           +--------------------------------
COMMAND-       !           !
               +--------+
               +------+           +---//---+
DATA OUT       ! cmnd !           !  data  !
            ---+frame +------//--+ frame  +---------------
                           +-+                 +-+       +-+
                           ! !                 ! !       ! !
DATA IN     ---------------+ +---------------+ +--//--+ +-
                           ACK                 ACK       CMPL

               !!        !!  ! !      !         !   ! !     !

               t0        t1 t2      t3              t4      t5
```

DATA RECEIVE sequence:

```
            ---+           +--------------------------------
COMMAND-       !           !
               +--------+

               +------+
DATA OUT       ! cmnd !
            ---+frame +---------------------------------
                           +-+     +-+ +----//----+
                           ! !     ! ! !  data    !
DATA IN     ---------------+ +--//--+ +-+ frame   +--------
                           ACK       CMPL

               !!        !!  ! !      !
               t0        t1 t2      t5
```

IMMEDIATE sequence:

```
              ---+          +-----------------------------------
COMMAND-         ! !        !
              +--------+

              +------+
DATA OUT      ! cmnd !
           ----+frame +----------------------------------------

                            +-+                    +-+
DATA IN                     ! !                    ! !
           --------------+ +-----------//-----+ +----------
                            ACK                    CMPL

              !!       !! ! !                    !

              t0       t1 t2              t5
```
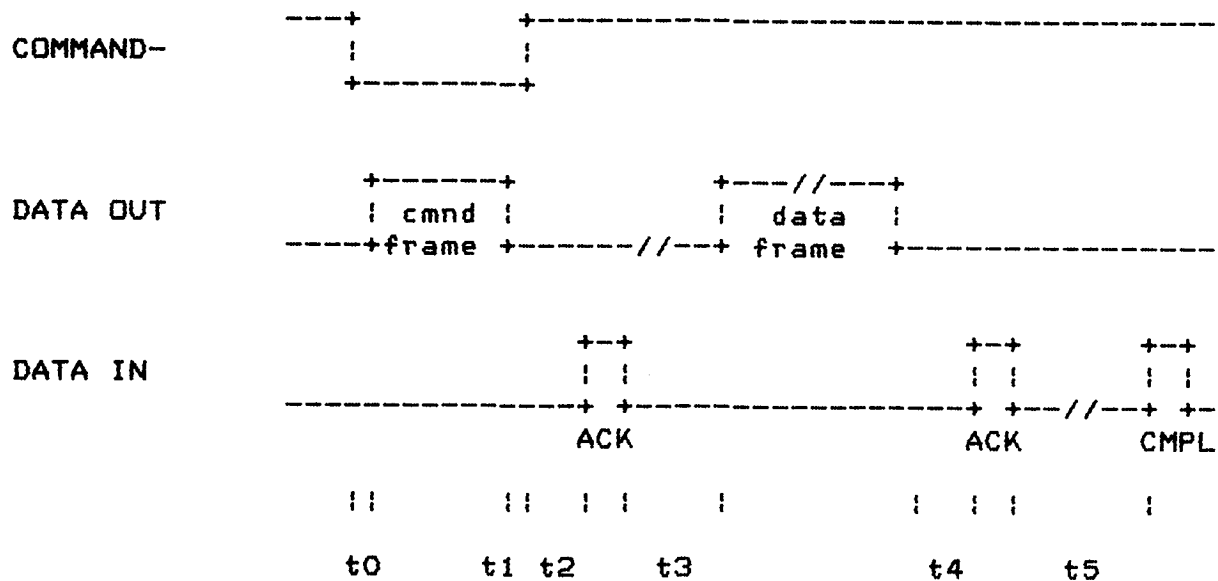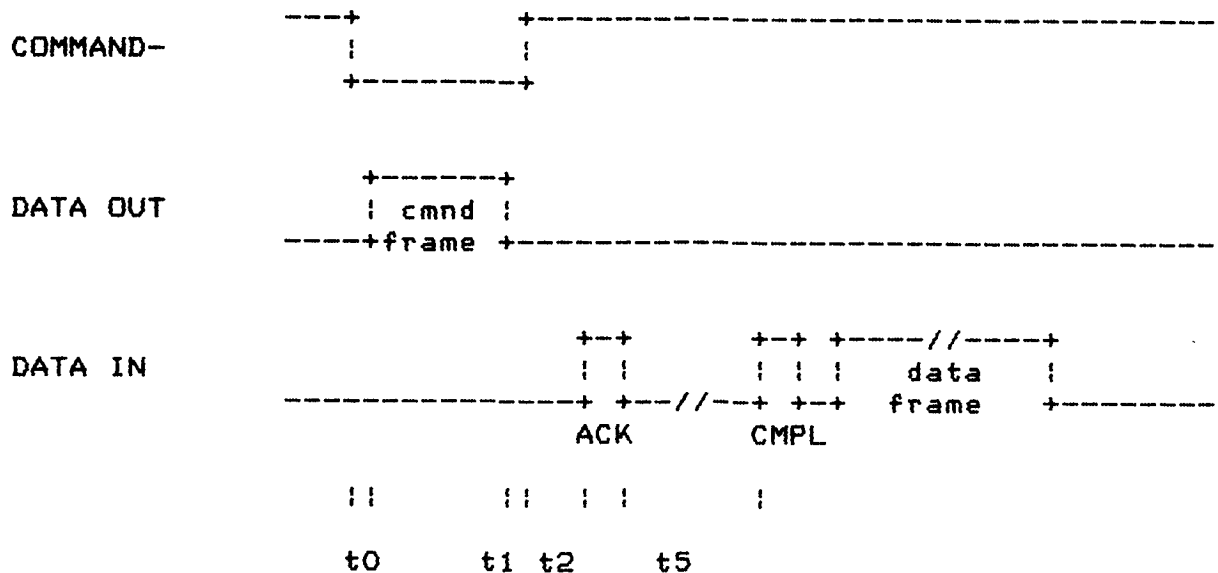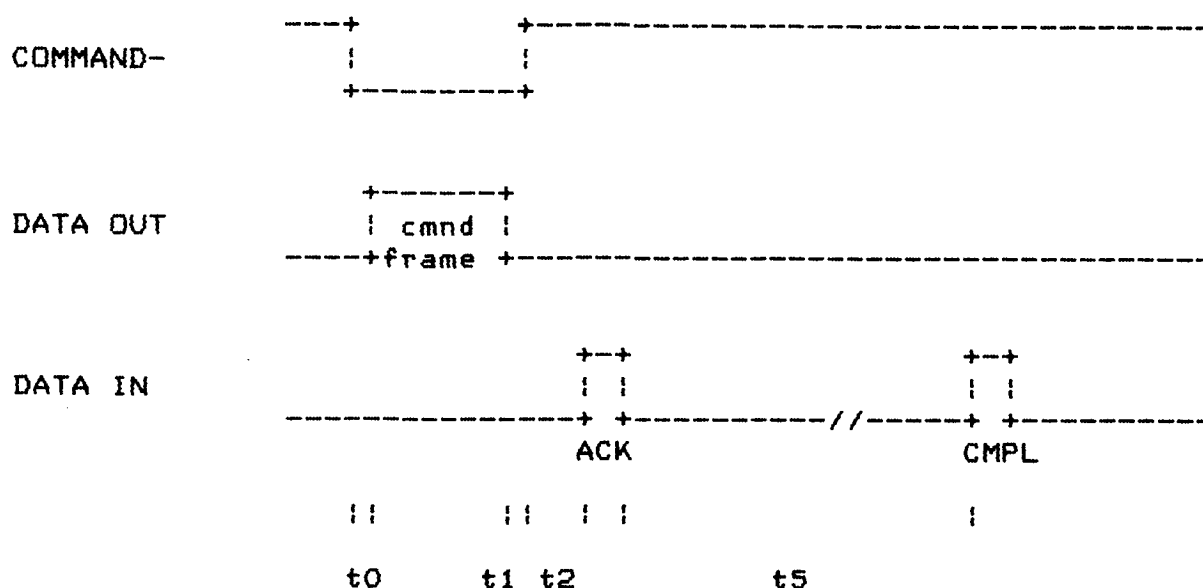
Figure 9-6   Serial Bus Timing Diagram

The computer generates a delay (t0) between the lowering of COMMAND-
and the transmission of the first byte of the command frame.

     computer t0 (min) = 750 microsec.
     computer t0 (max) = 1600 microsec.

     peripheral t0 (min) = ??
     peripheral t0 (max) = ??

The computer generates a delay (t1) between the transmission of
the last bit of the command frame and the raising of the COMMAND-
line.

     computer t1 (min) = 650 microsec.
     computer t1 (max) = 950 microsec.

     peripheral t1 (min) = ??
     peripheral t1 (max) = ??

The peripheral generates a delay (t2) between the raising of
COMMAND- and the transmission of the ACK byte by the peripheral.

     computer t2 (min) = 0 microsec.
     computer t2 (max) = 16 msec.

     peripheral t2 (min) = ??
     peripheral t2 (max) = ??

The computer generates a delay (t3) between the receipt of the
last bit of the ACK byte and the transmission of the first bit of
the data frame by the computer.

    computer t3 (min) = 1000 microsec.
    computer t3 (max) = 1800 microsec.

    peripheral t3 (min) = ??
    peripheral t3 (max) = ??

The peripheral generates a delay (t4) between the transmission of
the last bit of the data frame and the receipt of the first bit
of the ACK byte by the computer.

    computer t4 (min) = 850 microsec.
    computer t4 (max) = 16 msec.

    peripheral t4 (min) = ??
    peripheral t4 (max) = ??

The Peripheral generates a delay (t5) between the the receipt of
the last bit of the ACK byte and the first bit of the COMPLETE
byte by the computer.

    computer t5 (min) = 250 microsec.
    computer t5 (max) = 255 sec. (handler-dependent)

    peripheral t5 (min) = ??
    peripheral t5 (max) = N/A

HANDLER ENVIRONMENT

Nonresident handlers can be installed in at least three different
manners:

    1. As booted software from diskette or cassette.

    2. Resident in a cartridge (A or B).

    3. Downloaded from a serial bus device.

This section will discuss the basic mechanisms for handler
installation for these environments. In order to fully utilize the
information in this section, you must have read and understood the
following sections:

Bootable Handler

The diskette- or cassette-booted software will insert the
handler's vector table pointer and name to the device table
whenever the booted software's initialization entry point is
entered (on power-up and system reset). Remember that both
power-up and system reset clear the device table of all but the
resident handler entries.

Cartridge Resident Handler

The cartridge software will insert the handler's vector table
pointer and name to the device table whenever the cartridge's
initialization entry point is entered (on power-up and
system reset). Remember that both power-up and system reset
clear the device table of all but the resident handler entries;
therefore the device table must be reestablished by the
handler-initialization procedure upon every entry.

FLOWCHARTS

The following pages contain process flowcharts showing the SIO
and peripheral actions for the Serial bus command forms.

PERIPHERAL'S COMMAND FRAME PROCESSING
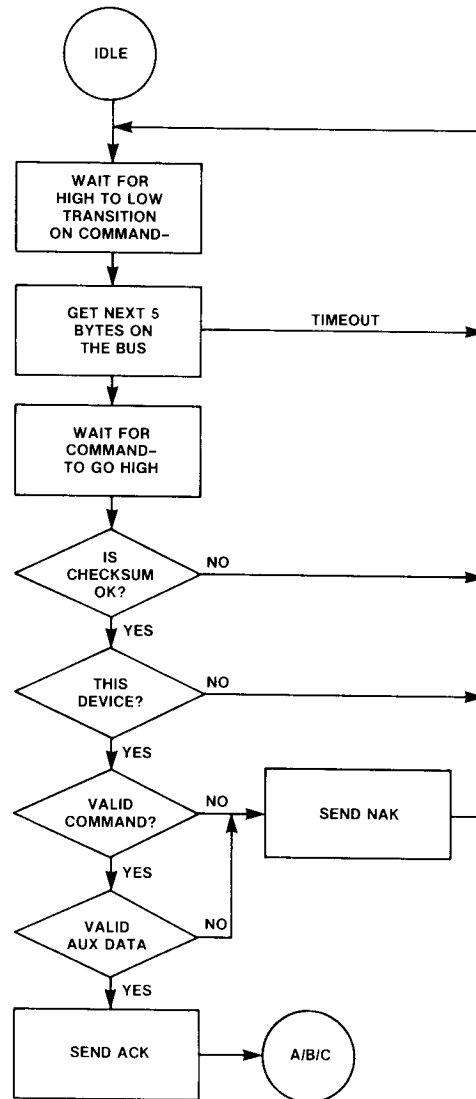
DATA FRAME TO PERIPHERAL

OPERATING SYSTEM CO16555 -- Section 9
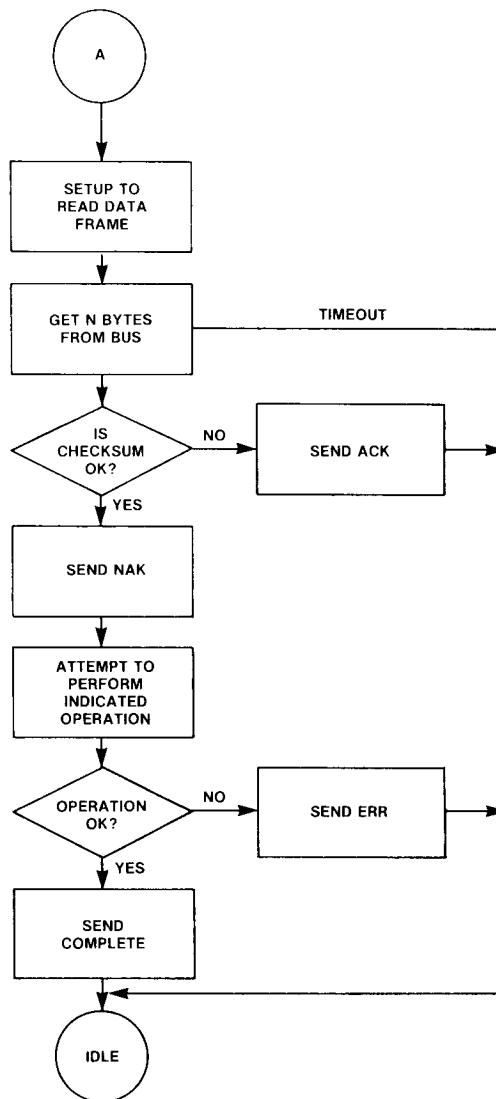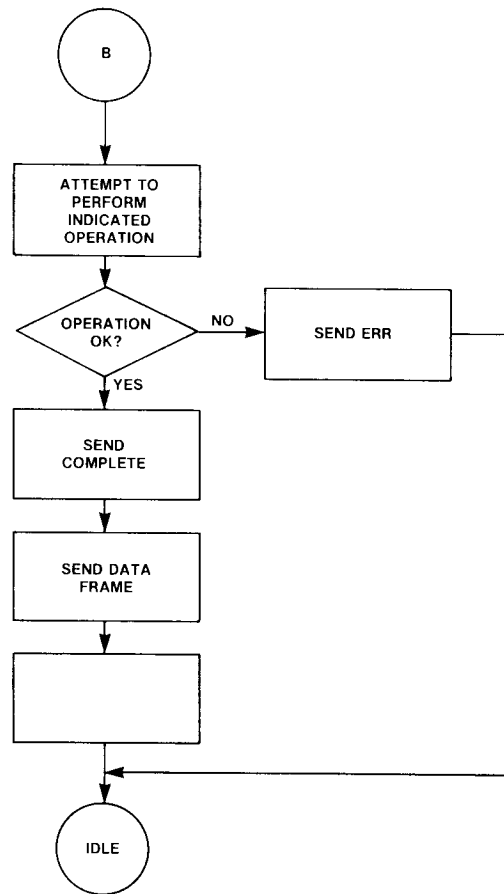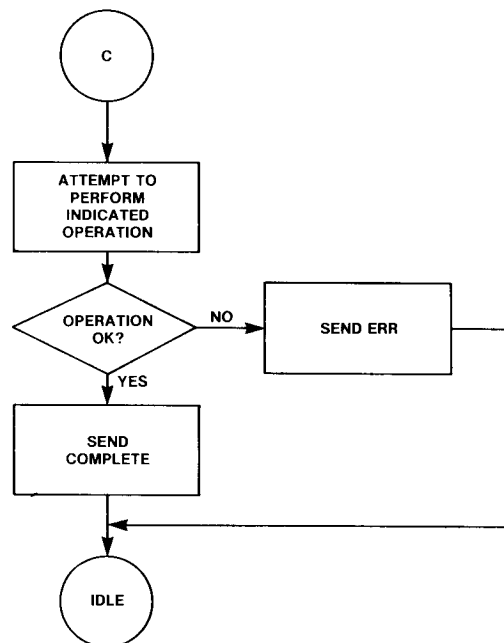
10   PROGRAM ENVIRONMENT AND INITIALIZATION

This section discusses possible alternative software environments
using OS Configurations. Environments other than those discussed
here are also possible. A thorough understanding of the power-up
and system reset processes (see Section 7) will be necessary to
evaluate all alternative environments.

CARTRIDGE

Most games (and some language processors) are supported via the
cartridge environment. The cartridge resident software is in
control of the system, sometimes using the OS and sometimes not.
A cartridge can specify whether the diskette is to be booted at
power-up time, whether the cartridge is to provide the
controlling software, or whether the cartridge is a special
diagnostic cartridge. These options are specified by bits in the
cartridge header, as shown below:

```
+-----------------+
!    cartridge    !        BFFA (9FFA for cartridge B)
+-             -+
! start address   !
+-----------------+
!       00        !
+-----------------+
!   option byte   !
+-----------------+
!    cartridge    !
+-             -+
!  init address   !        BFFF (9FFF for cartridge B)
+-----------------+
```
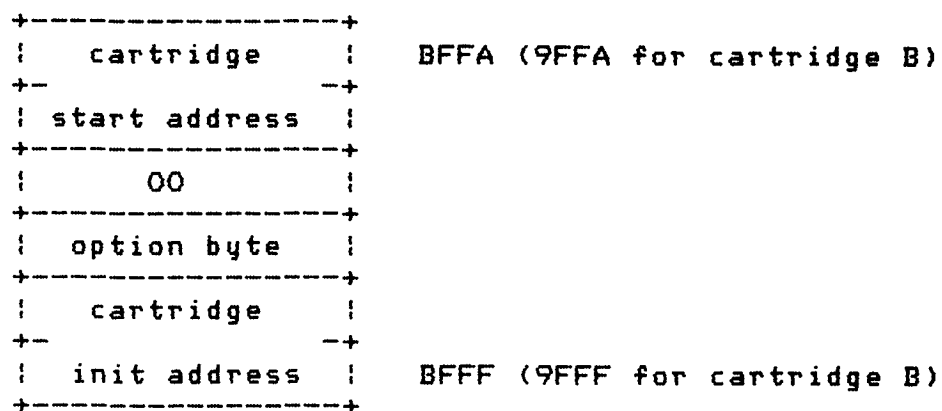
Figure 10-1   Cartridge Header Format

The byte of "00" is used to allow the OS to determine when a
cartridge is inserted; locations BFFC and 9FFC will not read zero
when there is neither RAM at those locations nor a cartridge
inserted. RAM is differentiated from a cartridge by its ability
to be altered.

The option byte has the following option bits:

        bit 0 = 0, then do not boot the diskette.
               1, then boot the diskette.

        Bit 2 = 0, then init but do not start the cartridge.
               1, then init and start the cartridge.

        bit 7 = 0, then cartridge is not a diagnostic cartridge.
               1, then cartridge is a diagnostic cartridge and control
                   will be given to the cartridge before any of the OS
                   is initialized (JMP (BFFE)).

The cartridge init address specifies the location to which the OS will
JSR during all power-up and system reset operations. As a minimum,
this vector should point to an RTS instruction.

The cartridge start address specifies the location to which the OS
will JMP during all power-up and system reset operations, if
bit 1 of the option byte is = 1. The application should examine
the variable WARMST [0008] if system reset action is to be
different than power-up (WARMST will be zero on power-up and
nonzero thereafter).

Cartridge Without Booted Support Package

A cartridge that does not specify the diskette-boot option and does
not support the cassette-boot possibility can use lower memory
(from 0480 to the address in MEMTOP [02E5]) in any way it sees
fit.

Cartridge With Booted Support Package

A cartridge that does specify the diskette-boot option or does
support the cassette-boot possibility must use some care in its
use of lower memory. The following regions are defined:

        0480-06FF is always available to the cartridge.
        MEMLO/MEMTOP region is always available to the cartridge.

DISKETTE-BOOTED SOFTWARE

Software can be booted from the disk drive at power-up time in
response to one of the following conditions:

Neither Cartridge A nor B is inserted.

Cartridge A is inserted and has bit 0 of its option byte [BFFD] = 1.

Cartridge B is inserted and has bit 0 of its option byte [9FFD] = 1.

If any of these conditions are met, the OS will attempt to read the boot record from sector #1 of disk drive 1 and then transfer control to the software that was read in. The exact sequence of operations will be explained later in this section.

Diskette-Boot File Format

The key region of a diskette-boot file is the first six bytes, which are formatted as shown below:

```
+------------------+
|      flags       |  first byte
+------------------+
|   # of sectors   |
+------------------+
| memory address   |
+-              -+
| to start load    |
+------------------+
|      init        |
+-              -+
|     address      |  sixth byte
+------------------+
|      boot        |
|  continuation    |
|      code        |
```

Figure 10-2   Diskette-Boot File Format

The first byte is stored in DFLAGS [0240], but is otherwise unused. It should equal zero.

The second byte contains the number of 128-byte diskette sectors to be read as part of the boot process (including the record containing this information). This number can range from 1 to 255, with 0 meaning 256.

The third and fourth bytes contain the address (lo,hi) at which to
start loading the first byte of the file.

The fifth and sixth bytes contain the address (lo,hi) to which the
booter will transfer control after the boot process is complete
and whenever the [SYSTEM.RESET] key is pressed.

The Diskette File Management System (FMS) has extra bytes assigned to
its boot record, but this is a special case of the generalized
diskette-boot and is discussed in Section 5.

Diskette-Boot Process

If no cartridge is installed, then the diskette will follow these
steps to boot up:

1. Read the first diskette record to the cassette buffer [0400].

2. Extract information from the first six bytes:

    Save the flags byte to DFLAGS [0240,1]. Save the # of sectors
    to boot to DBSECT [0241,1]. Save the load address to BOOTAD
    [0242,2]. Save the initialization address in DOSINI [000C,2].

3. Move the record just read to the load address specified.

4. Read the remaining records directly to the load area.

5. JSR to the load address+6 where a multistage boot process can
    continue. The carry bit indicates the success of this
    operation (carry set = error, carry reset = success).

    NOTE: During step 5, after the initial boot process is
    complete, the booter will transfer control to the seventh byte
    of the first record. The software should continue the boot
    process at this point, if it is a multistage boot. The value
    of MEMLO [02E7] should point to the first free RAM location
    beyond the software just booted. It should be established by
    the booted software as shown below:

```
                LDA     #END+1          ; SET UP LSB.
                STA     MEMLO
                STA     APPMHI
                LDA     #END+1/256      ; SET UP MSB.
                STA     MEMLO+1
                STA     APPMHI+1
```

    If the booted software is to take control of the
    system at the end of the boot operation, the
    vector DOSVEC [000A] must be set up by the
    application at this time; DOSVEC points to the

restart entry for the booted application. If the
booted software is not to take control, then
DOSVEC should remain unchanged.

```
        LDA     #RESTRT             ; RESTART LSB.
        STA     DOSVEC
        LDA     #RESTRT/256
        STA     DOSVEC+1
```

6. JSR indirectly through DOSINI for initialization of the
    application; the application will initialize and return.

    NOTE: The OS enters the initialization point on every
    system reset and power-up. Internal initialization can take
    place during system reset and power-up as well. Initialization
    can also be deferred until Step 7 for controlling
    applications.

7. JMP indirectly through DOSVEC to transfer control to the
    application.

    NOTE: Pressing the [SYSTEM.RESET] key after the application
    is fully booted will cause steps 6 and 7 to be repeated.

Sample Diskette-Bootable Program Listing

This skeletal program can be booted from the diskette. It retains
control when it is entered.


```
; THIS IS THE START OF THE PROGRAM FILE.

PST=    $0700                   ; (OR SOME OTHER LOCATION).
*=      PST                     ; (.ORG).


; THIS IS THE diskette-boot CONTROL INFORMATION.

        .BYTE   0               ;
        .BYTE   PND-PST+127/128 ; NUMBER OF RECORDS.
        .WORD   PST             ; MEMORY ADDRESS TO START LOAD.
        .WORD   PINIT           ; PROGRAM INIT.
```

; THIS IS THE START OF THE BOOT CONTINUATION.

```
        LDA     #PND             ; ESTABLISH LOW MEMORY LIMITS.
        STA     MEMLO
        STA     APPMHI
        LDA     #PND/256
        STA     MEMLO+1
        STA     APPMHI+1

        LDA     #RESTRT          ; ESTABLISH RESTART VECTOR.
        STA     DOSVEC
        LDA     #RESTRT/256
        STA     DOSVEC+1

        CLC                      ; SET FLAG FOR SUCCESSFUL BOOT.
        RTS
```

; APPLICATION INITIALIZATION ENTRY POINT.

```
PINIT   RTS                      ; NOTHING TO DO HERE FOR ...
                                 ; ... CONTROLLING APPLICATION.
```

; THE MAIN BODY OF THE PROGRAM FOLLOWS.

RESTRT=*

; THE MAIN BODY OF THE PROGRAM ENDS HERE.

```
PND=    *                        ; 'PND' = NEXT FREE LOCATION.
        .END
```

Figure 10-3   Diskette-Bootable Program Listing Example

Program to Create Diskette-Boot Files

This section provides a program that can be used to make bootable
files on diskettes. The program given is not the only one possible,
and no claims are made as to its elegance.

Shown below is a listing of the program to create diskette-boot files.

```
; THIS PROGRAM WRITES A SINGLE "FILE" TO THE DISKETTE AND IS
; USED IN CONJUNCTION WITH A PROCEDURE TO MAKE DISKETTE-
; BOOTABLE FILES. THE FOLLOWING TWO SYMBOLS MUST BE EQUATED
; USING THE MEMORY LIMITS OF THE PROGRAM TO BE COPIED:
;
;       'PST' = PROGRAM START ADDRESS (SEE SAMPLE PROGRAM).
;       'PND' = PROGRAM END ADDRESS (SEE SAMPLE PROGRAM).


SECSIZ=128                      ; DISKETTE SECTOR SIZE.
PST=    $0700
PND=    $1324
FLEN=   PND-PST+SECSIZ-1/SECSIZ ; # OF SECTORS IN FILE.

*=      $B000                   ; THIS PROGRAM'S ORIGIN.




BOOTB   BRK                     ; *** LOAD APPLICATION ***

; SET UP DEVICE CONTROL BLOCK FOR DISKETTE HANDLER CALL

        LDA     #FLEN           ; # OF SECTORS TO WRITE.
        STA     COUNT

        LDA     #1              ; DISK DRIVE #1.
        STA     DUNIT

        LDA     #'W             ; SET UP FOR WRITE WITH CHECK.
        STA     DCOMND

        LDA     #PST            ; POINT TO START OF APPLIC. PROG.
        STA     DBUFLO
        LDA     #PST/256
        STA     DBUFHI

        LDA     #01             ; SET UP STARTING SECTOR # = 0001.
        STA     DAUX1
        LDA     #00
        STA     DAUX2
```

; NOW WRITE THE FILE ONE SECTOR AT A TIME.

```
BOTO10 JSR     DSKINV          ; WRITE ONE SECTOR.
       BMI     DERR            ; ERROR.

       LDA     DBUFLO          ; INCREMENT MEMORY ADDRESS.
       CLC
       ADC     #SECSIZ
       STA     DBUFLO
       LDA     DBUFHI
       ADC     #0
       STA     DBUFHI

       INC     DAUX1           ; INCREMENT SECTOR #.
       BNE     BOTO20
       INC     DAUX2

BOTO20 DEC     COUNT           ; MORE SECTORS TO WRITE?
       BNE     BOTO10          ; YES.

       BRK                     ; STOP WHEN DONE.

DERR   BRK                     ; STOP ON ERROR.

COUNT  *=*+1                   ; SECTOR COUNT.

; THIS IS THE CARTRIDGE HEADER

*=     $BFF9                   ; "A" CARTRIDGE.

INIT   RTS
       .WORD   BOOTB
       .BYTE   0,4
       .WORD   INIT

       .END
```

CASSETTE-BOOTED SOFTWARE

You can boot software from the cassette as well as from the
diskette, at power-up. The following requirements must be met in order
to boot from the cassette:

o   You must be pressing the [START] key as power is
     applied to the system.

o   A cassette tape with a proper boot format file must be
     installed in the cassette drive, and the PLAY button must be
     pressed.

o   When you are given the audio prompt by the cassette
     handler you must press the [RETURN] key.

If all of these conditions are met, the OS will read the boot file
from the cassette and then transfer control to the software that
was read in. The exact sequence of operations will be explained
later in this section.

Cassette-Boot File Format

The key region of a cassette-boot file is the first six bytes, that
are formatted as shown below:


```
              +------------------+
              |                  |
              +------------------+
              | # of Records     |
              +------------------+
              | Memory Address   |
              +--            --+
              | To Start Load    |
              +------------------+
              |      Init        |
              +--            --+
              |    address       |
              +------------------+
```


The first byte is not used by the cassette-boot process.

The second byte contains the number of 128-byte cassette records to
be read as part of the boot process (including the record
containing this information). This number can range from 1 to 255,
with 0 meaning 256.

The third and fourth bytes contain the address (lo,hi) to which the
booter will transfer control after the boot process is complete and
whenever the [SYSTEM.RESET] key is pressed.

Cassette-Boot Process

The cassette-boot process is described step-by-step for a
configuration in that no cartridge is installed and no diskettes are
attached. For the general case see Section 7.

    1. Read the first cassette record to the cassette buffer.

    2. Extract information from the first six bytes:

Save the # of records to boot. Save the load address. Save
the initialization address in CASINI [0002]

3. Move the record just read to the load address specified.

4. Read the remaining records directly to the load area.

5. JSR to the load address+6 where a multistage boot process
can continue; the carry bit will indicate the success of
this operation (carry set=error, carry reset=success).

6. JSR indirectly through CASINI for initialization of the
application; the application will initialize and return.

7. JMP indirectly through DOSVEC to transfer control to the
application.

Pressing the [SYSTEM.RESET] key after the application is fully booted
will cause steps 6 and 7 to be repeated.

NOTE: After the initial boot process is complete, the booter will
transfer control to the seventh byte of the first record; at this
point the software should continue the boot process (if it is a
multistage boot) and then stop the cassette drive, which due to a
system bug will still be running, using the following instruction
sequence:

```
        LDA       #$3C
        STA       PACTL [D302]
```
The application should then set a value in MEMLO [0237] that
points to the first free RAM location beyond the software just
booted, as shown below:

```
        LDA       #END+1
        STA       MEMLO
        STA       APPMHI
        LDA       #END+1/256
        STA       MEMLO+1
        STA       APPMHI+1
```

If the booted software is to take control of the system at the end
of the boot operation, the vector DOSVEC [000A] must be set up by
the application at this time; DOSVEC points to the restart entry
for the booted application. If the booted software is not to take
control, then DOSVEC should remain unchanged.

```
    LDA     #RESTRT              ; RESTART LSB
    STA     DOSVEC
    LDA     #RESTRT/256
    STA     DOSVEC+1
```

NOTE: The initialization point is entered on every system reset
and power-up; internal initialization can take place here.

For controlling applications initialization can also be deferred
until step 7.

Sample Cassette-Bootable Program Listing

Shown below is a skeletal program that can be booted from the
cassette and that retains control when it is entered.

```
; THIS IS THE START OF THE PROGRAM FILE.

PST=      $0700                ; (OR SOME OTHER LOCATION).
*=        PST                  ; (.ORG).


; THIS IS THE cassette-boot CONTROL INFORMATION.

          .BYTE   0             ; (DOESN'T MATTER).
          .BYTE   PND-PST+127/128 ; NUMBER OF RECORDS.
          .WORD   PST           ; MEMORY ADDRESS TO START LOAD.
          .WORD   PINIT         ; PROGRAM INIT.

; THIS IS THE START OF THE BOOT CONTINUATION.

          LDA     #$3C          ; STOP THE CASSETTE.
          STA     PACTL

          LDA     #PND          ; ESTABLISH LOW MEMORY LIMITS.
          STA     MEMLO
          STA     APPMHI
          LDA     #PND/256
          STA     MEMLO+1
          STA     APPMHI+1

          LDA     #RESTRT       ; ESTABLISH RESTART VECTOR.
          STA     DOSVEC
          LDA     #RESTRT/256
          STA     DOSVEC+1

          CLC                   ; SET FLAG FOR SUCCESSFUL BOOT.
          RTS

; APPLICATION INITIALIZATION ENTRY POINT.

PINIT  RTS                      ; NOTHING TO DO HERE FOR ...
                                ; ... CONTROLLING APPLICATION.

; THE MAIN BODY OF THE PROGRAM FOLLOWS.

RESTRT=*

; THE MAIN BODY OF THE PROGRAM ENDS HERE.
```

```
PND=     *                           ;  'PND' = NEXT FREE LOCATION.
         .END
```

Figure 10-4   Sample Cassette-Bootable Program

Program to Create Cassette-Boot Files

This section provides a program listing that can be used to make
bootable files on cassette tapes. The program given is not the only
one possible, and no claims are made as to its elegance.

Shown below is a listing of the program to create a cassette-boot
file:

```
;   THIS PROGRAM WRITES A SINGLE FILE TO THE CASSETTE AND IS
;   USED IN CONJUNCTION WITH A PROCEDURE TO MAKE CASSETTE-
;   BOOTABLE FILES. THE FOLLOWING TWO SYMBOLS MUST BE EQUATED
;   USING THE MEMORY LIMITS OF THE PROGRAM TO BE COPIED:

;       'PST' = PROGRAM START ADDRESS (SEE SAMPLE PROGRAM).
;       'PND' = PROGRAM END ADDRESS (SEE SAMPLE PROGRAM).

PST=     $0700
PND=     $1324
FLEN=    PND-PST+127/128*128      ; ROUND UP TO MULTIPLE OF 128.

*=       $B000                    ; THIS PROGRAM'S ORIGIN.

BOOTB    LDX     #$10             ; USE IOCB #1.

; FIRST OPEN THE CASSETTE FILE FOR WRITING.

         LDA     #OPEN            ; SET UP FOR DEVICE "OPEN."
         STA     ICCOM, X

         LDA     #OPNOT           ; DIRECTION IS "OUTPUT."
         STA     ICAX1, X
         LDA     #$80             ; SELECT SHORT IRG.
         STA     ICAX2, X

         LDA     #CFILE           ; SET UP POINTER TO DEVICE NAME.
         STA     ICBAL, X
         LDA     #CFILE/256
         STA     ICBAH, X

         JSR     CIOV             ; ATTEMPT TO OPEN FILE.
         BMI     CERR             ; ERROR.

; NOW WRITE THE ENTIRE FILE AS ONE OPERATION.
```

```
        LDA     #PUTCHR         ; SET UP FOR "PUT CHARACTERS. "
        STA     ICCOM, X


        LDA     #PST            ; POINT TO START OF APPLIC. PROG.
        STA     ICBAL, X
        LDA     #PST/256
        STA     ICBAH, X


        LDA     #FLEN           ; SET UP # OF BYTES TO WRITE.
        STA     ICBLL, X
        LDA     #FLEN/256
        STA     ICBLH, X


        JSR     CIOV            ; WRITE ENTIRE FILE.
        BMI     CERR            ; ERROR.

; NOW CLOSE THE FILE AFTER SUCCESSFUL WRITE.

        LDA     #CLOSE          ; SET UP FOR "CLOSE. "
        STA     ICCOM, X


        JSR     CIOV            ; CLOSE THE FILE.
        BMI     CERR            ; ERROR.


        BRK                     ; STOP WHEN DONE.

CERR    BRK                     ; STOP ON ERROR.

CFILE   .BYTE   "C: ", CR       ; FILE NAME.

; THIS IS THE CARTRIDGE HEADER

*=      $BFF9                   ; "A" CARTRIDGE.

INIT    RTS
        .WORD   BOOTB
        .BYTE   0, 4

        .WORD   INIT
        .END
```

11 ADVANCED TECHNIQUES AND APPLICATION NOTES

This section presents information to use the capabilities of the OS
and some of the hardware capabilites that aren't directly available
through the OS ,and in fact, can be in direct conflict with parts of
the OS.

SOUND GENERATION

The OS uses the POKEY sound generation capabilities only in the I/O
subsystem, for cassette FSK tone generation, and for the "noisy
bus" option in SIO.

Capabilities

The hardware provides four independently programmable audio
channels that are mixed and sent to the television set as part of
the composite video signal. The POKEY registers shown below are all
concerned with sound control (as described in the ATARI Home
Computer Hardware Manual).

AUDCTL [D208]		Audio control.
AUDC1 [D201] and AUDF1 [D200]		Channel 1 control.
AUDC2 [D203] and AUDF2 [D202]		Channel 2 control.
AUDC3 [D205] and AUDF3 [D204]		Channel 3 control.
AUDC4 [D207] and AUDF4 [D206]		Channel 4 control.

Conflicts With OS

There are two potential conflicts with the OS involving sound
generation:

o   The OS can generate its own sounds and then turn off all sounds
     as part of I/O operations to the cassette and the serial bus
     peripherals.

o   The OS does not turn off sounds when you press [SYSTEM.RESET] or
     [BREAK]. If the sounds are to be turned off under those
     conditions, the controlling program must provide that capability.

SCREEN GRAPHICS

Hardware Capabilities

The hardware capabilities for screen presentations are quite versatile; the OS uses a very small amount of the capability provided. The means of extension, however, are non-trivial; and making changes to a screen format while still utilizing the resident Display Handler will be difficult. See the ATARI Home Computer Hardware Manual for information regarding screen presentations.

OS Capabilities

The resident Display Handler arbitrarily supports 8 of the 11 possible full screen modes (11 of 14 modes if the GTIA chip is used in place of the CTIA). The resident Display Handler allows for an optional "split-screen" text window of fixed size. The hardware allows for many more options than the Display Handler supports, as will be seen by reading the ATARI Home Computer Hardware Manual.

Cursor Control

You can control the Display Handler text and graphics cursors directly (see Section 5 and Appendix L, B1-4).

Color Control

You can alter the color register assignments that the Display Handler makes upon all OPEN commands (see Appendix L B7-8 and elsewhere). Note that every system reset or Display Handler OPEN will reset the values back to the system default.

Alternate Character Sets

Two character sets are available in screen text modes 1 and 2. The
value stored in the data base variable CHBAS [02F4] selects the
character set of interest to you. The default value of $E0
provides capital (uppercase) letters, numbers and the punctuation
characters corresponding to display codes $20 through $5F in
Appendix E). The alternate value of $E2 provides lowercase letters
and the special character graphics set (corresponding to display
codes $60 through $7F and $00 through $1F in Appendix E).

User-defined character sets can also be obtained for text modes 0,
1, and 2 by providing the character matrix definitions in RAM and
setting CHBAS to point to those definitions. CHBAS always contains
the most significant bits of the memory address of the start of the
character definitions, as shown below:

```
             7                 0
             +-+-+-+-+-+-+-+-+
CHBAS        |    MSB     |x x|     Text mode 0
             +-+-+-+-+-+-+-+-+

             +-+-+-+-+-+-+-+-+
             |    MSB     |x|       Text modes 1 and 2
             +-+-+-+-+-+-+-+-+
```
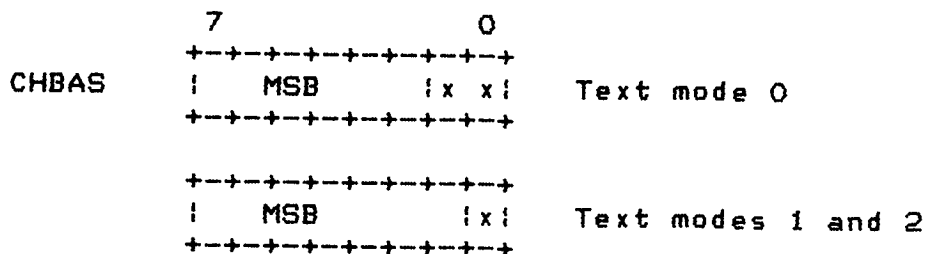
    Figure 11-1   User-Defined Character Set Bit Memory Addresses

                        (X indicates an ignored address bit
                        assumed to be 0.)

Each character is defined by an 8 x 8 bit matrix; the
character '@' is defined as shown below:

```
          7               0
  Byte  +-+-+-+-+-+-+-+-+
         !0!0!0!0!0!0!0!0!   0
         +-+-+-+-+-+-+-+-+
         !0!0!1!1!1!1!0!0!   1
         +-+-+-+-+-+-+-+-+
         !0!1!1!0!1!1!0!0!   2
         +-+-+-+-+-+-+-+-+
         !0!1!1!0!1!1!0!0!   3
         +-+-+-+-+-+-+-+-+
         !0!1!1!0!1!1!0!0!   4
         +-+-+-+-+-+-+-+-+
         !0!1!1!0!0!0!0!0!   5
         +-+-+-+-+-+-+-+-+
         !0!0!1!1!1!1!0!0!   6
         +-+-+-+-+-+-+-+-+
         !0!0!0!0!0!0!0!0!   7
         +-+-+-+-+-+-+-+-+
```

Figure 11-2   User Defined 8 x 8 Character Matrix Bit Table

The storage for the character set involves eight consecutive
bytes for each character with characters ordered consecutively by
their internal code value (see the discussion in Appendix L
relating to B55).

```
                   +----------------+
 Character base    ! Character for   !
                   !   code $00      ! 8 bytes
                   +----------------+
                   ! Character for   !
                   !   code $01      ! increasing addresses
                   +----------------+
                   !                 !
                   =                 =
                   !                 !
                   +----------------+
                   ! Character for   !
                   !   code $7E      !
                   +----------------+
                   ! Character for   !
                   !   code $7F      !
                   +----------------+
```
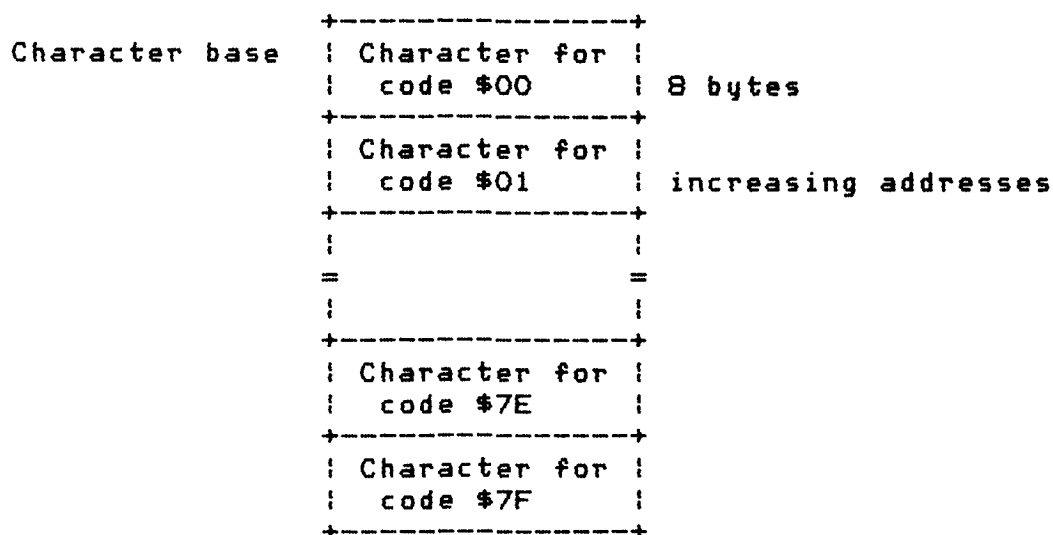
Figure 11-3   Character Base Diagram

PLAYER/MISSILE GRAPHICS

The OS makes no use of the player/missile generation capability
of the hardware. It can be used independently of the OS with no
conflict.

Hardware Capabilities

The hardware allows a number of independently moveable screen
objects of limited width to be positioned and moved about the
screen without affecting the "playfield" (bit-mapped graphics or
character) data. Priority control allows the various objects to
have a display precedence in case of conflict (overlap).

Conflicts With OS

You must assure that the player/missile data is
address-aligned as required by PMBASE [D407]. You also must
find a suitable free area that the OS guarantees to be free under
all environments.

READING GAME CONTROLLERS

The OS reads the game controllers (shown below) as part of the
stage 2 VBLANK process (see Appendix L J1-9):

    Joysticks/triggers 1-4.
    Paddle controllers/triggers 1-8.
    Driving controllers/triggers 1-4.
    Light pen/trigger

In addition to these controllers, other information can be sensed
or sent using the PIA chip to that the console connectors are
interfaced.

Keyboard Controller Sensing

Data can be read from an ATARI keyboard controller connected to the
first port. This program alters registers on a chip called a PIA.
To set these back to the default values to do further I/O, hit
[SYSTEM.RESET] or POKE PACTL,60. If this program is to be loaded from
diskette, use LOAD, not RUN and wait for the busy light on the disk
drive to go out. Do not execute the program before this light goes
out, otherwise the diskette continues to spin.

```
  1 GRAPHICS 0
  5 PRINT :PRINT "    KEYBOARD CONTROLLER DEMO"
 10 DIM ROW(3),I$(13),BUTTON$(1)
 30 GOSUB 6000
 40 FOR CNT=1 TO 4
 60 POSITION 2,CNT*2+5:PRINT "CONTROLLER # ";CNT;": ";
```

```
70 NEXT CNT
80 FOR CNT=1 TO 4:GOSUB 7000:POSITION 19,CNT+CNT+5:PRINT BUTTON$;
   :NEXT CNT
120 GOTO 80
6000 REM ** SET UP FOR CONTROLLERS **
6010 PORTA=54016:PORTB=54017:PACTL=54018:PBCTL=54019
6020 POKE PACTL,48:POKE PORTA,255:POKE PACTL,52:POKE PORTA,221
6025 POKE PBCTL,48:POKEPORTB,255:POKE PBCTL,52:POKE PORTB,221
6030 ROW(0)=238:ROW(1)=221:ROW(2)=187:ROW(3)=119
6040 I$=" 123456789*0#"
6050 RETURN
7000 REM ** RETURN BUTTON$ WITH CHARACTER FOR BUTTON WHICH HAS
     BEEN PRESSED ON CONTROLLER CNT (1-4). **
7001 REM ** NOTE: A 1 WILL BE RETURNED IF NO CONTROLLER IS
     CONNECTED. **
7002 REM ** A SPACE WILL BE RETURNED IF THE CONTROLLER IS
     CONNECTED BUT NO KEY HAS BEEN PRESSED. **
7003 PORT=PORTA:IF CNT>2 THEN PORT=PORTB
7005 P=1
7008 PAO=CNT+CNT-2
7010 FOR J=0 TO 3
7020 POKE PORT,ROW(J)
7030 FOR I=1 TO 10:NEXT I
7050 IF PADDLE(PAO+1)>10 THEN P=J+J+J+2:GOTO 7090
7060 IF PADDLE(PAO)>10 THEN P=J+J+J+3:GOTO 7090
7070 IF STRIG(CNT-1)=0 THEN P=J+J+J+4:GOTO 7090
7080 NEXT J
7090 BUTTON$=I$(P,P)
7095 RETURN
```

Figure 11-4 Reading Data From an ATARI Keyboard Controller

The table below shows the variable/register values used for reading a keyboard controller from each of the four controller ports.

	Port 1	Port 2	Port 3	Port 4
PORT A direction bits	OF	FO	—	—
PORT B direction bits	—	—	OF	FO
Port A row select	FE,FD, FB,F7	EF,DF BF,7F	—	—
Port B row select	—	—	FE,FD, FB,F7	EF,DF, BF,7F
Column 1 Sense	PADDL1	PADDL3	PADDL5	PADDL7
Column 2 Sense	PADDL0	PADDL2	PADDL4	PADDL6
Column 3 Sense	STRIG0	STRIG1	STRIG2	STRIG3

Figure 11-5   ATARI Keyboard Controller Variable/Register Value
Table

Front Panel Connectors as I/O Ports

The three pages that follow show how some of the pins in the front panel (game controller) connectors can be used as general I/O pins.

   Hardware Information

   PIA   (6520 / 6820)
      Out: TTL levels, 1 load
      In :   TTL levels, 1 load   For more information refer
                                     to 6520 chip manual.

Port A Circuit (typical):

```
 ┌──────────┐        220
 │      A   │       ╱╲╱╲        ┤├ Jack
 │ 6520     ├────────────┬───────
 │     port │           ═╪═ 001
 └──────────┘            ▽
```

```
        1          5
      ┌─────────────┐
      │ •  •  •  •  •│
      │  •  •  •  •  │
       ╲───────────╱
        6        9
```
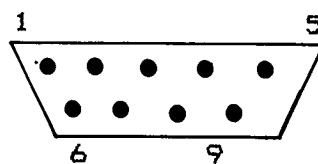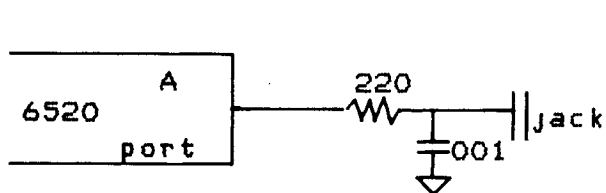
Male connector, FRONT view
Pin 8 = Ground
Pin 7 = Vcc 8+5v *)
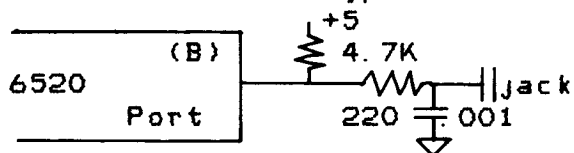
Note: 50mA maximum
total external drain
on power supply allowed

─────────────

Port B Circuit (typical):

```
 ┌──────────┐      +5
 │     (B)  │     ╱╲╱╲ 4. 7K
 │ 6520     ├──────┬──╱╲╱╲──┬──── ┤├ Jack
 │     Port │          220 ═╪═ 001
 └──────────┘              ▽
```

─────────────

"Trigger" Port Circuit (typical):

```
 ┌──────────┐        220
 │ CTIA Trig├────────╱╲╱╲───┬──────┤├  Jack
 └──────────┘             ═╪═ .001
                           ▽
```

─────────────
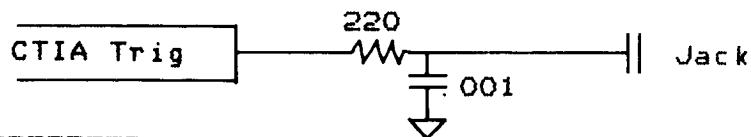
Software Information

6520 PIA: (This also pertains to all of the following: **)
   Port A control (address D302)

```
   7   6   5   4   3   2   1   0
 ┌───┬───┬───┬───┬───┬───┬───┬───┐
 │ O │ O │ 1 │ 1 │ 1 │ X │ O │ O │ Write this into this register
 └───┴───┴───┴───┴───┴───┴───┴───┘
                   ↑
                   └──── Port A Data/Data direction addres
                         ing control
                         O = Data Direction is at D300
                         1 = data is at D300
```

─────────────

Port A data direction (address D300)

```
   7   6   5   4   3   2   1   0
 ┌───┬───┬───┬───┬───┬───┬───┬───┐
 │ X │ X │ X │ X │ X │ X │ X │ X │   Write this into this register.
 └───┴───┴───┴───┴───┴───┴───┴───┘
   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑
   └───┴───┴───┴───┴───┴───┴───┴──── Data direction control
                                     for Port A
                                         1 = Out
                                         O = In
```
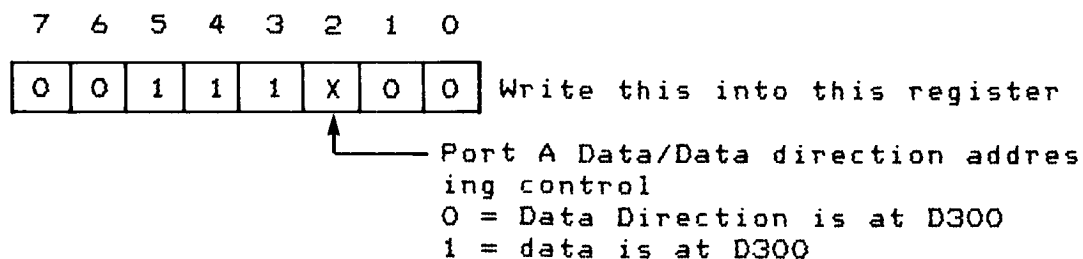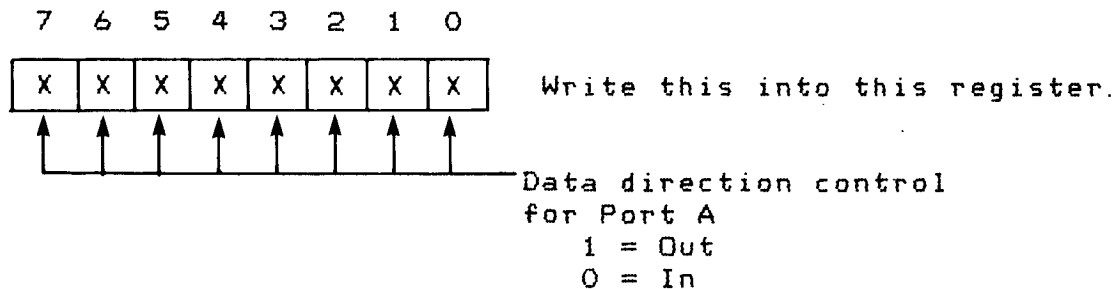
Port A data (address D300)
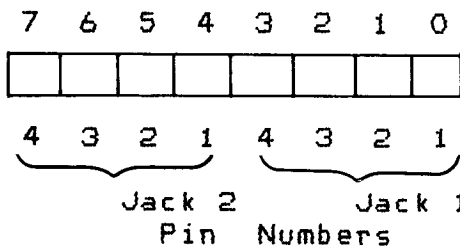
```
 7  6  5  4  3  2  1  0
┌──┬──┬──┬──┬──┬──┬──┬──┐
│  │  │  │  │  │  │  │  │     Read or Write this register
└──┴──┴──┴──┴──┴──┴──┴──┘
 4  3  2  1  4  3  2  1
 └────────┘  └────────┘
    Jack 2      Jack 1
      Pin  Numbers
```

Port B Control (address D303)

```
┌──┬──┬──┬──┬──┬──┬──┬──┐
│ 0│ 0│ 1│ 1│ 1│ X│ 0│ 0│
└──┴──┴──┴──┴──┴──┴──┴──┘
```
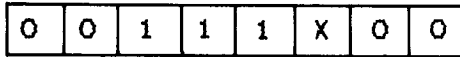
6520 PIA:
      Port B Control (address D303)
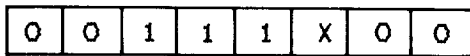
```
         7  6  5  4  3  2  1  0
      ┌──┬──┬──┬──┬──┬──┬──┬──┐
      │ 0│ 0│ 1│ 1│ 1│ X│ 0│ 0│     write this into this register
      └──┴──┴──┴──┴──┴──┴──┴──┘
                      ↑
                      └──────Port B Data/Data direction
                                  addressing control
                             0 = D301 contains data
                                  direction
                             1 = $D301 contains
```

Port B data direction (address D301)
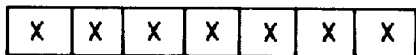
```
       7  6  5  4  3  2  1
    ┌──┬──┬──┬──┬──┬──┬──┐
    │ X│ X│ X│ X│ X│ X│ X│     write this into this register
    └──┴──┴──┴──┴──┴──┴──┘
      ↑  ↑  ↑  ↑  ↑  ↑  ↑
      └──┴──┴──┴──┴──┴──┴───────data direction control for Port B
                             1 = Out
                             0 = In
```

Port B data (address D301)

```
    7  6  5  4  3  2  1  0
 ┌──┬──┬──┬──┬──┬──┬──┬──┐
 │  │  │  │  │  │  │  │  │
 └──┴──┴──┴──┴──┴──┴──┴──┘
  4  3  2  1  4  3  2  1
  └────────┘  └────────┘
    Jack 4       J ack 3
      Pin  Numbers
```
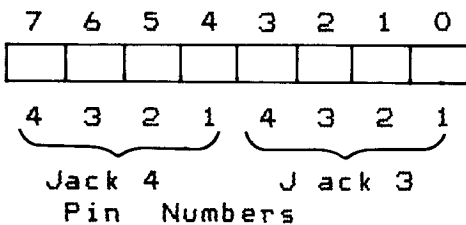
Four "Trigger" ports: D010, D011, D012, D013
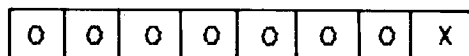
```
    7  6  5  4  3  2  1  0
 ┌──┬──┬──┬──┬──┬──┬──┬──┐
 │ 0│ 0│ 0│ 0│ 0│ 0│ 0│ X│  Read this port
 └──┴──┴──┴──┴──┴──┴──┴──┘
                      ↑
                      └──Trigger Value
                         D010 = Port 1 Pin 6
                         D013 = Port 4 pin 6
```

Other Miscellaneous Software Information

1). The OS sets up all PIA ports as inputs during
initialization.
2). The OS usually reads the above once per television frame
(during
vertical-blank) into RAM as follows:

Data Base Name	Address	Data	Pins S
STICK0	0278	7 6 5 4 3 2 1 0	Jack 1, pins 4,3,2, if 10053,7
		0 0 0 0 X X X X	
STICK1	0729		Jack 2, Pins 4,3,2,1
STICK2	027A		Jack 3, Pins 4,3,2,1
STICK3	027B		Jack 4, Pins 4,3,2,1
STRIG0	0284		Jack 1, Pin 6
STRIG1	0285	7 6 5 4 3 2 1 0	Jack 2, Pin 6
		0 0 0 0 0 0 0	
STRIG2	0286		Jack 3, Pin 6
STRIG3	0287		Jack 4, Pin 6
PADDL1	0270	7 6 5 4 3 2 1 0	Jack 1, Pin 5
		X X X X X X X X	
PADDL3	0272		Jack 2, Pin 5
PADDL5	0274		Jack 3, Pin 5
PADDL7	0276		Jack 4, Pin 5
PADDL0	0271		Jack 1, Pin 9
PADDL2	0273		Jack 2, Pin 9
PADDL4	0275		Jack 3, Pin 9
PADDL 6	0277		Jack 4, Pin 9

Figure 11-6   Using Front Panel Connectors As I/O Ports: Pin
Function Tables

* Pins 5 and 9 are read through the paddle controller circuitry
a nominal value of 7 indicates that the pin is high (or floating)
and a nominal value of 228 indicates that the pin is pulled low.

Appendix A -- CIO COMMAND BYTE VALUES

The following hex values are known to be legitimate CIO commands.

Most handlers:

    03 -- OPEN
    05 -- GET RECORD
    07 -- GET CHARACTERS
    09 -- PUT RECORD
    OB -- PUT CHARACTERS
    OC -- CLOSE
    OD -- GET STATUS

Display Handler only:

    11 -- FILL
    12 -- DRAW

Diskette File Manager only:

    20 -- RENAME
    21 -- DELETE
    22 -- FORMAT
    23 -- LOCK
    24 -- UNLOCK
    25 -- POINT
    26 -- NOTE

Appendix B -- CIO STATUS BYTE VALUES

Shown below are the known CIO STATUS BYTE values.

01 (001) -- OPERATION COMPLETE (NO ERRORS)

80 (128) -- [BREAK] KEY ABORT
81 (129) -- IOCB ALREADY IN USE (OPEN)
82 (130) -- NON-EXISTENT DEVICE
83 (131) -- OPENED FOR WRITE ONLY
84 (132) -- INVALID COMMAND
85 (133) -- DEVICE OR FILE NOT OPEN
86 (134) -- INVALID IOCB NUMBER (Y reg only)
87 (135) -- OPENED FOR READ ONLY
88 (136) -- END OF FILE
89 (137) -- TRUNCATED RECORD
8A (138) -- DEVICE TIMEOUT (DOESN'T RESPOND)
8B (139) -- DEVICE NAK
8C (140) -- SERIAL BUS INPUT FRAMING ERROR
8D (141) -- CURSOR out-of-range
8E (142) -- SERIAL BUS DATA FRAME OVERRUN ERROR
8F (143) -- SERIAL BUS DATA FRAME CHECKSUM ERROR
90 (144) -- DEVICE DONE ERROR
91 (145) -- BAD SCREEN MODE
92 (146) -- FUNCTION NOT SUPPORTED BY HANDLER
93 (147) -- INSUFFICIENT MEMORY FOR SCREEN MODE

A0 (160) -- DISK DRIVE # ERROR
A1 (161) -- TOO MANY OPEN DISK FILES
A2 (162) -- DISK FULL
A3 (163) -- FATAL DISK I/O ERROR
A4 (164) -- INTERNAL FILE # MISMATCH
A5 (165) -- FILE NAME ERROR
A6 (166) -- POINT DATA LENGTH ERROR
A7 (167) -- FILE LOCKED
A8 (168) -- COMMAND INVALID FOR DISK
A9 (169) -- DIRECTORY FULL (64 FILES)
AA (170) -- FILE NOT FOUND
AB (171) -- POINT INVALID

Appendix C -- SIO STATUS BYTE VALUES

Shown below are the known SIO STATUS BYTE hexadecimal values.

01 (001) -- OPERATION COMPLETE (NO ERRORS)

8A (138) -- DEVICE TIMEOUT (DOESN'T RESPOND)
8B (139) -- DEVICE NAK
8C (140) -- SERIAL BUS INPUT FRAMING ERROR
8E (142) -- SERIAL BUS DATA FRAME OVERRUN ERROR
8F (143) -- SERIAL BUS DATA FRAME CHECKSUM ERROR
90 (144) -- DEVICE DONE ERROR

Appendix D -- ATASCII CODES

	0X	2X	4X	6X	8X	AX	CX	EX
00	♥	Space	@	●				
01		!	A	a				
02		"	B	b				
03		#	C	c				
04		$	D	d				
05		%	E	e				
06		&	F	f				
07		'	G	g				
08		(	H	h				
09		)	I	i				
0A		*	J	j				
0B		+	K	k				
0C		,	L	l				
0D		-	M	m				
0E		.	N	n				
0F		/	O	o				
10		0	P	p				
11		1	Q	q				
12		2	R	r				
13		3	S	s				
14		4	T	t				
15		5	U	u				
16		6	V	v				
17		7	W	w				
18		8	X	x				
19		9	Y	y				
1A		:	Z	z				
1B	ESC	;	[	●	EOL			
1C	↑	<	\	|	DEL LINE			
1D	↓	=	]	CLEAR	INS LINE		BELL	
1E	←	>	^	BACKSP	CLR TAB		DEL CHAR	
1F	→	?	_	TAB	SET TAB		INS CHAR	

	ØX	2X	4X	6X	8X	AX	CX	EX
ØØ	♥	Space	@	◻				
Ø1	⊡	!	A	a				
Ø2	⊟	"	B	b				
Ø3	⊡	#	C	c				
Ø4	⊡	$	D	d				
Ø5	⊡	%	E	e				
Ø6	⧄	&	F	f				
Ø7	◣	'	G	g				
Ø8	◥	(	H	h				
Ø9	�	)	I	i				
ØA	◤	*	J	j				
ØB	⊡	+	K	k				
ØC	⊡	,	L	l				
ØD	⊟	−	M	m				
ØE	⊟	.	N	n				
ØF	◪	/	O	o				
1Ø	⊡	Ø	P	p				
11	⊡	1	Q	q				
12	⊟	2	R	r				
13	✛	3	S	s				
14	⊡	4	T	t				
15	⊟	5	U	u				
16	◫	6	V	v				
17	⊡	7	W	w				
18	⊡	8	X	x				
19	◫	9	Y	y				
1A	⊡	:	Z	z				
1B	⧉	;	[	◻				
1C	↑	<	\	│				
1D	↓	=	⌐	◰				
1E	←	>	∧	◧				
1F	→	?	−	◨				

CODES 8Ø-FF SHOW AS
THE INVERSE VIDEO
OF CODES ØØ-7F

CTRL			SHIFT & LOWER			SHIFT			LOWER		
00	,	20	20	<space>	21	40	@	35	60	^.	22
01	A	3F	21	!	1F	41	A	3F	61	a	3F
02	B	15	22	"	1E	42	B	15	62	b	15
03	C	12	23	#	1A	43	C	12	63	c	12
04	D	3A	24	$	18	44	D	3A	64	d	3A
05	E	2A	25	%	1D	45	E	2A	65	e	2A
06	F	38	26	&	1B	46	F	38	66	f	38
07	G	3D	27	'	33	47	G	3D	67	g	3D
08	H	39	28	(	30	48	H	39	68	h	39
09	I	0D	29	)	32	49	I	0D	69	i	0D
0A	J	01	2A	*	07	4A	J	01	6A	J	01
0B	K	05	2B	+	06	4B	K	05	6B	k	05
0C	L	00	2C	,	20	4C	L	00	6C	l	00
0D	M	25	2D	-	0E	4D	M	25	6D	m	25
0E	N	23	2E	.	22	4E	N	23	6E	n	23
0F	O	08	2F	/	26	4F	O	08	6F	o	08
10	P	0A	30	0	32	50	P	0A	70	p	0A
11	Q	2F	31	1	1F	51	Q	2F	71	q	2F
12	R	28	32	2	1E	52	R	28	72	r	28
13	S	3E	33	3	1A	53	S	3E	73	s	3E
14	T	2D	34	4	18	54	T	2D	74	t	2D
15	U	0B	35	5	1D	55	U	0B	75	u	0B
16	V	10	36	6	1B	56	V	10	76	v	10
17	W	2E	37	7	33	57	W	2E	77	w	2E
18	X	16	38	8	35	58	X	16	78	x	16
19	Y	2B	39	9	30	59	Y	2B	79	y	2B
1A	Z	17	3A	:	02	5A	Z	17	7A	z	17
1B	<esc>	1C	3B	;	0D	5B	[	20	7B	;	02
1C	^<up>	0E	3C	<	36	5C	\	06	7C	!	0F
1D	^<down>	0F	3D	=	0F	5D	]	22	7D	<clear>	36
1E	^<left>	06	3E	>	37	5E	^	07	7E	<back>	34
1F	^<right>	07	3F	?	26	5F	_	0E	7F	<tab>	2C

```
80-9A   /!\    00-1A        9F  s<tab>      2C
9B  <return> and ^3  0C,1A   A0-FC  /!\    20-7C
9C  s<del>    34             FD  ^2          1E
9D  s<insert>37              FE  ^<del>      34
9E  ^<tab>    2C             FF  ^<insert>37
```

```
<clear> ::= s< or  ^<
<return> ::= <return> or s<return> or  ^<return>
<esc> ::= <esc> or s<esc> or  ^<esc>
<space> ::= <space> or s<space> or  ^<space>
```

```
Where:  s as a prefix indicates [SHIFT].
        ^ as a prefix indicates [CTRL].
        /!\ as a prefix indicates ATARI key inverse active.
```

Appendix G -- PRINTER CODES (ATASCII)

Character set for "normal" mode printing:

20	<space>	40	@	60	`
21	!	41	A	61	a
22	"	42	B	62	b
23	#	43	C	63	c
24	$	44	D	64	d
25	%	45	E	65	e
26	&	46	F	66	f
27	'	47	G	67	g
28	(	48	H	68	h
29	)	49	I	69	i
2A	*	4A	J	6A	j
2B	+	4B	K	6B	k
2C	,	4C	L	6C	l
2D	-	4D	M	6D	m
2E	.	4E	N	6E	n
2F	/	4F	O	6F	o
30	0	50	P	70	p
31	1	51	Q	71	q
32	2	52	R	72	r
33	3	53	S	73	s
34	4	54	T	74	t
35	5	55	U	75	u
36	6	56	V	76	v
37	7	57	W	77	w
38	8	58	X	78	x
39	9	59	Y	79	y
3A	:	5A	Z	7A	z
3B	;	5B	[	7B	{
3C	<	5C	\	7C	¦
3D	=	5D	]	7D	}
3E	>	5E	^	7E	~
3F	?	5F	_	7F	<space>

Note: The following codes print differently than defined by
the ATASCII definition.

    00 through 1F print blank.
    60 prints ` instead of "diamond".
    7B prints { instead of "spade".
    7D prints } instead of "clear".
    7E prints ~ instead of "backspace".
    7F prints blank instead of "tab".

Character set for "sideways" mode printing:

```
                    40   @        60   @
                    41   A        61   A
                    42   B        62   B
                    43   C        63   C
                    44   D        64   D
                    45   E        65   E
                    46   F        66   F
                    47   G        67   G
                    48   H        68   H
                    49   I        69   I
                    4A   J        6A   J
                    4B   K        6B   K
                    4C   L        6C   L
                    4D   M        6D   M
                    4E   N        6E   N
                    4F   O        6F   O
      30   0        50   P        70   P
      31   1        51   Q        71   Q
      32   2        52   R        72   R
      33   3        53   S        73   S
      34   4        54   T        74   T
      35   5        55   U        75   U
      36   6        56   V        76   V
      37   7        57   W        77   W
      38   8        58   X        78   X
      39   9        59   Y        79   Y
      3A   :        5A   Z        7A   Z
      3B   ;        5B   [        7B   [
      3C   <        5C   \        7C   \
      3D   =        5D   ]        7D   ]
      3E   >        5E   <up>     7E   <up>
      3F   ?        5F   <left>   7F   <left>
```

Note: the following codes print differently than defined by
the ATASCII definition.

```
    00 through 2F print blank.
    5E prints "up arrow" instead of  .
    5F prints "left arrow" instead of _.
    60 through 7F repeats 40 through 5F instead of proper set.
```

Appendix H -- SCREEN MODE CHARACTERISTICS

Mode #	Horiz. Posit.	Vert. W/O Sp	Vert. W Sp	Colors	Data Value	Color Reg.	Memory Reqd. (split)	(full)
0	40	24	--	2	backgd. 00-FF "	BAK PF 2 PF 1*	992	992
1	20	24	20	5	backgd. 00-3F 40-7F 80-BF C0-FF	BAK PF 0 PF 1 PF 2 PF 3	674	672
2	20	12	10	5	backgd. 00-3F 40-7F 80-BF C0-FF	BAK PF 0 PF 1 PF 2 PF 3	424	420
3	40	24	20	4	0 1 2 3	BAK PF 0 PF 1 PF 2	434	432
4	80	48	40	2	0 1	BAK PF 0	694	696
5	80	48	40	4	0 1 2 3	BAK PF 0 PF 1 PF 2	1174	1176
6	160	96	80	2	0 1	BAK PF 0	2174	2184
7	160	96	80	4	0 1 2 3	BAK PF 0 PF 1 PF 2	4190	4200
8	320	192	160	2	0 1	PF 2 PF 1*	8112	8138
9	80	192	--	1	Note 2			8138
10	80	192	--	9	0 1 2 3 4	PM 0 PM 1 PM 2 PM 3 PF 0		8138

```
                                          5          PF 1
                                          6          PF 2
                                          7          PF 3
                                          8          BAK
                                          9          BAK
                                          A          BAK
                                          B          BAK
                                          C          PF 0
                                          D          PF 1
                                          E          PF 2
                                          F          PF 3


11    80      192        --       16     Note 3                  8138
```

Notes:

```
*     Uses color of PF 2, lum of PF 1.
2     Uses color of BAK, lum of data value ($0-F).
3     Uses color of data value ($0-F), lum of BAK.
```

PF x ::= Playfield color register x.
PM x ::= Player/Missile Graphics color register x.
BAK ::= Background color register (also known as PF 4).

The default values for the color registers are shown below:

```
            BAK  =  $00
            PF0  =  $28
            PF1  =  $CA
            PF2  =  $94
            PF3  =  $46
```

The form of a color register byte is shown below:

```
      7 6 5 4 3 2 1 0
     +-+-+-+-+-+-+-+-+
     : color : lum :O:
     +-+-+-+-+-+-+-+-+
```

Where: color (hex values)   lum

```
    0 = gray                0 = minimum luminance
    1 = light orange        1 =         :
    2 = orange              2 =         :
    3 = red orange          3 =    (increasing
    4 = pink                4 =     luminance)
    5 = purple              5 =         :
    6 = purple-blue         6 =         :
    7 = blue                7 = maximum luminance
    8 = blue
    9 = light blue
    A = turquoise
    B = green-blue
    C = green
    D = yellow-green
    E = orange-green
    F = light orange
```

Appendix I -- SERIAL BUS ID AND COMMAND SUMMARY

Serial bus device IDs

        Floppy diskettes D1-D4 $31-34
        Printer   P1 $40
        RS-232-C   R1-R4 $50-53

Serial bus control codes

        ACK   - $41 ('A')
        NAK   - $4E ('N')
        COMPLETE   - $43 ('C')
        ERR   - $45 ('E')

Serial bus command codes

        READ   - $52 ('R')   Disk
        WRITE   - $57 ('W')   Printer/Disk
        STATUS   - $53 ('S')   Printer/Disk
        PUT(no check)   - $50 ('P')   Disk
        FORMAT   - $21 ('!')   Disk
        READ ADDRESS   - $54 ('T')
        READ SPIN   - $51 ('Q')   Disk
        MOTOR ON   - $55 ('U')   Disk
        VERIFY SECTOR   - $56 ('V')   Disk

The fixed address OS ROM JMP vectors are shown below; at each address is a JMP instruction to the indicated routine.

Name	Addr	Reference	Function
DISKIV	E450	*	Diskette Handler initialization
DSKINV	E453	5.4.2	Diskette Handler entry.
CIOV	E456	5.2	CIO utility entry.
SIOV	E459	9.3	SIO utility entry.
SETVBV	E45C	6.7.2	Set System Timers routine.
SYSVBV	E45F	6.3	Stage 1 VBLANK entry.
XITVBV	E462	6.3	Exit VBLANK entry.
SIOINV	E465	*	SIO utility initialization.
SENDEV	E468	*	Send enable routine.
INTINV	E46B	*	Interrupt Handler initialization.
CIOINV	E46E	*	CIO utility initialization.
BLKBDV	E471	3.1.1	Blackboard mode entry.
WARMSV	E474	7.	Warmstart ([SYSTEM.RESET]) entry.
COLDSV	E477	7.	Coldstart (power-up) entry.
RBLOKV	E47A	*	Cassette-read block entry.
CSOPIV	E47D	*	Cassette-OPEN input entry.

* These vectors are for OS internal use only.

The fixed address Floating Point Package ROM routine entry point addresses are shown below; complete descriptions of the corresponding routines are provided in Section 8.

AFP	D800	ASCII to FP convert.
FASC	D8E6	FP to ASCII convert.
IFP	D9AA	Integer to FP convert.
FPI	D9D2	FP to integer convert.
FADD	DA66	FP add.
FSUB	DA60	FP subtract.
FMUL	DADB	FP multiply
FDIV	DB28	FP divide.
LOG	DECD	FP base e logarithm.
LOG10	DED1	FP base 10 logarithm.
EXP	DDC0	FP base e exponentiation.
EXP10	DDCC	FP base 10 exponentiation.
PLYEVL	DD40	FP polynomial evaluation.
ZFRO	DA44	Clear FR0.
ZF1	DA46	Clear FP number.
FLD0R	DD89	Load FP number.
FLD0P	DD8D	Load FP number.
FLD1R	DD98	Load FP number.
FLD1P	DD9C	Load FP number.
FSTOR	DDA7	Store FP number.
FSTOP	DDAB	Store FP number.
FMOVE	DDB6	Move FP number.

The base addresses of the Handler vectors for the resident
handlers are shown below:

```
 Screen Editor (E)            E400
 Display Handler (S)          E410
 Keyboard Handler (K)         E420
 Printer Handler (P)          E430
 Cassette Handler (C)         E440
```

See Section 5 for the format of the entry for each Handler.

The 6502 Computer interrupt vector values are shown below:

Function	Address	Value
NMI	FFFA	E7B4
RESET	FFAC	E477
IRQ	FFFE	E6FE

Appendix K -- DEVICE CHARACTERISTICS

This appendix describes the physical characteristics of the
devices that interface to the ATARI 400 and ATARI 800 Home
Computers. Where applicable, data capacity, data transfer
rate, storage format, SIO interface, and cabling will be
detailed.

KEYBOARD

The keyboard input rate is limited by the OS keyboard reading
procedure to be 60 characters per second. The code for each key
is shown in Table 5-4. The keyboard hardware has no buffering and is
rate-limited by the debounce algorithm used.

DISPLAY

The television screen display generator has many capabilities
that are not used by the Display Handler (as described in Section
5 and shown in Appendix H). There are additional display modes,
object generators, hardware display scrolling, and many other
features that are described in the ATARI Home Computer
Hardware Manual.

Since all display data is stored in RAM, the display data update
rate is limited primarily by the software routines that generate
and format the data and access the RAM. The generation of the
display from the RAM is accomplished by the ANTIC and CTIA or GTIA
chips using Direct Memory Access (DMA) to access the RAM data.

The internal storage formats for display data for the various
modes are detailed in the ATARI Home Computer Hardware
Manual.

ATARI 410 PROGRAM RECORDER

The ATARI 410 Program Recorder has the following characteristics:

DATA CAPACITY:

    100 characters per C-60 tape (unformatted).

DATA TRANSFER RATES:

    * 600 Baud (60 characters per second)

*Note: The OS has the ability to adjust to different tape speeds
  (447 - 895 Baud).

STORAGE FORMAT:

Tapes are recorded in 1/4 track stereo format at 1 7/8 inches per
second. The tape can be recorded in both directions, where tracks
1 and 2 are side A left and right; and tracks 3 and 4 are side B
right and left (industry standard). On each side, the left
channel (1 or 4) is used for audio and the right channel (2 and
3) is used for digital information.

The audio channel is recorded the normal way. The digital channel
is recorded using the POKEY two-tone mode producing FSK data at
up to 600 baud. The MARK frequency is 5327 Hz and the SPACE
frequency is 3995 Hz. The transmission of data is asynchronous
byte serial as seen from the computer; POKEY reads or writes a
bit serial FSK sequence for each byte, in the following order:

```
     1 start bit (SPACE)
     data bit 0 —+
     data bit 1  |
     .              +—  0 = SPACE, 1 = MARK.
     data bit 6  |
     data bit 7 —+
     1 stop bit (MARK)
```

The only control the computer has over tape motion is motor
start/stop; and this only if the PLAY button is pressed by the
user. In order for recording to take place, the user must press
both the REC and PLAY buttons on the cassette. The computer has
no way to sense the position of these buttons, nor even if an
ATARI 410 Program Recorder is cabled to the computer, so the user must
be careful when using this device.

   SIO INTERFACE

The cassette device utilizes portions of the serial bus hardware,
but does not follow any of the protocol as defined in Section 9.

ATARI 820[TM] 40-COLUMN IMPACT PRINTER

The ATARI 820 Printer has the following characteristics:

DATA CAPACITY:

     40 characters per line (normal printing)
     29 characters per line (sideways printing)

DATA TRANSFER RATES:

Bus rate: xx characters per second.
Print time (burst): xx characters per second.
Print time (average): xx characters per second.

STORAGE FORMAT:

    3 7/8 inch wide paper.
    5X7 dot matrix, impact printing.

 Normal format --
    40 characters per line.
    6 lines per inch (vertical).
    12 characters per inch (horizontal).

 Sideways format --
    29 characters per line.
    6 lines per inch (vertical).
    9 characters per inch (horizontal).

   SIO INTERFACE

The controller serial bus ID is $40.

The controller supports the following SIO commands (see Section 5
for more information regarding the Handler and Section 9 for a
general discussion of bus commands):

   GET STATUS

The computer sends a command frame of the format shown below:

    Device ID = $40.
    Command byte = $53.
    auxiliary 1 = doesn't matter.
    auxiliary 2 = doesn't matter.
    Checksum   = checksum of bytes above.

The printer controller responds with a data frame of the format
shown earlier in this appendix as part of the GET STATUS
discussion.

   PRINT LINE

The computer sends a command frame of the format shown below:

    Device ID = $40.
    Command byte = $57.

```
auxiliary 1 = doesn't matter.
auxiliary 2 = $4E for normal print or $53 for sideways.
Checksum    = checksum of bytes above.
```

The computer sends a data frame of the format shown below:

```
Leftmost character of line (column 1).
Next character of line (column 2).
.

.
Rightmost character of line (column 40 or 29).
Checksum byte.
```

Note that the data frame size is variable, either 41 or 30 bytes in length, depending upon the print mode specified in the command frame.

ATARI 810 DISK DRIVE

The ATARI 810[TM] Disk Drive has the following characteristics:

DATA CAPACITY:

    720 sectors of 128 bytes each (Disk Handler format).
    709 sectors of 125 data bytes each (Disk File Manager format).

DATA TRANSFER RATES:

    Bus rate: 1920 characters per second.
    Seek time: 5.25 msec. per track + 10 to 210 msec.
    Rotational latency: 104 msec maximum (288 rpm).

STORAGE FORMAT:

    5 1/4 inch diskette, soft sectored by the controller.
    40 tracks per diskette.
    18 sectors per track.
    128 bytes per sector.
    Controlled by National INS1771-1 formatter/controller chip.
    Sector sequence per track is: 18, 1, 3, 5, 7, 9, 11, 13, 15,
                                   17, 2, 4, 6, 8, 10, 12, 14, 16

SIO INTERFACE

The controller serial bus IDs range from $31 (for 'D1') to $34 (for 'D4').

The controller supports the following SIO commands (see earlier
in this Appendix for information about the Diskette Handler and
Section 9 for a general discussion of bus commands):

   GET STATUS

The computer sends a command frame of the format shown below:

     Device ID = $31-34.
     Command byte = $53.
     auxiliary 1 = doesn't matter.
     auxiliary 2 = doesn't matter.
     Checksum   = checksum of bytes above.

The diskette controller responds with a data frame of the format
shown earlier in this Appendix as part of the STATUS REQUEST
discussion.

   PUT SECTOR (WITH VERIFY)

The computer sends a command frame of the format shown below:

     Device ID = $31-34
     Command byte = $57.
     auxiliary 1 = low byte of sector number.
     auxiliary 2 = high byte of sector number (1-720).
     Checksum   = checksum of bytes above.

The computer sends a data frame of the format shown below:

     128 data bytes.
     Checksum byte.

The diskette controller writes the frame data to the specified
sector, then reads the sector and compares the content with the
frame data. The COMPLETE byte value indicates the status of the
operation.

   PUT SECTOR (NO VERIFY)

The computer sends a command frame of the format shown below:

     Device ID = $31-34
     Command byte = $50.
     auxiliary 1 = low byte of sector number.
     auxiliary 2 = high byte of sector number (1-720).
     Checksum   = checksum of bytes above.

The computer sends a data frame of the format shown below:

128 data bytes.
Checksum byte.

The diskette controller writes the frame data to the specified sector, then sends a COMPLETE byte value that indicates the status of the operation.

GET SECTOR

The computer sends a command frame of the format shown below:

        Device ID = $31-34
        Command byte = $52.
        auxiliary 1 = low byte of sector number.
        auxiliary 2 = high byte of sector number (1-720).
        Checksum   = checksum of bytes above.

The diskette controller sends a data frame of the format shown below:

        128 data bytes.
        Checksum byte.

FORMAT DISKETTE

The computer sends a command frame of the format shown below:

        Device ID = $31-34
        Command byte = $21.
        auxiliary 1 = doesn't matter.
        auxiliary 2 = doesn't matter.
        Checksum   = checksum of bytes above.

The diskette controller completely formats the diskette (generates 40 tracks of 18 soft sectors per track with the data portion of each sector equal to all zeros) and then reads each sector to verify its integrity. A data frame of 128 bytes plus checksum is returned in that the sector numbers of all bad sectors (up to a maximum of 63 sectors) are contained, followed by two consecutive bytes of $FF. If there are no bad sectors on the diskette the first 2 bytes of the data

Appendix L -- OS DATA BASE VARIABLE FUNCTIONAL DESCRIPTIONS

CENTRAL DATA BASE DESCRIPTION

This appendix provides detailed information for those variables
in the OS data base that can be altered by the user. Remaining
variables are provided narrative descriptions. Information on the
variables is presented in a multiple access scheme: Lookup
tables are referenced to a common set of narratives, that is
itself ordered by function.

Variable descriptions are referenced by a label called a variable
identifier (VID) number. The label comprises a single letter
followed by a number. A different letter is assigned for each
major functional area being described, and the numbers are
assigned sequentially within each functional area. Those
variables that are not considered to be of interest to any user
are flagged with an asterisk (*) after their names. The data base
lookup tables provided are:

1. Functional grouping -- index to the function narrative and
   descriptions of variables, giving VID and variable name.

2. Alphabetic list of names -- giving VID of description.

3. Address ordered list -- giving VID of description.

Item 1, the functional grouping index, starts on the next page;
the other two lookup tables are at the end of Appendix L.

FUNCTIONAL INDEX TO DATA BASE VARIABLE DESCRIPTIONS

A. Memory configuration
    A1 MEMLO
    A2 MEMTOP
    A3 APPMHI
    A4 RAMTOP
    A5 RAMSIZ

B. Text/graphics screen

    Cursor control
        B1 CRSINH
        B2 ROWCRS,COLCRS
        B3 OLDROW,OLDCOL
        B4 TXTROW,TXTCOL

    Screen margins
        B5 LMARGN
        B6 RMARGN

    Color control
        B7 PCOLRO - PCOLR3
        B8 COLORO - COLOR4

    Text scrolling
        B9 SCRFLG*

    Attract mode
        B10 ATRACT
        B11 COLRSH*
        B12 DRKMSK*

    Tabbing
        B13 TABMAP

    Logical text lines
        B14 LOGMAP*
        B15 LOGCOL*

    Split screen

```
     B16 BOTSCR*

FILL/DRAW function
   B17 FILDAT
   B18 FILFLG*
   B19 NEW 5ROW*,NEWCOL*
   B20 HOLD4*




   B21 ROWINC*,COLINC*
   B22 DELTAR*,DELTAC*
   B23 COUNTR*
   B24 ROWAC*,COLAC*
   B25 ENDPT*

Displaying control characters

   Escape (display following control char)
      B26 ESCFLG*

   Display control characters mode
      B27 DSPFLG

Bit mapped graphics
   B28 DMASK*
   B29 SHFAMT*
```

Internal working variables
    B30 HOLD1*
    B31 HOLD2*
    B32 HOLD3*
    B33 TMPCHR*
    B34 DSTAT*
    B35 DINDEX
    B36 SAVMSC
    B37 OLDCHR*
    B38 OLDADR*
    B39 ADRESS*
    B40 MLTTMP/OPNTMP/TOADR*
    B41 SAVADR/FRMADR*
    B42 BUFCNT*
    B43 BUFSTR*
    B44 SWPFLG*
    B45 INSDAT*
    B46 TMPROW*, TMPCOL*
    B47 TMPLBT*
    B48 SUBTMP*
    B49 TINDEX*
    B50 BITMSK*
    B51 LINBUF*
    B52 TXTMSC
    B53 TXTOLD*

Internal character code conversion
    B54 ATACHR
    B55 CHAR*

C. Disk Handler
    C1 BUFADR*
    C2 DSKTIM*

D. Cassette (part in SIO part in Handler)

    Baud rate determination
      D1 CBAUDL*,CBAUDH*
      D2 TIMFLG*
      D3 TIMER1*,TIMER2*
      D4 ADDCOR*
      D5 TEMP1*
      D6 TEMP3*
      D7 SAVIO*

    Cassette mode
      D8 CASFLG*

    Cassette buffer
      D9 CASBUF*
      D10 BLIM*
      D11 BPTR*

    Internal working variables
      D12 FEOF*
      D13 FTYPE*
      D14 WMODE*
      D15 FREQ*

E. Keyboard

    Key reading and debouncing
      E1 CH1*
      E2 KEYDEL*
      E3 CH

Special functions

Start/stop
E4 SSFLAG

[BREAK]
E5 BRKKEY

[SHIFT]/[CONTROL] lock
E6 SHFLOK
E7 HOLDCH*

Autorepeat
E8 SRTIMR*

Inverse video
E9 INVFLG

Console switches ([SELECT],[START],and [OPTION])

F. Printer

printer-buffer

F1 PRNBUF*
F2 PBUFSZ*
F3 PBPNT*

Internal working variables
F4 PTEMP*
F5 PTIMOT*

G. Central I/O routine (CIO)

User call parameters

G1 IOCB
G2 ICHID
G3 ICDNO
G4 ICCOM
G5 ICSTA
G6 ICBAL, ICBAH
G7 ICPTL, ICPTH
G8 ICBLL, ICBLH
G9 ICAX1, ICAX2
G10 ICSPR

Device status
G11 DVSTAT

device table
G12 HATABS

CIO/Handler interface Parameters

    G13 ZIOCB (IOCBAS)
      G14 ICHIDZ
      G15 ICDNOZ
      G16 ICCOMZ
      G17 ICSTAZ
      G18 ICBALZ, ICBALH
      G19 ICPTLZ, ICPTHZ
      G20 ICBLLZ, ICBLHZ
      G21 ICAX1Z, ICAX2Z
      G22 ICSPRZ (ICIDNO, CIOCHR)

  Internal working variables
    G23 ICCOMT*
    G24 ICIDNO*
    G25 CIOCHR*

H. Serial I/O routine (SIO)

  User call parameters

    H1 DCB control block
      H2 DDEVIC
      H3 DUNIT
      H4 DCOMND
      H5 DSTATS
      H6 DBUFLO, DBUFHI
      H7 DTIMLO
      H8 DBYTLO, DBYTHI
      H9 DAUX1, DAUX2

    Bus sound control
      H10 SOUNDR

  Serial bus control

    Retry logic
      H11 CRETRY*
      H12 DRETRY*

    Checksum
      H13 CHKSUM*
      H14 CHKSNT*
      H15 NOCKSM*

Data buffering

    General buffer control

        H16 BUFRLO*,BUFRHI*
        H17 BFENLO*,BFENHI*

    Command frame output buffer
        H18 CDEVIC*
        H19 CCOMND*
        H20 CAUX1*,CAUX2*

    Receive/transmit data buffering
        H21 BUFRFL*
        H22 RECVDN*
        H23 TEMP*
        H24 XMTDON*

    SIO timeout
        H25 TIMFLG*
        H26 CDTMV1*
        H27 CDTMA1*

Internal working variables
    H28 STACKP*
    H29 TSTAT*
    H30 ERRFLG*
    H31 STATUS*
    H32 SSKCTL*

J. ATARI controllers

Joysticks
    J1 STICKO – STICK3
    J2 STRIGO – STRIG3

Paddles
    J3 PADDLO – PADDL7
    J4 PTRIGO – PTRIG7

Paddle controllers
    J8 STICKO – STICK3
    J9 STRIGO – STRIG3

K. Disk file manager
    K1 FMSZPG*
      K2 ZBUFP*
      K3 ZDRVA*
      K4 ZSBA*
      K5 ERRNO*

L. Disk utilities (DOS)
    L1 DSKUTL*

M. Floating point package
    M1 FR0
    M2 FRE*
    M3 FR1
    M4 FR2*
    M5 FRX*
    M6 EEXP*
    M7 NSIGN*
    M8 ESIGN*
    M9 FCHRFLG*
    M10 DIGRT*
    M11 CIX
    M12 INBUFF
    M13 ZTEMP1*
    M14 ZTEMP4*
    M15 ZTEMP3*
    M16 FLPTR
    M17 FPTR2*
    M18 LBPR1*
    M19 LBPR2*
    M20 LBUFF
    M21 PLYARG*
    M22 FPSCR/FSCR*
    M23 FPSCR1/FSCR1*
    M24 DEGFLG/RADFLG*

N. Power-Up and System Reset
    RAM sizing
        N1 RAMLO*,TRAMSZ*
        N2 TSTDAT*

    Diskette/cassette-boot
        N3 DOSINI
        N4 CKEY*
        N5 CASSBT*
        N6 CASINI
        N7 BOOT?*
        N8 DFLAGS*
        N9 DBSECT*
        N10 BOOTAD*

    Environmental control
        N11 COLDST
        N12 DOSVEC

[S RESET]
            N13 WARMST

P. Interrupts
    P1 CRITIC
    P2 POKMSK

    System Timers

      Real-time clock
         P3 RTCLOK

      System Timer 1
         P4 CDTMV1
         P5 CDTMA1

      System Timer 2
         P6 CDTMV2
         P7 CDTMA2

      System Timers 3-5
         P8 CDTMV3,CDTMV4,CDTMV5
         P9 CDTMF3,CDTMF4,CDTMF5

    RAM-interrupt vectors

      NMI-interrupt vectors
         P10 VDSLST
         P11 VVBLKI
         P12 VVBLKD

      IRQ-interrupt vectors
         P13 VIMIRQ
         P14 VPRCED
         P15 VINTER
         P16 VBREAK
         P17 VKEYBD
         P18 VSERIN
         P19 VSEROR
         P20 VSEROC
         P21 VTIMR1,VTIMR2,VTIMR4

    Hardware register updates
         P22 SDMCTL
         P23 SDLSTL,SDLSTH
         P24 GPRIOR
         P25 CHACT
         P26 CHBAS
         P27 PCOLRx,COLORx

         OPERATING SYSTEM CO16555 -- Appendix L

Internal working variable
           P28 INTEMP*

  R. User areas
      R1 (unlabeled)
      R2 USAREA

This appendix contains descriptions of many of the data base
variables; descriptions are included for all of the user-
accessible variables and for some of the "internal" variables as
well. Those variables that are not considered to be normally of
interest to any user are flagged with an asterisk (*) after their
names; the other variables can be of interest to one or more of
the following classes of users:

          o End user.
          o Game developer.
          o Applications programmer.
          o System utility writer.
          o Language processor developer.
          o Device Handler Writer.

Each variable is specified by its system equate file name
followed by its address (in hex) and the number of bytes reserved
in the data base (in decimal), in the following form:

      <name> [<address>,<size>]

For example:

      MEMLO [02E7,2]

Note that most word (2 byte) variables are ordered with the least
significant byte at the lower address.

A. MEMORY CONFIGURATION

See Section 4 for a general discussion of memory dynamics and
section 7 for details of system initialization.

A1   MEMLO [02E7,2] -- User-free memory low address

MEMLO contains the address of the first location in the free
memory region. The value is established by the OS during power-up
and system reset initialization and is never altered by the OS
thereafter.

A2   MEMTOP [02E5,2] -- User-free memory high address

MEMTOP contains the address of the first non-useable memory
location above the free memory region. The value is established
by the OS during power-up and system reset initialization; and
then is re-established whenever the display is opened, based upon
the requirements of the selected graphics mode.

A3   APPMHI [000E,2] -- User-free memory screen lower limit

APPMHI is a user-controlled variable that contains the address
within the free memory region below which the Display Handler cannot
go in setting up a display screen. This variable is
initialized to zero by the OS at power-up.

A4   RAMTOP* [006A,1] -- Display Handler top of RAM address (MSB)

RAMTOP permanently retains the RAM top address that was contained
in TRAMSZ (as described in N1) for the Display Handler's use. The
value is set up as part of Handler initialization.

A5   RAMSIZ [02E4,1] -- Top of RAM address (MSB only)

RAMSIZ permanently retains the RAM top address that was contained
in TRAMSZ (as described in N1).

B. TEXT/GRAPHICS SCREEN

See Section 5 for a discussion of the text and graphics screens and their Handlers.

Cursor Control

For the text screen and split-screen text window there is a visible cursor on the screen which shows the position of the next input or output operation. The cursor is represented by inversing the video of the character upon which it resides; but the cursor can be made invisible, at the user's option. The graphics screen always has an invisible cursor.

The cursor position is sensed by examining data base variables and can be moved by altering those same variables; in addition, when using the Screen Editor, there are cursor movement control codes that can be sent as data (as explained in Section 5).

B1 CRSINH [02F0, 1] -- Cursor display inhibit flag

When CRSINH is zero, all outputs to the text screen will be followed by a visible cursor (inversed character); and when CRSINH is nonzero, no visible cursor will be generated.

CRSINH is set to zero by power-up, the [SYSTEM.RESET] or [BREAK] keys or an OPEN command to the Display Handler or Screen Editor.

Note that altering CRSINH does not cause the visible cursor to change states until the next output to the screen; if an immediate change to the cursor state is desired, without altering the screen data, follow the CRSINH change with the output of CURSOR UP, CURSOR DOWN, or some other innocuous sequence.

B2 ROWCRS [0054, 1] and COLCRS [0055, 2] -- Current cursor position

ROWCRS and COLCRS define the cursor location (row and column, respectively) for the next data element to be read from or written to the main screen segment. When in split-screen mode, the variables TXTROW and TXTCOL define the cursor for the text window at the bottom of the screen as explained in B4 below.

The row and column numbering start with the value zero, and increase in increments of one to the number of rows or columns minus 1; with the upper left corner of the screen being the origin (0,0).

ROWCRS is a single-byte variable with a maximum allowable value of 191 (screen modes 8-11); COLCRS is a 2-byte variable with a maximum allowable value of 319 (screen mode 8).

B3   OLDROW [005A,1] and OLDCOL [005B,2] -- Prior cursor position

OLDROW and OLDCOL are updated from ROWCRS and COLCRS before every
operation. The variables are used only for the DRAW and FILL
operations.

B4   TXTROW [0290,1] and TXTCOL [0291,2] -- Split-screen text cursor
position

TXTROW and TXTCOL define the cursor location (row and column,
respectively) for the next data element to be read from or
written to the split-screen text window.

The row and column numbering start with the value zero, and
increase in increments of one to 3 and 39, respectively; with the
upper left corner of the split-screen text window being the origin
(0,0).

Screen Margins

The text screen and split-screen text window have user-alterable
left and right margins that define the normal domain of the text
cursor.

B5   LMARGN [0052,1] -- Text column left margin

LMARGN contains the column number (0-39) of the text screen left
margin; the text cursor will remain on or to the right of the
left margin as a result of all operations, unless the cursor
column variable is directly updated by the user (see B2 and B4
above). The default value for LMARGN is 2 and is established upon
power-up or system reset.

B6   RMARGN [0053,1] -- Text column right margin

RMARGN contains the column number (0-39) of the text screen right
margin; the text cursor will remain on or to the left of the
right margin as a result of all operations, unless the cursor
column variable is directly updated by the user (see B2 and B4
above). The default value for RMARGN is 39 and is established
upon power-up or system reset.

Color Control

As part of the stage 2 VBLANK process (see Section 6), the values of
nine data base variables are stored in corresponding hardware
color control registers. The color registers are divided into two
groups: the player/missile colors and the playfield colors. The
playfield color registers are utilized by the different screen modes
as shown in Appendix H. The player/missile color registers are not
used by the standard OS.

B7 PCOLRO - PCOLR3 [02C0,4] -- Player/missile graphics colors

Each color variable is stored in the corresponding hardware
register as shown below:

        PCOLRO [02C0]   COLPMO [D012]
        PCOLR1 [02C1]   COLPM1 [D013]
        PCOLR2 [02C2]   COLPM2 [D014]
        PCOLR3 [02C3]   COLPM3 [D015]

Each color variable has the format shown below:

        7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+
        | color | lum |x|
        +-+-+-+-+-+-+-+-+

See Appendix H for information regarding the color and luminance
field values.

B8 COLORO - COLOR4 [02C5,5] -- Playfield colors

Each color variable is stored in the corresponding hardware
register as shown below:

        COLORO [02C4]   COLPFO [D016]
        COLOR1 [02C5]   COLPF1 [D017]
        COLOR2 [02C6]   COLPF2 [D018]
        COLOR3 [02C7]   COLPF3 [D019]
        COLOR4 [02C8]   COLBK [D01A]

Each color variable has the format shown below:

        7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+
        | color | lum |x|
        +-+-+-+-+-+-+-+-+

See Appendix H for information regarding the color and luminance
field values.

Text Scrolling

The text screen or split-screen text window "scrolls" upward whenever one of the two conditions shown below occurs:

o   A text line at the bottom row of the screen extends past the right margin.

o   A text line at the bottom row of the screen is terminated by an EOL.

Scrolling has the effect of removing the entire logical line that starts at the top of the screen and then moving all subsequent lines upward to fill in the void. The cursor will also move upward if the logical line deleted exceeds one physical line.

B9   SCRFLG* [02BB,1] -- Scroll flag

SCRFLG is a working variable that counts the number of physical lines minus 1 that were deleted from the top of the screen; since a logical line ranges in size from 1 to 3, SCRFLG ranges from 0 to 2.

Attract Mode

Attract mode is a mechanism that protects the television screen from having patterns "burned into" the phosphors due to a fixed display being left on the screen for extended periods of time. When the computer is left unattended for more than 9 minutes, the color intensities are limited to 50 percent of maximum and the hues are continually varied every 8.3 seconds. Pressing any keyboard data key will be sufficient to remove the attract mode for 9 more minutes.

As part of the stage 2 VBLANK process, the color registers from the data base are sent to the corresponding hardware color registers; before they are sent, they undergo the following transformation:

    hardware register = database variable XOR COLRSH AND DRKMSK

Normally COLRSH = $00 and DRKMSK = $FE, thus making the above calculation a null operation; however, once attract mode becomes active, COLRSH = the content of RTCLOK+1 and DRKMSK = $F6, that has the effect of modifying all of the colors and keeping their luminance always below the 50 percent level.

Since RTCLOK+1 is incremented every 256/60 of a second and since the least significant bit of COLRSH is of no consequence, a

color/lum change will be effected every 8.3 seconds (512/60).

B10 ATRACT [004D,1] -- Attract mode timer and flag

ATRACT is the timer (and flag) that controls the initiation and
termination of attract mode. Whenever a keyboard key is pressed,
the keyboard IRQ service routine sets ATRACT to zero, thus
terminating attract mode; the [BREAK] key logic behaves
accordingly. As part of the stage 1 VBLANK process, ATRACT is
incremented every 4 seconds; if the value exceeds 127 (after 9
minutes without keyboard activity), the value of ATRACT will
be set to $FE and will retain that value until attract mode is
terminated.

Since the attract mode is prevented and terminated by the OS
based only upon keyboard activity, some users can want to reset
ATRACT based upon Atari-controller event detection,
user-controlled Serial I/O bus activity or any other signs of
life.

B11 COLRSH* [004F,1] -- Color shift mask

COLRSH has the value $00 when attract mode is inactive, thus
effecting no change to the screen colors; when attract mode is
active, COLRSH contains the current value of the timer variable
middle digit (RTCLOK+1).

B12 DRKMSK* [004E,1] -- Dark (luminance) mask

DRKMSK has the value $FE when attract mode is inactive, which does not
alter the luminance; and has the value $F6 when attract mode
is active, which forces the most significant bit of the luminance
field to zero, thus guaranteeing that the luminance will never
exceed 50 percent.

Tabbing

See Section 5 for a discussion of the use of tabs in conjunction
with the Screen Editor.

B13 TABMAP [02A3,15] -- Tab stop setting map

The tab settings are retained in a 15-byte (120 bit) map, where a
bit value of 1 indicates a tab setting; the diagram below shows
the mapping of the individual bits to tab positions.

```
       7   6   5   4   3   2   1   0
     +---+---+---+---+---+---+---+---+
     ! 0 ! 1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7 !   TABMAP+0
     +---+---+---+---+---+---+---+---+
     ! 8 ! 9 ! 10! 11! 12! 13! 14! 15!       +1
     +---+---+---+---+---+---+---+---+
     !                               !
     =                               =
     !                               !
     +---+---+---+---+---+---+---+---+
     !112!113!114!115!116!117!118!119!      +14
     +---+---+---+---+---+---+---+---+
```

Whenever the Display Handler or Screen Editor is opened, this map
is initialized to contain the value of $01 in every byte, thus
providing the default tab stops at 7, 15, 23, etc.

Logical Text Lines

The text screen is invisibly divided into logical lines of text,
each comprising from one to three physical lines of text. The
screen is initialized to 24 logical lines of one physical line
each; but data entry and/or data insertion can increase the size
of a logical line to two or three physical lines.

B14 LOGMAP* [02B2,4] -- Logical line starting row map

The beginning physical line number for each logical line on the
screen is retained in a four byte (32 bit) map, where a bit value
of one indicates the start of a logical line; the diagram below
shows the mapping of the individual bits to physical line (row)
numbers.

```
     7  6  5  4  3  2  1  0
   +--+--+--+--+--+--+--+--+
   ! 0! 1! 2! 3! 4! 5! 6! 7!     LOGMAP+0
   +--+--+--+--+--+--+--+--+
   ! 8! 9!10!11!12!13!14!15!       +1
   +--+--+--+--+--+--+--+--+
   !16!17!18!19!20!21!22!23!       +2
   +--+--+--+--+--+--+--+--+
   !  !  !  !  !  !  !  !  !       +3
   +--+--+--+--+--+--+--+--+
```

The map bits are all set to 1 whenever the text screen is
opened or cleared. From that point, the map is updated as
logical lines are entered, edited and deleted from the screen.

B15 LOGCOL* [0063,1] -- Cursor/logical line column number

LOGCOL contains the logical-line column number for the current
cursor position; note that a logical line can comprise up to
three physical lines. This variable is for the internal use of
the Display Handler.

Split Screen

The Display Handler and Screen Editor together support the
operation of a split-screen mode (see Section 5) in which the main
portion of the screen is in one of the graphics modes and is
controlled by the Display Handler, and there are 4 physical lines in
the text window at the bottom of the screen which is controlled by the
Screen Editor.

B16 BOTSCR* [02BF,1] -- Text screen lines count

BOTSCR contains the number of lines of text for the current
screen: 24 for mode 0 or 4 for a split-screen mode. The Handler
also uses this variable as an indication of the split-screen
status; tests are made for the specific values 4 and 24.

DRAW/FILL Function

The DRAW function line drawing algorithm is shown below
translated to the PASCAL language from assembly language.

```
    NEWROW := ROWCRS; NEWCOL := COLCRS;

    DELTAR := ABS (NEWROW-OLDROW);
    ROWINC := SIGN (NEWROW-OLDROW); { +1 or -1 }

    DELTAC := ABS (NEWCOL-OLDCOL);
    COLINC := SIGN (NEWCOL-OLDCOL); { +1 or -1 }

    ROWAC := 0; COLAC := 0;
    ROWCRS := OLDROW; COLCRS := OLDCOL;

    COUNTR := MAX (DELTAC,DELTAR);
    ENDPT := COUNTR;
    IF COUNTR = DELTAC
        THEN ROWAC := ENDPT DIV 2
        ELSE COLAC := ENDPT DIV 2;

    WHILE COUNTR > 0 DO
        BEGIN
```

```
            ROWAC := ROWAC + DELTAR;
            IF ROWAC >= ENDPT
                THEN
                    BEGIN
                        ROWAC := ROWAC - ENDPT;
                        ROWCRS := ROWCRS + ROWINC
                    END;

            COLAC := COLAC + DELTAC;
            IF COLAC >= ENDPT
                THEN
                    BEGIN
                        COLAC := COLAC - ENDPT;
                        COLCRS := COLCRS + COLINC
                    END;

            PLOT_POINT; { point defined by ROWCRS and COLCRS }

            IF FILFLG <> O THEN FILL_LINE;

            COUNTR := COUNTR - 1

        END;
```

The FILL function algorithm (FILL_LINE above) is described briefly in Section 5.

B17 FILDAT [O2FD,1] -- Fill data

FILLDAT contains the fill region data value as part of the calling sequence for a FILL command as described in Section 5.

B18 FILFLG* [O2B7,1] -- Fill flag

FILFLG indicates to the shared code within the Display Handler whether the current operation is FILL (FILFLG <> O) or DRAW (FILFLG = O).

B19 NEWROW* [OO6O,1] and NEWCOL* [OO61,2] -- Destination point

NEWROW and NEWCOL are initialized to the values in ROWCRS and COLCRS, which represent the destination endpoint of the DRAW/FILL command. This is done so that ROWCRS and COLCRS can be altered during the performance of the command.

B20 HOLD4* [O2BC,1] -- Temporary storage

HOLD4 is used to save and restore the value in ATACHR during the FILL process; ATACHR is temporarily set to the value in FILDAT to accomplish the filling portion of the command.

B21 ROWINC* [0079,1] and COLINC* [007A,1] -- Row/column increment/decrement

ROWINC and COLINC are the row and column increment values; they are each set to +1 or -1 to control the basic direction of line drawing. ROWINC and COLINC represent the signs of NEWROW - ROWCRS and NEWCOL - COLCRS, respectively.

B22 DELTAR* [0076,1] and DELTAC* [0077,2] -- Delta row and delta column

DELTAR and DELTAC contain the absolute values of NEWROW - ROWCRS and NEWCOL - COLCRS, respectively; together with ROWINC and COLINC, they define the slope of the line to be drawn.

B23 COUNTR* [007E,2] -- Draw iteration count

COUNTR initially contains the larger of DELTAR and DELTAC, that is the number of iterations required to generate the desired line. COUNTR is then decremented after every point on the line is plotted, until it reaches a value of zero.

B24 ROWAC* [0070,2] and COLAC* [0072,2] -- Accumulators

ROWAC and COLAC are working accumulators that control the row-and column-point plotting and increment (or decrement) function.

B25 ENDPT* [0074,2] -- Line length

ENDPT contains the larger of DELTAR and DELTAC, and is used in conjunction with ROWAC/COLAC and DELTAR/DELTAC to control the plotting of line points.

Displaying Control Characters

Often it is useful to have ATASCII control codes (such as CLEAR, CURSOR UP, etc). displayed in their graphic forms instead of having them perform their control function. This display capability is provided in two forms when outputting to the Screen Editor: 1) a data content form in which a special character (ESC) precedes each control character to be displayed and 2) a mode control form.

Escape (Display Following Control Character)

Whenever an ESC character is detected by the Screen Editor, the
next character following this code is displayed as data, even if
it would normally be treated as a control code; the EOL code is
the sole exception. It is always treated as a control code. The
sequence ESC ESC will cause the second ESC character to be
displayed.

B26   ESCFLG* [02A2,1] -- Escape flag

ESCFLG is used by the Screen Editor to control the escape
sequence function; the flag is set (to $80) by the detection of
an ESC character ($1B) in the data stream and is reset (to 0)
following the output of the next character.

Display Control Characters Mode

When it is desired to display ATASCII control codes other than
EOL in their graphics form, but not have an ESC character
associated with each control code, a display mode can be
established by setting a flag in the data base. This capability
is used by language processors when displaying high-level
language statements, that can contain control codes as data
elements.

B27   DSPFLG [02FE,1] -- Display control characters flag

When DSPFLG is nonzero, ATASCII control codes other than EOL are
treated as data and displayed on the screen when output to the
Screen Editor. When DSPFLG is zero, ATASCII control codes are
processed normally.

DSPFLG is set to zero by Power-up and [SYSTEM.RESET].

Bit-Mapped Graphics

A number of temporary variables are used by the Display Handler
when handling data elements (pixels) going to or from the screen;
of interest here are those variables that are used to control the
packing and unpacking of graphics data, where a memory byte
typically contains more than one data element (for example,
screen mode 8 contains 8 pixels per memory byte).

B28   DMASK* [02A0,1] -- Pixel location mask

DMASK is a mask that contains zeros for all bits that do not
correspond to the specific pixel to be operated upon, and
1's for all bits that do correspond. DMASK can contain
the values shown below in binary notation:

        11111111   --   screen modes 1 and 2; one pixel per byte.

        11110000   --   screen modes 9-11; two pixels per byte.
        00001111

        11000000   --   screen modes 3, 5 and 7; four pixels per byte.
        00110000
        00001100
        00000011

        10000000   --   screen modes 4, 6 and 8; eight pixels per byte.
        01000000

        00000010
        00000001

B29   SHFAMT* [006F,1] -- Pixel justification

SHFAMT indicates the amount to shift the right-justified pixel
data on output, or the amount to shift the input data to right
justify it on input. The value is always the same as for DMASK
prior to the justification process.

Internal Working Variables

B30   HOLD1* [0051,1] -- Temporary storage

B31   HOLD2* [029F,1] -- Temporary storage

B32   HOLD3* [029D,1] -- Temporary storage

B33   TMPCHR* [0050,1] -- Temporary storage

B34   DSTAT* [004C,1] -- Display status

B35   DINDEX [0057,1] -- Display mode

DINDEX contains the current screen mode obtained from the low
order four bits of the most recent OPEN AUX1 byte.

B36   SAVMSC [0058,2] -- Screen Memory Address

SAVMSC contains the lowest address of the screen data region; the
data at that address is displayed at the upper left corner of the
screen.

B37 OLDCHR* [005D,1] -- Cursor character save/restore

OLDCHR retains the value of the character under the visible text cursor; this variable is used to restore the original character value when the cursor is moved.

B38 OLDADR* [005E,2] -- Cursor memory address

OLDADR retains the memory address of the current visible text cursor location; this variable is used in conjunction with OLDCHR (B37) to restore the original character value when the cursor is moved.

B39 ADRESS* [0064,2] -- Temporary storage

B40 MLTTMP/OPNTMP/TOADR* [0066,2] -- Temporary storage

B41 SAVADR/FRMADR* [0068,2] -- Temporary storage

B42 BUFCNT* [006B,1] -- Screen Editor current logical line size

B43 BUFSTR* [006C,2] -- Temporary storage

B44 SWPFLG* [007B,1] -- Split-screen cursor control

In split-screen mode, the graphics cursor data and the text window cursor data are frequently swapped as shown below in order to get the variables associated with the region being accessed into the ROWCRS-OLDADR variables.

```
    ROWCRS B2 ------- TXTROW B4
    COLCRS B2 ------- TXTCOL B4
    DINDEX B35 ------ TINDEX B49
    SAVMSC B36 ------ TXTMSC B52
    OLDROW B3 ------- TXTOLD B53
    OLDCOL B3 -------    "      "
    OLDCHR B37 ------    "      "
    OLDADR B38 ------    "      "
```

SWPFLG is used to keep track of what data set is currently in the ROWCRS-OLDADR region; SWPFLG is equal to $FF when split-screen text window cursor data is in the main region, otherwise SWPFLG is equal to 0.

B45 INSDAT* [007D,1] -- Temporary storage

B46   TMPROW* [02B8,1] and TMPCOL* [02B9,2] -- Temporary storage

B47   TMPLBT* [02A1,1] -- Temporary storage

B48   SUBTMP* [029E,1] -- Temporary storage

B49   TINDEX* [0293,1] -- Split screen text window screen mode

TINDEX is the split-screen text window equivalent of DINDEX and is
always equal to zero when SWPFLG is equal to zero (see B44).

B50   BITMSK* [006E,1] -- Temporary storage

B51   LINBUF* [0247,40] -- Physical line buffer

LINBUF is used to temporarily buffer one physical line of text
when the Screen Editor is moving screen data.

B52   TXTMSC [0294,2] -- Split screen memory address

TXTMSC is the split-screen text window version of SAVMSC (B36).

See B44 for more information.

B53   TXTOLD* [0296,6] -- Split screen cursor data

See B44 for more information.

Internal Character Code Conversion

Two variables are used to retain the current character being
processed (for both reading and writing); ATACHR contains the
value passed to or from CIO, and CHAR contains the internal code
corresponding to the value in ATACHR. Because the hardware does
not interpret ATASCII characters directly, the transformations
shown below are applied to all text data read and written:

ATASCII CODE	INTERNAL CODE
00-1F	40-5F
20-3F	00-1F
40-5F	20-3F
60-7F	60-7F
80-9F	C0-DF

```
            AO-BF                    BO-9F
            CO-DF                    AO-BF
            EO-FF                    EO-FF
```

See P26 for more information.

B54 ATACHR [02FB,1] -- Last ATASCII character or plot point

ATACHR contains the ATASCII value for the most recent character
read or written, or the value of the graphics point. This
variable can also be considered to be a parameter of the
FILL/DRAW commands, as the value in ATACHR will determine the
line color when a DRAW or FILL is performed.

B55 CHAR* [02FA,1] -- Internal character code

CHAR contains the internal code value for the most recent
character read or written.

C. DISKETTE HANDLER

See Section 5 for a discussion of the resident Diskette Handler.

C1 BUFADR* [0015,2] -- Data buffer pointer

BUFADR acts as temporary page zero pointer to the current
diskette buffer.

C2 DSKTIM* [0246,1] -- Disk format operation timeout time

DSKTIM contains the timeout value for SIO calling sequence
variable DTIMLO (see Section 9). DSKTIM is set to 160 (which
represents a 171-second timeout) at initialization time, and is
updated after each diskette status request operation. It contains the
value returned in the third byte of the status frame (see Section
5). Note that all diskette operations other than format have a
fixed (7) second timeout, established by the Diskette Handler.

D. CASSETTE

See Section 5 for a general description of the Cassette Handler. The
cassette uses the Serial I/O bus hardware, but does not conform with
the Serial I/O bus protocol as defined in Section 9. Hence, the Serial

I/O utility (SIO) has cassette specific code within it. Some variables
in this subsection are utilized by SIO and some by the Cassette
Handler.

Baud Rate Determination

The input baud rate is assumed to be a nominal 600 baud, but will
be adjusted, if necessary, by the SIO routine to account for
drive-motor variations, stretched tape, etc. The beginning of
every cassette record contains a pattern of alternating 1's and
zeros that is used solely for speed correction; by measuring the
time to read a fixed number of bits, the true-receive baud rate
is determined and the hardware adjusted accordingly. Input baud
rates ranging from 318 to 1407 baud can theoretically be handled
using this technique.

The input baud rate is adjusted by setting the POKEY counter that
controls the bit sampling period.

D1   CBAUDL* [02EE,1] and CBAUDH* [02EF,1] -- Cassette baud rate

Initialized to 05CC hex, which represents a nominal 600 baud.
After baud rate calculation, these variables will contain POKEY
counter values for the corrected baud rate.

D2   TIMFLG* [0317,1] -- Baud rate determination timeout flag

TIMFLG is used by SIO to timeout an unsuccessful baud rate
determination. The flag is initially set to 1, and if it attains a
value of zero (after 2 seconds) before the first byte of the cassette
record has been read, the operation will be aborted. See also H24.

D3   TIMER1* [030C,2] and TIMER2* [0310,2] -- Baud rate timers

These timers contain reference times for the beginning and end of
the fixed bit pattern receive period. The first byte of each
timer contains the then current vertical line counter value read
from ANTIC, and the second byte of each timer contains the then
current value of the least significant byte of the OS real time
clock (RTCLOK+2).

The difference between the timers is converted to raster pair
counts and is then used to perform a table lookup with
interpolation to determine the new values for CBAUDL and CBAUDH.

D4   ADDCOR* [030E,1] -- Interpolation adjustment variable

ADDCOR is a temporary variable used for the interpolation
calculation of the above computation.

D5   TEMP1* [0312,2] -- Temporary storage

D6   TEMP3* [0315,1] -- Temporary storage

D7   SAVIO* [0316,1] -- Serial in data detect

SAVIO is used to retain the state of SKSTAT [D20F] bit 4 (serial
data in); it is used to detect (and is updated after) every bit
arrival.

Cassette Mode

D8   CASFLG* [030F,1] -- Cassette I/O flag

CASFLG is used internally by SIO to control the program flow
through shared code. A value of zero indicates that the current
operation is a standard Serial I/O bus operation, and a nonzero
value indicates a cassette operation.

Cassette Buffer

D9   CASBUF* [03FD,131] -- Cassette record buffer

CASBUF is the buffer used by the Cassette Handler for the packing
and unpacking of cassette-record data, and by the initialization
cassette-boot logic. The format for the standard cassette record
in the buffer is shown below:

```
        7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+
        !0 1 0 1 0 1 0 1!     CASBUF+0
        +-+-+-+-+-+-+-+-+
        !0 1 0 1 0 1 0 1!        +1
        +-+-+-+-+-+-+-+-+
        ! control byte  !        +2
        +-+-+-+-+-+-+-+-+
        !      128      !        +3
        =      data      =
        !     bytes     !        +130
        +-+-+-+-+-+-+-+-+
```

See Section 5 for an explanation of the standard cassette-record
format.

D10   BLIM* [028A,1] -- Cassette record data size

BLIM contains the count of the number of data bytes in the
current cassette record being read. BLIM will have a value
ranging from 1 to 128, depending upon the record control byte as
explained in Section 5.

D11   BPTR* [003D,1] -- Cassette-record data index

BPTR contains an index into the data portion of the cassette
record being read or written. The value will range from 0 to the
then current value of BLIM. When BPTR equals BLIM then the buffer
(CASBUF) is full if writing or empty if reading.

Internal Working Variables

D12   FEOF* [003F,1] -- Cassette end-of-file flag

FEOF is used by the Cassette Handler to flag the detection of an
end of file condition (control byte = $FE). FEOF equal to zero
indicates that an EOF has not yet been detected, and a nonzero
value indicates that an EOF has been detected. The flag is reset
at every OPEN.

D13   FTYPE* [003E,1] -- Interrecord gap type

FTYPE is a copy of ICAX2Z from the OPEN command and indicates the
type of interrecord gap selected; a positive value indicates
normal record gaps, and a negative value indicates continuous
mode gaps.

D14   WMODE* [0289,1] -- Cassette read/write mode flag

WMODE is used by the Cassette Handler to indicate whether the
current operation is a read or write operation; a value of zero
indicates read, and a value of $80 indicates write.

D15   FREQ* [0040,1] -- Beep count

FREQ is used to retain and count the number of beeps requested of
the BEEP routine by the Cassette Handler during the OPEN command
process.

E. KEYBOARD

See Section 5 for a general description of the Keyboard Handler.

Key Reading and Debouncing

The console key code register is read in response to an IRQ
interrupt that is generated whenever a key stroke is detected by
the hardware. The key code is compared with the prior key code
accepted (CH1); if the codes are not identical, then the new code
is accepted and stored in the key code FIFO (CH) and in the prior
key code variable (CH1). If the codes are identical, then the new
code is accepted only if a suitable key debounce delay has
transpired since the prior value was accepted.

If the key code read and accepted is the code for [CTRL] 1, then
the display start/stop flag (SSFLAG) is complemented and the
value is not stored in the key code FIFO (CH).

In addition to the reading of the key data, SRTIMR is set to $30
for all interrupts received (see E8), and ATRACT is set to 0
whenever a new code is accepted (see B10).

The Keyboard Handler obtains all key data from CH; whenever a
code is extracted from that 1-byte FIFO, the Handler stores a
value of $FF to the FIFO to indicate that the code has been read.
See Section 5 for further discussion of the Keyboard Handler's
processing of the key codes.

E1   CH1* [02F2,1] -- Prior keyboard character code.

CH1 contains the key code value of the key most recently read and
accepted.

E2   KEYDEL* [02F1,1] -- Debounce delay timer.

KEYDEL is set to a value of 3 whenever a key code is accepted,
and is decremented every 60th of a second by the stage 2 VBLANK
process (until it reaches zero).

E3   CH [02FC,1] -- Keyboard character code FIFO.

CH is a 1-byte FIFO that contains either the value of the most
recently read and accepted key code or the value $FF (which
indicates that the FIFO is empty). The FIFO is normally read by
the Keyboard Handler, but can be read by a user program.

Key data can also be stored into CH by the Autorepeat logic as
explained in the discussion relating to E8.

Special Functions

Start/Stop

Display Handler and Screen Editor output to the text or graphics
mode screen can be stopped and started (without losing any of the
output data) through the use of the [CTRL] 1 key combination.
Each key depression toggles a flag that is monitored by the above
mentioned Handlers. When the flag is nonzero, the handlers wait
for it to go to zero before continuing any output.

E4   SSFLAG [02FF,1] -- Start/stop flag

The flag is normally zero, indicating that screen output is not
to be stopped. The flag is complemented by every occurrence of
the [CTRL] 1 key combination by the keyboard IRQ service routine.

The flag is set to zero upon power-up, [SYSTEM.RESET] or [BREAK]
key processing.

[BREAK] Key

E5   BRKKEY [0011,1] -- [BREAK] key flag

BRKKEY is used to indicate that the [BREAK] key has been pressed.
The value is normally nonzero and is set to zero whenever the
[BREAK] key is pressed. The code that detects and processes the
[BREAK] condition (flag = 0) should set the flag nonzero again.

BRKKEY is monitored by the following OS routines: Keyboard
Handler, Display Handler, Screen Editor, Cassette Handler, xx?
The detection of a [BREAK] condition during an I/O operation
will cause the operation to be aborted and a status of $80 to be
returned to the user.

The flag is set to nonzero upon Power-up, [SYSTEM.RESET] or upon
aborting a pending I/O operation.

[SHIFT]/[CONTROL] Lock

The keyboard control has three different modes for code
generation that apply to the alphabetic keys A through Z:
1) normal, 2) caps lock, and 3) control lock.

In normal mode, all unmodified alphabetic character keys generate the lowercase letter ATASCII code ($61-7A).

In caps lock mode, all unmodified alphabetic character keys generate the uppercase letter ATASCII code ($41-5A).

In control lock mode, all unmodified alphabetic character keys generate the control letter ATASCII code ($01-1A).

In all three modes, any alphabetic character key that is modified (by being pressed in conjunction with the [SHIFT] or [CTRL] key) will generate the desired modified code.

E6   SHFLOK [02BE,1] -- Shift/control lock control flag

SHFLOK normally has one of three values:

    $00 = normal mode (no locks in effect).
    $40 = caps lock.
    $80 = control lock.

SHFLOK is set to $40 upon Power-up and [SYSTEM.RESET] and is modified thereafter by the OS only when the [CAPS.LOWER] key is pressed (either by itself or in conjunction with the [SHIFT] or [CTRL] key).

E7   HOLDCH* [007C,1] -- Character holding variable

HOLDCH is used to retain the current character value prior to the [SHIFT]/[CONTROL] logic process.

Autorepeat

The Autorepeat feature responds to the continuous depression of a keyboard key by replicating the key code 10 times per second, after an initial 1/2 second delay. The timer variable SRTIMR is used to control both the initial delay and the repeat rate.

Whenever SRTIMR is equal to zero and a key is being held down, the value of the key code is stored in the key code FIFO (CH). This logic is part of the stage 2 VBLANK process.

E8   SRTIMR* [022B,1] -- Autorepeat timer

SRTIMR is controlled by two independent processes: 1) the keyboard IRQ service routine, which establishes the initial delay value and 2) the stage 2 VBLANK routine that establishes the repeat rate, decrements the timer and implements the auto repeat logic.

Inverse Video Control

The Keyboard Handler allows the direct generation of more than
half of the 256 ATASCII codes; but codes $80-9A and codes $A0- FC
can be generated only with the "inverse video mode" active. The
ATARI key acts as an on/off toggle for this mode, and all
characters (except for screen editing control characters) will be
subject to inversion when the mode is active.

E9   INVFLG [02B6,1] -- Inverse video flag

INVFLG is normally zero, indicating that normal video ATASCII
codes (bit 7 = 0) are to be generated from keystrokes; whenever INVFLG
is nonzero, inverse video ATASCII codes (bit 7 = 1) will be generated.
The special control codes are exempt from this bit manipulation.

INVFLG is set to zero by power-up and system reset.

The Keyboard Handler inverts bit 7 of INVFLG whenever the ATARI key
is pressed; the lower order bits are not altered and are assumed to be
zero.

The Keyboard Handler's "exclusive or's" (XOR's) the ATASCII key data
with the value in INVFLG at all times; the normal values of $00 and
$80 thus lead to control of the inverse video bit (bit 7).

Console Keys: [SELECT],[START], and [OPTION]

The console keys are sensed directly from the hardware
register CONSOL [D01F]; see the ATARI Home Computer
Hardware Manual for details.

F. PRINTER

See Section 5 for a general description of the Printer Handler.

Printer-Buffer

F1   PRNBUF* [03C0,40] -- Printer-record buffer

PRNBUF is the buffer used by the Printer Handler for packing printer
data to be sent to the device controller. The buffer is 40 bytes long

and contains nothing but printer data.

F2   PBUFSZ* [001E,1] -- Printer-record size

PBUFSZ contains the size of the Printer-record for the current mode selected; the modes and respective sizes (in decimal bytes) are shown below:

        Normal   40
        Double width   20 (not currently supported by the device)
        Sideways   29

        Status request 4

F3   PBPNT* [001D,1] -- Printer-buffer index

PBPNT contains the current index to the Printer-buffer. PBPNT ranges in value from zero to the value of PBUFSZ.

Internal Working Variables

F4   PTEMP* [001F,1] -- Printer Handler temporary data save

PTEMP is used by the Printer Handler to temporarily save the value of a character to be output to the printer.

F5   PTIMOT* [001C,1] -- Printer timeout value

PTIMOT contains the timeout value for SIO calling sequence variable DTIMLO (see Section 9); PTIMOT is set to 30 (which represents a 32 second timeout) at intialization time, and is updated after each printer status request operation to contain the value returned in the third byte of the status frame (see Section 5).

G. CENTRAL I/O ROUTINE (CIO)

See Section 5 for a description of the Central I/O Utility.

User Call Parameters

CIO call parameters are passed primarily through an I/O Control
Block (IOCB); although additional device status information can be
returned in DVSTAT, and Handler information is obtained from the
device table (HATABS).

I/O Control Block

IOCB is the name applied collectively to the 16 bytes associated
with each of the 8 provided control structures; see Section 5.

G1   IOCB [0340,16] -- I/O Control Block

The label IOCB is the location of the first byte of the first IOCB in
the data base. For VIDs G2 through G10, the addresses given are for
IOCB #0 only, the addresses for all of the IOCB's are shown below:

                0340-034F   IOCB #0
                0350-035F   IOCB #1
                0360-036F   IOCB #2
                0370-037F   IOCB #3
                0380-038F   IOCB #4
                0390-039F   IOCB #5
                03A0-03AF   IOCB #6
                03B0-03BF   IOCB #7

G2   ICHID [0340,1] -- Handler ID

See Section 5. Initialized to $FF at power-up and system reset.

G3   ICDNO [0341,1] -- Device number

See Section 5.

G4   ICCOM [0342,1] -- Command byte

See Section 5.

G5   ICSTA [0343,1] -- Status

See Section 5.

G6   ICBAL, ICBAH [0344,2] -- Buffer address

See Section 5.

G7 ICPTL, ICPTH [0346,2] -- PUT BYTE vector

See Section 5. Initialized to point to CIO's "IOCB not OPEN" routine
at power-up and system reset.

G8 ICBLL, ICBLH [0348,2] -- Buffer length / byte count

See Section 5.

G9 ICAX1, ICAX2 [034A,2] -- Auxiliary information

See Section 5.

G10 ICSPR [034C,4] -- Spare bytes for Handler use

There is no fixed assignment of these four bytes; the Handler
associated with an IOCB can or may not use these bytes.

Device Status

G11 DVSTAT [02EA,4] -- Device status

See Section 5 for a discussion of the GET STATUS command.

Device Table

G12 HATABS [031A,38] -- Device table

See Section 9 for a description of the device table.

CIO/Handler Interface Parameters

Communication between CIO and a Handler is accomplished using the
6502 machine registers, and a data structure called the Zero-page
IOCB (ZIOCB). The ZIOCB is essentially a copy of the particular
IOCB being used for the current operation.

G13   ZIOCB (IOCBAS) [0020,16] -- Zero-page IOCB

The Zero-page IOCB is an exact copy (except as noted in the discussions that follow) of the IOCB specified by the 6502 X register upon entry to CIO; CIO copies the outer level IOCB to the Zero-page IOCB, performs the indicated function, moves the (possibly altered) Zero-page IOCB back to the outer level IOCB, and then returns to the caller.

Although both the outer level IOCB and the Zero-page IOCB are defined to be 16 bytes in size, only the first 12 bytes are moved by CIO.

G14   ICHIDZ [0020,1] -- Handler index number

See Section 5. Set to $FF on CLOSE.

G15   ICDNOZ [0021,1] -- Device drive number

See Section 5.

G16   ICCOMZ [0022,1] -- Command byte

See Section 5.

G17   ICSTAZ [0023,1] -- Status byte

See Section 5.

G18   ICBALZ,ICBALH [0024,2] -- Buffer address

See Section 5. This pointer variable is modified by CIO in the course of processing some commands; however, the original value is restored before returning to the caller.

G19   ICPTLZ,ICPTHZ

See Section 5. Set to point to CIO's "IOCB not OPEN" routine on CLOSE.

G20   ICBLLZ,ICBLHZ [0028,2] -- Buffer length / byte count

See Section 5. This double-byte variable, which starts out representing the buffer length, is modified by CIO in the course

of processing some commands; then, before returning to the
caller, the transaction byte count is stored therein.

G21   ICAX1Z,ICAX2Z [002A,2] -- Auxiliary information

See Section 5.

G22   ICSPRZ (ICIDNO,CIOCHR) [002C,4] -- CIO working variables

ICSPRZ and ICSPRZ+1 are used by CIO in obtaining the appropriate
Handler entry point from the handler's vector table (see Section 9).

ICSPRZ+2 is also labeled ICIDNO and retains the value of the 6502 X
register from CIO entry. The X register is loaded from ICIDNO as CIO
returns to the caller.

ICSPRZ+3 is also labeled CIOCHR and retains the value of the 6502 A
register from CIO entry, except for data reading type commands, in
which case the most recent data byte read is stored in CIOCHR. The
6502 A register is loaded from CIOCHR as CIO returns to the caller.

Internal Working Variables

G23   ICCOMT* [0017,1] -- Command table index

ICCOMT is used as an index to CIO's internal command table, which maps
command byte values to Handler entry offsets (see Section 9 for more
information). ICCOMT contains the value from ICCOMZ except when ICCOMZ
is greater than $0E, in which case ICCOMT is set to $0E.

G24   ICIDNO* [002E,1] -- CIO call X register save/restore

See G22.

G25   CIOCHR* [002F,1] -- CIO call A register save/restore

See G22.

H. SERIAL I/O ROUTINE (SIO)

See Section 9 for discussions relating to SIO.

User Call Parameters

SIO call parameters are passed primarily through a Device Control
Block; although an additional "noisy bus" option exists that is
selectable through a separate variable.

Device Control Block

H1   DCB [0300,12] -- Device Control Block

DCB is the name applied collectively to the 12 bytes at locations
0300-030B. These bytes provide the parameter passing mechanism for SIO
and are described individually below.

H2   DDEVIC [0300,1] -- Device bus ID

See Section 9.

H3   DUNIT [0301,1] -- Device unit number

See Section 9.

H4   DCOMND [0302,1] -- Device command

See Section 9.

H5   DSTATS [0303,1] -- Device status

See Section 9.

H6   DBUFLO,DBUFHI [0304,2] -- Handler buffer address

See Section 9.

H7   DTIMLO [0306,1] -- Device timeout

See Section 9.

H8   DBYTLO,DBYTHI [0308,2] -- Buffer length / byte count

See Section 9.

H9   DAUX1,DAUX2 [030A,2] -- Auxiliary information

See Section 9.

Bus Sound Control

H10   SOUNDR [0041,1] -- Quiet/noisy I/O flag

SOUNDR is a flag used to indicate to SIO whether noise is to be
generated on the television audio circuit when Serial I/O bus
activity is in progress. SOUNDR equal to zero indicates that
sound is to be inhibited, and nonzero indicates that sound is to
be enabled. SIO sets SOUNDR to 3 at power-up and system reset.

Serial Bus Control

Retry Logic

SIO will attempt one complete command retry if the first attempt
is not error free, where a complete command try consists of up to
14 attempts to send (and acknowledge) a command frame, followed
by a single attempt to receive COMPLETE and possibly a data
frame.

H11   CRETRY* [0036,1] -- Command frame retry counter

CRETRY controls the inner loop of the retry logic, that associated
with sending and receiving an acknowledgement of the command frame.
CRETRY is set to 13 by SIO at the beginning of every command
initiation, thus allowing for an initial attempt and up to 13
additional retries.

H12   DRETRY* [0037,1] -- Device retry counter

DRETRY controls the outer loop of the retry logic, that
associated with initiating a command retry after a failure
subsequent to the command frame acknowledgement. DRETRY is set to
1 by SIO at entry, thus allowing for an initial attempt and
1 additional retry.

Checksum

The Serial I/O bus protocol specifies that all command and data
frames must contain a checksum validation byte; this byte is the
arithmetic sum (with end-around carry) of all of the other bytes
in the frame.

H13   CHKSUM* [0031,1] -- Checksum value

CHKSUM contains the frame checksum as computed by SIO for all frame
transfers.

H14   CHKSNT* [003B,1] -- Checksum sent flag

CHKSNT indicates to the serial bus transmit interrupt service
routine whether the frame checksum byte has been sent yet. CHKSNT
equal to zero indicates that the checksum byte has not yet been
sent; after the checksum is sent, CHKSNT is then set nonzero.

H15   NOCKSM* [003C,1] -- No checksum follows data flag

NOCKSM is a flag used to communicate between the SIO top level
code and the Serial bus receive interrupt service routine that
the next input will not be followed by a checksum byte. A value
of zero specifies that a checksum byte will follow, nonzero specifies
that a checksum byte will not follow.

Data Buffering

General Buffer Control

H16   BUFRLO* [0032,1] and BUFRHI* [0033,1] -- Next byte address

BUFRLO and BUFRHI comprise a pointer to the next buffer location
to be read from or written to. For a data frame transfer, the
pointer is initially set to the value contained in the SIO call
parameters DBUFLO and DBUFHI, and is then incremented by the
interrupt service routines as a part of normal bus data transfer.
For a command frame transfer, the pointer is set to point to the
SIO-maintained command frame output buffer.

H17 BFENLO* [0034,1] and BFENHI* [0035,1] -- Buffer end address

BFENLO/BFENHI form a pointer to the byte following the last frame
data byte (not including the checksum) to be sent or received.

BFENLO/BFENHI is the arithmetic sum of BUFRLO/BUFRHI plus the
frame size plus -1.

Command Frame Output Buffer

See Section 9 for the command frame format and description.

H18   CDEVIC* [023A,1] -- Command frame device ID

CDEVIC is set to the value obtained by adding SIO call parameter
DDEVIC to DUNIT and subtracting 1.

H19   CCOMND* [023B,1] -- Command frame command.

CCOMND is set to the value obtained from SIO call parameter
DCOMND.

H20   CAUX1* [023C,1] and CAUX2* [023D,1] -- Auxiliary information

CAUX1 and CAUX2 are set to the values obtained from SIO call
parameters DAUX1 and DAUX2, respectively.

Receive/Transmit Data Buffering

H21   BUFRFL* [0038,1] -- Buffer full flag

BUFRFL is a flag used by the serial bus receive interrupt service
routine to indicate when the main portion of a bus frame has been
received -- all but the checksum byte. BUFRFL equal to zero
indicates that the main portion has not been completely received,
a nonzero value indicates that the main portion has been
received.

H22   RECVDN* [0039,1] -- Receive frame done flag

RECVDN is a flag used by SIO to communicate between the Serial
bus receive interrupt service routine and the main SIO code. The
flag is initially set to zero by SIO, and later set nonzero by
the interrupt service routine after the last byte of a bus frame
has been received.

H23   TEMP* [023E,1] -- SIO 1-byte I/O data

TEMP is used to receive 1-byte responses from serial bus
controllers, such as ACK, NAK, COMPLETE or ERROR.

H24 XMTDON* [003A,1] -- Transmit frame done flag

XMTDON is a flag used by SIO to communicate between the Serial
bus transmit interrupt service routine and the main SIO code. The
flag is initially set to zero by SIO, and later set nonzero by
the interrupt service routine after the last byte of a bus frame
has been transmitted.

SIO Timeout

SIO uses System Timer 1 to provide the timeout capability for
various operations initiated internally. See Section 6 for a
discussion of the capabilities of the System Timers. TIMFLG is
the flag used to communicate between SIO and the timer initiated
code pointed to by CDTMA1.

H25 TIMFLG* [0317,1] -- SIO operation timeout flag

TIMFLG is used to indicate a timeout situation for a bus
operation . The flag is initially set to 1, and if it attains a
value of zero (after the timeout period) before the current
operation is complete, the operation will be aborted. See also
D2.

H26 CDTMV1* [0218,2] -- System Timer 1 value

This 2-byte count takes on various values depending upon the
operation being timed. See also P4.

H27 CDTMA1* [0226,2] -- System Timer 1 address

This vector always points to the JTIMER routine, whose only
function is to set TIMFLG to zero. This vector is initialized by
SIO before every use, so that System Timer 1 can be used by any
process that does not use SIO within a timing function. See also
P5.

Internal Working Variables

H28 STACKP* [0318,1] -- Stack pointer save/restore

STACKP contains the value of the 6502 SP register at entry to
SIO; this is retained to facilitate a direct error exit from an
SIO subroutine.

H29 TSTAT* [0319,1] -- Temporary status

TSTAT is used to return the operation status from the WAIT
routine and will contain one of the SIO status byte values as
shown in Appendix B.

H30 ERRFLG* [023F,1] -- I/O error flag

ERRFLG is used for communication between the WAIT routine and the
outer level SIO code. ERRFLG is normally zero, but is set to $FF
when a device responds with an invalid response byte.

H31 STATUS* [0030,1] -- SIO operation status

STATUS is a zero-page variable that is used within SIO to contain
the operation status that will be stored to the calling sequence
parameter variable DSTATS when SIO returns to the caller.

H32 SSKCTL* [0232,1] -- SKCTL copy

SSKCTL is utilized by SIO to keep track of the content of the
SKCTL [D20F] register, which is a write-only register.

J. ATARI CONTROLLERS

The ATARI controllers are read as part of the Stage 2 VBLANK
process. The encoded data is partially decoded and processed as
shown in the subsections that follow.

Joysticks

Up to four joystick controllers can be attached to the computer
console, each with a 9-position joystick plus a trigger button.

J1 STICK0 - STICK3 [0278,4] -- Joystick position sense

The 4 joystick position sense variables contain a bit-encoded
position sense as shown below:


```
        7 6 5 4 3 2 1 0
       +-+-+-+-+-+-+-+-+
       !0 0 0 0!R!L!D!U!
       +-+-+-+-+-+-+-+-+
```

where: R = 0 indicates joystick RIGHT sensor true.
        L = 0 indicates joystick LEFT sensor true.
        D = 0 indicates joystick DOWN sensor true.
        U = 0 indicates joystick UP sensor true.

Nine unique combinations are possible, indicating the possible
joystick positions shown below:

```
    CENTER      $0F
    UP          $0E
    UP/RIGHT     $06
    RIGHT       $07
    DOWN/RIGHT  $05
    DOWN        $0D
    DOWN/LEFT   $09
    LEFT        $0B
    UP/LEFT     $0A
```


J2 STRIG0 - STRIG3 [0284,4] -- Joystick trigger sense

The four joystick trigger sense variables each contain a single bit
indicating the position of the joystick trigger as shown below:


```
        7 6 5 4 3 2 1 0
       +-+-+-+-+-+-+-+-+
       !0 0 0 0 0 0 0!T!
       +-+-+-+-+-+-+-+-+
```

where: T = 0 indicates trigger pressed.

Paddles

Up to eight paddle controllers can be connected to the computer,
each with a potentiometer and a trigger sense.

J3 PADDL0 - PADDL7 [0270,8] -- Paddle position sense

There is a single-byte variable associated with each paddle
position sense; the values range from 228 for full

counterclockwise rotation to 1 for full clockwise rotation.

The paddle values are often converted by the user, as shown below, to give a result of 0 for full counterclockwise rotation and 227 for full clockwise rotation:

     VALUE := 228 - PADDLX;

J4 PTRIG0 - PTRIG7 [027C,8] -- Paddle trigger sense

The 8-paddle trigger sense variables each contain a single bit indicating the position of the paddle trigger as shown below:

         7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+
        !0 0 0 0 0 0 0!T!
        +-+-+-+-+-+-+-+-+

where: T = 0 indicates trigger pressed.

Light Pen

The OS reads the position of a single light pen and stores the horizontal and vertical position codes in two variables; these codes are not the same as the actual screen coordinates. The pen position codes for different portions of the screen are shown below:

  Left edge -- 67.
  Codes increase in increments of one to a value of 227, then go to 0 and continue to increase monotonically (one count per color clock).
  Right edge -- 7.

  Upper edge -- 16.
  Codes increase in increments of one (one count per two raster lines). Lower edge -- 111.

The light pen hardware will read and latch the pen position 60 times per second, independent of the pen button position, which is separately sensed.

In order for the light pen to operate it must be positioned over a portion of the screen which has sufficient luminance to activate the photosensor in the pen; a blank (dark) screen will generally not provide enough luminance to utilize the light pen.

J5 LPENH [0234,1] -- Light pen horizontal position code

LPENH contains the horizontal position code for the light pen; the algorithm below (written in Pascal) shows the conversion from position code to screen coordinate (screen mode 7):

  IF LPENH < 33   { check for rollover point }
     THEN   { adjust values to right of rollover }

```
          XPOS := LPENH + 227
     ELSE                      { no adjustment to left of rollover point }
          XPOS := LPENH;
  XPOS := XPOS - 67;    { adjust for left edge offset }
  IF XPOS < 0 THEN XPOS := 0;
  IF XPOS > 159 THEN XPOS := 159;
```

J6 LPENV [0235, 1] -- Light pen vertical position code

LPENV contains the vertical position code for the light pen; the
algorithm below (written in Pascal) shows the conversion from position
code to screen coordinate (screen mode 7):

```
  YPOS := LPENV - 16;   { adjust for upper edge offset }
  IF YPOS < 0 THEN YPOS := 0;
  IF YPOS > 95 THEN YPOS := 95;
```

J7 STICK0 - STICK3 [0278, 4] -- Light pen button sense

The light pen button sense is encoded in one of STICK0 - STICK3
(depending upon the actual controller port used) as shown
below:

```
      7                 0
     +-+-+-+-+-+-+-+-+
     |         |0|0|0|T|
     +-+-+-+-+-+-+-+-+
```

where: T = 0 indicates the light pen button is pressed.

Driving Controllers

The driving controller has no position stops and thus allows unlimited
rotation in either direction; the output of the controller is a 2-bit
Gray code which can be used to determine the direction of rotation.
The controller is sensed using the same internal hardware as the
joystick, thus the same data base variables are used for both.

J8   STICK0 - STICK3 [0278,4] -- Driving controller sense

The 4 driving controller sense variables contain an encoded
rotation (position) sense value, as shown below:

```
    7 6 5 4 3 2 1 0
   +-+-+-+-+-+-+-+-+
   !0 0 0 0 1 1!val!
   +-+-+-+-+-+-+-+-+
```

where a clockwise rotation of the controller produces the following
continuous sequence of four values (shown in hexadecimal):

    0F, 0D, 0C, 0E, 0F, 0D, ...

and a counterclockwise rotation of the controller produces the
following continuous sequence of four values:

    0F, 0E, 0C, 0D, 0F, 0E, ...

J9   STRIG0 - STRIG3 [0284,4] -- Driving trigger sense

The four driving trigger sense variables each contain a single bit
indicating the position of the driving trigger as shown below:

```
    7 6 5 4 3 2 1 0
   +-+-+-+-+-+-+-+-+
   !0 0 0 0 0 0 0!T!
   +-+-+-+-+-+-+-+-+
```

where: T = 0 indicates trigger pressed.

K. DISK FILE MANAGER

See Section 5 for information relating to the Disk File Manager.

K1   FMSZPG* [0043,7] -- FMS reserved space

FMSZPG is the reserved space in the database for the variables shown
below; the names associated with K2 through K5 are not in the system
equate file.

K2   ZBUFP* [0043,2] -- Buffer pointer

K3   ZDRVA* [0045,2] -- Drive pointer

K4   ZSBA* [0047,2] -- Sector buffer pointer

K5   ERRNO* [0049,1] -- Error number

L.   DISK UTILITY POINTER

L1   DSKUTL* [001A,2] -- Page-zero pointer variable

M.   FLOATING POINT PACKAGE

See Section 8 for a description of the Floating Point Package.

M1   FR0 [00D4,6] -- FP register 0

M2   FRE* [00DA,6] -- FP register (internal)

M3   FR1 [00E0,6] -- FP register 1

M4   FR2* [00E6,6] -- FP register 2 (internal)

M5   FRX* [00EC,1] -- Spare (unused)

M6   EEXP* [00ED,1] -- Exponent value (internal)

M7   NSIGN* [00EE,1] -- Sign of mantissa (internal)

M8   ESIGN* [00EF,1] -- Sign of exponent (internal)

M9   FCHRFLG* [00F0,1] -- First character flag (internal)

M10   DIGRT* [00F1,1] -- Digits to right of decimal point

M11   CIX [00F2,1] -- Character index

M12   INBUFF [00F3,2] -- Input text buffer pointer

M13 ZTEMP1* [00F5,2] -- Temporary storage

M14 ZTEMP4* [00F7,2] -- Temporary storage

M15 ZTEMP3* [00F9,2] -- Temporary storage

M16 FLPTR [00FC,2] -- Pointer to FP number

M17 FPTR2* [00FE,2] -- FP package use

M18 LBPR1* [057E,1] -- LBUFF preamble

M19 LBPR2* [057F,1] -- LBUFF preamble

M20 LBUFF [0580,96] -- Text buffer

M21 PLYARG* [05E0,6] -- FP register (internal)

M22 FPSCR/FSCR* [05E6,6] -- FP register (internal)

M23 FPSCR1/SCR1* [05EC,6] -- FP register (internal)

M24 DEGFLG/RADFLG [00FB,1] -- Degrees/radians flag

DEGFLG = 0 indicates radians, 6 indicates degrees.

N. Power-Up and SYSTEM RESET

See Section 7 for details of the power-up and system reset
operations.

RAM Sizing

During power-up and system reset the first non-RAM address above 1000
hex is located and its address retained using a nondestructive
test. The first byte of every 4K memory "block" is tested to see if
it is alterable; if so, the original value is restored and the next
block is tested, and if not, that address is considered to be the
end of RAM.

N1 RAMLO*/TRAMSZ* [0004,3] -- RAM data/test pointer (temporary)

RAMLO+1 contains the LSB of the address to be tested (always = 0)
and TRAMSZ (same as RAMLO+2) contains the MSB of the address to be
tested. RAMLO+0 contains the complemented value of the data
originally contained in the memory location being tested.

Later in the initialization process these variables are used for
totally unrelated functions; but first the value in TRAMSZ is moved
to the variables RAMSIZ and MEMTOP+1.

N2 TSTDAT* [0007,1] -- Test data byte save

TSTDAT contains the original value of the memory location being
tested.

Diskette/Cassette-Boot

As a part of the Power-up sequence, software can be booted from an
attached disk drive or cassette player as explained in Section 10.

N3 DOSINI [000C,2] -- Diskette-boot initialization vector.

DOSINI contains the disk booted software initialization address
from the beginning of the boot file (see Section 10) whenever a
diskette-boot is successfully completed.

N4 CKEY* [004A,1] -- Cassette-boot request flag

CKEY is an internal flag used to indicate that the console [START]
key was pressed during Power-up, thus indicating that a
cassette-boot is desired. CKEY equals zero when no cassette-boot is
requested, and is nonzero when a cassette-boot is requested. The
flag is cleared to zero after a cassette-boot.

N5 CASSBT* [004B,1] -- Cassette-booting flag

CASSBT is used during the cassette-boot process to indicate to
shared code that the cassette is being booted and not the diskette.
CASSBT equal to zero indicates a diskette-boot, and nonzero indicates
a cassette-boot.

N6 CASINI [0002,2] -- Cassette-boot initialization vector

CASINI contains the cassette-booted software initialization address
from the beginning of the boot file (see Section 10) whenever a

cassette-boot is successfully completed.

N7 BOOT?* [0009,1] -- Successful diskette/cassette-boot flag.

BOOT? indicates to the initialization processor which, if any, of
the boot operations went to successful completion. The flag values
are set by the OS and the format for the variable is shown below:

```
    7 6 5 4 3 2 1 0
   +-+-+-+-+-+-+-+-+
   :         :C:D:
   +-+-+-+-+-+-+-+-+
```

where: C = 1 indicates that the cassette-boot was completed.
       D = 1 indicates that the diskette-boot was completed.

N8 DFLAGS* [0240,1] -- Diskette flags

DFLAGS contains the value of the first byte of the boot file, after a
diskette-boot. See Section 10.

N9 DBSECT* [0241,1] -- Diskette-boot sector count

DBSECT is initially set to the value of the second byte of the boot
file, during a diskette-boot, and is then used to control the number
of additional diskette sectors read, if any.

N10 BOOTAD* [0242,2] -- Diskette-boot memory address

BOOTAD is initially set to the value of the third and fourth
bytes of the boot file, during a diskette-boot, and is not
modified thereafter.

Environment Control

If, at the end of a power-up or system reset, control is not
given to one of the cartridges (as explained in Sections 7 and 10),
then program control passes to the address contained in the data
base variable DOSVEC.

N11 COLDST* [0244,1] -- Coldstart complete flag

COLDST is used by the initialization routine to detect the case of
a system reset occurring before the completion of the power-up
process. COLDST is set to $FF at the beginning of the power-up

sequence and is set to 0 at the completion; if a system reset
occurs while the value is nonzero, the power-up sequence will be
reinitiated (rather than initiating a system reset sequence).

N12 DOSVEC [000A,2] -- Noncartridge control vector

At the beginning of power-up the OS sets DOSVEC to point to the
"blackboard" routine; DOSVEC can then be altered as a consequence
of a diskette-boot or cassette-boot (as explained in Section 10) to
establish a new control program. Control will be passed through
DOSVEC on all power-up and system reset conditions in which a
cartridge does not take control first.

System Reset

N13 WARMST [0008,1] -- Warmstart flag

WARMST equals $FF during a system reset (warmstart)
initialization and equals 0 during a power-up initialization
(coldstart).

P. INTERRUPTS

See Section 6 for a discussion of interrupt processing.

P1 CRITIC [0042,1] -- Critical code section flag

CRITIC is used to signal to the VBLANK interrupt processor that a
critical code section is executing without IRQ interrupts being
inhibited; the VBLANK interrupt processor will stop interrupt
processing after stage 1 and before stage 2, just as if the 6502
processor I bit were set, when CRITIC is set.

CRITIC equal to zero indicates that the currently executing code
section is noncritical, while any nonzero value indicates that the
currently executing code section is critical.

P2 POKMSK [0010,1] -- POKEY interrupt mask

POKMSK is a software maintained interrupt mask that is used in
conjunction with the enabling and disabling of the various POKEY
interrupts. This mask is required because the POKEY interrupt
enable register IRQEN [D20E] is a write-only register, and at any
point in time the system can have several users independently
enabling and disabling POKEY interrupts. POKMSK is updated by the

users to always contain the current content of IRQEN.

System Timers

The System Timers are discussed in detail in Section 6.

Realtime Clock

The realtime clock (or frame counter, as it is sometimes called) is incremented as part of the stage 1 VBLANK process as explained in Section 6.

P3   RTCLOK [0012,3] -- Realtime frame counter

RTCLOK+0 is the most significant byte, RTCLOK+1 the next most significant byte, and RTCLOK+2 the least significant byte. See the discussions at D3 and preceding B10 for OS use of RTCLOK.

System Timer 1

System Timer 1 is maintained as part of the stage 1 VBLANK process, and thus has the highest priority of any of the user timers.

P4   CDTMV1 [0218,2] -- System Timer 1 value

CDTMV1 contains zero when the timer is inactive, otherwise it contains the number of VBLANKs remaining until timeout. Also see H26.

P5   CDTMA1 [0226,2] -- System Timer 1 jump address

CDTMA1 contains the address to which to JSR should the timer timeout. See also H27 and Section 6.

System Timer 2

System Timer 2 is maintained as part of the stage 2 VBLANK process,
and has the second highest priority of the user timers. The OS does
not have any direct use for System Timer 2.

P6   CDTMV2 [021A,2] -- System Timer 2 value

CDTMV2 contains zero when the timer is inactive, otherwise it
contains the number of VBLANKs remaining until timeout.

P7   CDTMA2 [0228,2] -- System Timer 2 jump address

CDTMA2 contains the address to which to JSR should the timer
timeout. See Section 6.

System Timers 3, 4 and 5

System Timers 3, 4 and 5 are maintained as part of the stage 2
VBLANK process, and have the lowest priority of the user timers.
The OS does not have any direct use for these timers.

P8   CDTMV3 [021C,2], CDTMV4 [021E,2] and CDTMV5 [0220,2]

These variables contain zero when the corresponding timers are
inactive, otherwise they contain the number of VBLANKs remaining
until timeout.

P9   CDTMF3 [022A,1], CDTMF4 [022C,1] and CDTMF5 [022E,2]

Each of these 1-byte variables will be set to zero should its
corresponding timer timeout. The OS never modifies these bytes
except to set them to zero upon timeout (and initialization).

RAM Interrupt Vectors

There are RAM vectors for many of the interrupt conditions within
the system. See Section 6 for a discussion of the placing of values
to these vectors.

NMI Interrupt Vectors

P10   VDSLST [0200,2] -- Display-list interrupt vector

This vector is not used by the OS. See Section 6.

P11   VVBLKI [0222,2] -- Immediate VBLANK vector

This vector is initialized to point to the OS stage 1 VBLANK

P12   VVBLKD [0224,2] -- Deferred VBLANK vector

This vector is initialized to point to the OS VBLANK exit routine.
See Section 6.

IRQ Interrupt Vectors

P13   VIMIRQ [0216,2] -- General IRQ vector

This vector is initialized to point to the OS IRQ interrupt
processor. See Section 6.

P14   VPRCED [0202,2] -- Serial I/O bus proceed signal

The serial bus line that produces this interrupt is not used in the
current system. See Section 6.

P15   VINTER [0204,2] -- Serial I/O bus interrupt signal

The serial bus line that produces this interrupt is not used in the
current system. See Section 6.

P16   V[BREAK] [0206,2] -- BRK instruction vector

This vector is initialized to point to a PLA, RTI sequence as the
OS proper does not utilize the BRK instruction. See Section 6.

P17   VKEYBD [0208,2] -- Keyboard interrupt vector

This vector is initialized to point to the Keyboard Handler's
interrupt service routine. See Section 6 and the discussion
preceding E1.

P18 VSERIN [020A,2] -- Serial I/O bus receive data ready

This vector is initialized to point to the SIO utility's interrupt
service routine. See Section 6.

P19 VSEROR [020C,2] -- Serial I/O bus transmit ready

This vector is initialized to point to the SIO utility's interrupt
service routine. See Section 6.

P20 VSEROC [020E,2] -- Serial I/O bus transmit complete

This vector is initialized to point to the SIO utility's interrupt
service routine. See Section 6.

P21 VTIMR1 [0210,2], VTIMR2 [0212,2] and VTIMR4 [0214,2] -- POKEY
timer vectors

The POKEY timer interrupts are not used by the OS See Section 6.

Hardware Register Updates

As part of the stage 2 VBLANK process, certain hardware registers
are updated from OS data base variables as explained in Section 6.

P22 SDMCTL* [022F,1] -- DMA control

SDMCTL is set to a value of $02 at the beginning of a Display
Handler OPEN command, and then later set to a value of $22. The
value of SDMCTL is stored to DMACTL [D400] as part of the stage 2
VBLANK process.

P23 SDLSTL* [0230,1] and SDLSTH* [0231,1] -- Display list address

The Display Handler formats a new display list with every OPEN
command and puts the display list address in SDLSTL and SDLSTH. The
value of these bytes are stored to DLISTL [D402] and DLISTH [D403]
as part of the stage 2 VBLANK process.

```
         0360-036F        IOCB #2
         0370-037F        IOCB #3
         0380-038F        IOCB #4
         0390-039F        IOCB #5
         03A0-03AF        IOCB #6
         03B0-03BF        IOCB #7
```

NOTE: There is a potential timing problem associated with the
updating of the hardware registers from the data base variables.
Since the stage 2 VBLANK process is performed with interrupts
enabled, it is possible for an IRQ interrupt to occur before the
updating of DLISTH and DLISTL. If the processing of that
interrupt (plus other nested interrupts) exceeds the
vertical-blank delay (1 msec), then the display list pointer
register will not have been updated when display list processing
commences for the new frame, and a screen glitch will result.

P24   GPRIOR* [026F,1] -- Priority control

The Display Handler alters bits 6 and 7 of GPRIOR as part of
establishing the GTIA mode. The value of GPRIOR is stored to
PRIOR [D01B] as part of the stage 2 VBLANK process.

P25   CHACT* [02F3,1] -- Character control

The Display Handler sets CHACT to $02 on every OPEN command. The
value of CHACT is stored to CHACTL [D401] as part of the stage 2
VBLANK process.

P26   CHBAS [02F4,1] -- Character address base

The Display Handler sets CHBAS to $E0 on every OPEN command. The
value of CHBAS is stored to CHBASE [D409] as part of the stage 2
VBLANK process. This variable controls the character subset for
screen modes 1 and 2; a value of $E0 provides the capital letters
and number set whereas a value of $E2 provides the lowercase
letters and special graphics set. See B55 for more information.

P27   PCOLRx [02C0,4] and COLORx [02C4,5] -- Color registers

See B7 and B8.

Internal Working Variables

P28   INTEMP* [022D,1] -- Temporary storage

INTEMP is used by the SETVBL (SETVBV) routine.

R. USER AREAS

The areas shown below are available to the user in a non-nested
environment. See Section 4 for further information.

R1   [0080,128]

R2   [0480,640]

ALPHABETICAL LIST OF DATA BASE VARIABLES

NAME	VID	ADDRESS SIZE
ADDCOR	D4	030E, 1
ADRESS	B39	0064, 2
APPMHI	A3	000E, 2
ATACHR	B54	02FB, 1
ATRACT	B10	004D, 1
BFENHI	H17	0035, 1
BFENLO	H17	0034, 1
BITMSK	B50	006E, 1
BLIM	D10	028A, 1
BOOT?	N7	0009, 1
BOOTAD	N10	0242, 2
BOTSCR	B16	02BF, 1
BPTR	D11	003D, 1
BRKKEY	E5	0011, 1
BUFADR	C1	0015, 2
BUFCNT	B42	006B, 1
BUFRFL	H21	0038, 1
BUFRHI	H16	0033, 1
BUFRLO	H16	0032, 1
BUFSTR	B43	006C, 2
CASBUF	D9	03FD, 131
CASFLG	D8	030F, 1
CASINI	N6	0002, 2
CASSBT	N5	004B, 1
CAUX1	H20	023C, 1
CAUX2	H20	023D, 1
CBAUDH	D1	02EF, 1
CBAUDL	D1	02EE, 1
CCOMND	H19	023B, 1
CDEVIC	H18	023A, 1
CDTMA1	P5, H27	0226, 2
CDTMA2	P7	0228, 2
CDTMF3	P9	022A, 1
CDTMF4	P9	022C, 1
CDTMF5	P9	022E, 1
CDTMV1	P4, H26	0226, 2
CDTMV2	P6	021A, 2
CDTMV3	P8	021C, 2

CDTMV4	P8	021E, 2
CDTMV5	P8	0220, 2
CH	E3	02FC, 1
CHKSNT	H14	003B, 1
CH1	E1	02F2, 1
CHACT	P25	02F3, 1
CHAR	B55	02FA, 1
CHBAS	P26	02F4, 1
CHKSNT	H14	003B, 1
CHKSUM	H13	0031, 1
CIOCHR	G25	002F, 1
CIX	M11	00F2, 1
CKEY	N4	004A, 1
COLAC	B24	0072, 2
COLCRS	B2	0055, 2
COLDST	N11	0244, 1
COLINC	B21	007A, 1
COLOR0	B8, P27	02C4, 1
COLOR1	B8, P27	02C5, 1
COLOR2	B8, P27	02C6, 1
COLOR3	B8, P27	02C7, 1
COLOR4	B8, P27	02C8, 1
COLRSH	B11	004F, 1
COUNTR	B23	007E, 2
CRETRY	H11	0036, 1
CRITIC	P1	0042, 1
CRSINH	B1	02F0, 1
CSTAT	S2	0288, 1
DAUX1	H9	030A, 1
DAUX2	H9	030B, 2
DBSECT	N9	0241, 1
DBUFHI	H6	0304, 1
DBUFLO	H6	0305, 1
DBYTHI	H8	0308, 1
DBYTLO	H8	0309, 1
DCB	H1	0300, 12
DCOMND	H4	0302, 1
DDEVIC	H2	0300, 1
DEGFLG	M24	00FB, 1
DELTAC	B22	0077, 2
DELTAR	B22	0076, 1
DFLAGS	N8	0240, 1
DIGRT	M10	00F1, 1
DINDEX	B35	0057, 1
DMASK	B28	02A0, 1
DOSINI	N3	000C, 2
DOSVEC	N12	000A, 2
DRETRY	H12	0037, 1
DRKMSK	B12	004E, 1
DSKTIM	C2	0246, 1
DSKUTL	L1	001A, 2
DSPFLG	B27	02FE, 1
DSTAT	B34	004C, 1

DSTATS	H5	0303, 1
DTIMLO	H7	0306, 1
DUNIT	H3	0301, 1
DUNUSE	S3	0307, 1
DVSTAT	G11	02EA, 4
EEXP	M6	00ED, 1
ENDPT	B25	0074, 2
ERRFLG	H30	023F, 1
(ERRNO	K5)	0049, 1
ESCFLG	B26	02A2, 1
ESIGN	M8	00EF, 1
FCHRFL	M9	00F0, 1
FEOF	D12	003F, 1
FILDAT	B17	02FD, 1
FILFLG	B18	02B7, 1
FLPTR	M16	00FC, 2
FMSZPG	K1	0043, 7
FPSCR	M22	05E6, 6
FPSCR1	M23	05EC, 6
FPTR2	M17	00FE, 2
FR0	M1	00D4, 6
FR1	M3	00E0, 6
FR2	M4	00E6, 6
FRE	M2	00DA, 6
FREQ	D15	0040, 1
FRMADR	B41	0068, 2
FRX	M5	00EC, 1
FSCR	M22	05E6, 6
FSCR1	M23	05EC, 6
FTYPE	D13	003E, 1
GPRIOR	P24	026F, 1
HATABS	G12	031A, 38
HOLD1	B30	0051, 1
HOLD2	B31	029F, 1
HOLD3	B32	029D, 1
HOLD4	B20	02BC, 1
HOLDCH	E7	007C, 1
ICAX1	G9	034A, 1
ICAX1Z	G21	002A, 1
ICAX2	G9	034B, 1
ICAX2Z	G21	002B, 1
ICBAH	G6	0345, 1
ICBAHZ	G18	0025, 1
ICBAL	G6	0344, 1
ICBALZ	G18	0024, 1
ICBLH	G8	0349, 1
ICBLHZ	G20	0029, 1
ICBLL	G8	0348, 1
ICBLLZ	G20	0028, 1

ICCOM	G4	0342, 1
ICCOMT	G23	0017, 1
ICCOMZ	G16	0022, 1
ICDNO	G3	0341, 1
ICDNOZ	G15	0021, 1
ICHID	G2	0340, 1
ICHIDZ	G14	0020, 1
ICIDNO	G24, G2	2002E, 1
ICPTH	G7	0347, 1
ICPTHZ	G19	0027, 1
ICPTL	G7	0346, 1
ICPTLZ	G19	0026, 1
ICSPR	G10	034C, 4
ICSPRZ	G22	002C, 4
ICSTA	G5	0343, 1
ICSTAZ	G17	0023, 1
INBUFF	M12	00F3, 2
INSDAT	B45	007D, 1
INTEMP	P28	022D, 1
INVFLG	E9	02B6, 1
IOCB	G1	0340, 16
IOCBAS	G13	0020, 16
KEYDEL	E2	02F1, 1
LBFEND	M20	0580, 96
LBPR1	M18	057E, 1
LBPR2	M19	057F, 1
LBUFF	M20	0580, 96
LINBUF	B51	0247, 40
LMARGN	B5	0052, 1
LOGCOL	B15	0063, 1
LOGMAP	B14	02B2, 4
MEMLO	A1	02E7, 2
MEMTOP	A2	02E5, 2
MLTTMP	B40	0066, 2
NEWCOL	B19	0061, 2
NEWROW	B19	0060, 1
NOCKSM	H15	003C, 1
NSIGN	M7	00EE, 1
OLDADR	B38	005E, 2
OLDCHR	B37	005D, 1
OLDCOL	B3	005B, 2
OLDROW	B3	005A, 1
OPNTMP	B40	0066, 2
PADDL0	J3	0270, 1
PADDL1	J3	0271, 1
PADDL2	J3	0272, 1
PADDL3	J3	0273, 1
PADDL4	J3	0274, 1

```
PADDL5        J3             0275, 1
PADDL6        J3             0276, 1
PADDL7        J3             0277, 1
PBPNT         F3             001D, 1
PBUFSZ        F2             001E, 1
PCOLR0        B7, P27        02C0, 1
PCOLR1        B7, P27        02C1, 1
PCOLR2        B7, P27        02C2, 1
PCOLR3        B7, P27        02C3, 1
PLYARG        M21            05E0, 6
POKMSK        P2             0010, 1
PRNBUF        F1             03C0, 40
PTEMP         F4             001F, 1
PTIMOT        F5             001C, 1
PTRIG0        J4             027C, 1
PTRIG1        J4             027D, 1
PTRIG2        J4             027E, 1
PTRIG3        J4             027F, 1
PTRIG4        J4             0280, 1
PTRIG5        J4             0281, 1
PTRIG6        J4             0282, 1
PTRIG7        J4             0283, 1

RADFLG        M24            00FB, 1
RAMLO         N1             0004, 3
RAMSIZ        A5             02E4, 1
RAMTOP        A4             006A, 1
RECVDN        H22            0039, 1
RMARGN        B6             0053, 1
ROWAC         B24            0070, 2
ROWCRS        B2             0054, 1
ROWINC        B21            0079, 1
RTCLOK        P3             0012, 3

SAVADR        B41            0068, 2
SAVIO         D7             0316, 1
SAVMSC        B36            0058, 2
SCRFLG        B9             02BB, 1
SDLSTH        P23            0231, 1
SDLSTL        P23            0230, 1
SDMCTL        P22            022F, 1
SHFAMT        B29            006F, 1
SHFLOK        E6             02BE, 1
SOUNDR        H10            0041, 1
SRTIMR        E8             022B, 1
SSFLAG        E4             02FF, 1
SSKCTL        H32            0232, 1
STACKP        H28            0318, 1
STATUS        H31            0030, 1
STICK0        J1, J7, J8     0278, 1
STICK1        J1, J7, J8     0279, 1
STICK2        J1, J7, J8     027A, 1
STICK3        J1, J7, J8     027B, 1
STRIG0        J2, J7, J9     0284, 1
```

STRIG1	J2, J7, J9	0285, 1
STRIG2	J2, J7, J9	0286, 1
STRIG3	J2, J7, J9	0284, 4
SUBTMP	B48	029E, 1
SWPFLG	B44	007B, 1
TABMAP	B13	02A3, 15
TEMP	H23	023E, 1
TEMP1	D5	0312, 2
TEMP3	D6	0315, 1
TIMER1	D3	030C, 2
TIMER2	D3	0310, 2
TIMFLG	D2, H25	0317, 1
TINDEX	B49	0293, 1
TMPCHR	B33	0050, 1
TMPCOL	B46	02B9, 2
TMPLBT	B47	02A1, 1
TMPROW	B46	02B8, 1
TOADR	B40	0066, 2
TRAMSZ	N1	0004, 3
TSTAT	H29	0319, 1
TSTDAT	N2	0007, 1
TXTCOL	B4	0291, 2
TXTMSC	B52	0294, 2
TXTOLD	B53	0296, 6
TXTROW	B4	0290, 1
USAREA	R1	0080, 128
VBREAK	P16	0206, 2
VDSLST	P10	0200, 2
VIMIRQ	P13	0216, 2
VINTER	P15	0204, 2
VKEYBD	P17	0208, 2
VPRCED	P14	0202, 2
VSERIN	P18	020A, 2
VSEROC	P20	020E, 2
VSEROR	P19	020C, 2
VTIMR1	P21	0210, 2
VTIMR2	P21	0212, 2
VTIMR4	P21	0214, 2
VVBLKD	P12	0224, 2
VVBLKI	P11	0222, 2
WARMST	N13	0008, 1
WMODE	D14	0289, 1
XMTDON	H24	003A, 1
(ZBUFF	K2)	0043, 2
(ZDRVA	K3)	0045, 2
ZIOCB	G13	0020, 16
(ZSBA	K4)	0047, 2
ZTEMP1	M13	00F5, 2

```
ZTEMP3          M15          00F9,2
ZTEMP4          M14          00F7,2
```

MEMORY ADDRESS ORDERED LIST OF DATABASE VARIABLES

ADDRESS	VID	NAME
0000-0001	S7	LNZBS
0002-0003	N6	CASINI
0004-0006	N1	RAMLO, TRAMSZ
0007	N2	TSTDAT
0008	N13	WARMST
0009	N7	BOOT?
000A-000B	N12	DOSVEC
000C-000D	N3	DOSINI
000E-000F	A3	APPMHI
0010	P2	POKMSK
0011	E5	BRKKEY
0012-0014	P3	RTCLOK
0015-0016	C1	BUFADR
0017	G23	ICCOMT
001A-001B	L1	DSKUTL
001C	F5	PTIMOT
001D	F3	PBPNT
001E	F2	PBUFSZ
001F	F4	PTEMP
0020	G13, G14	ICHIDZ
0021	G15	ICDNOZ
0022	G16	ICCOMZ
0023	G17	ICOBAS
0024-0025	G18	ICBALZ, ICBAHZ
0026-0027	G19	ICPTLZ, ICPTHZ
0028-0029	G20	ICBLLZ, ICBLHZ
002A-002B	G21	ICAX1Z, ICAX2Z
002C-002F	G22, G24, G25	ICSPRZ
0030	H31	STATUS
0031	H13	CHKSUM
0032-0033	H16	BUFRLO, BUFFRHI
0034-0035	H17	BFENLO, BFENHI
0036	H11	CRETRY
0037	H12	DRETRY
0038	H21	BUFRFL
0039	H22	RECVDN
003A	H24	XMTDON
003B	H14	CHKSNT
003C	H15	NOCKSM
003D	D11	BPTR
003E	D13	FTYPE
003F	D12	FEOF
0040	D15	FREQ
0041	H10	SOUNDR
0042	P1	CRITIC
0043-0049	K1, K2, K3, K4, K5	ZBUFF, ZBUFP, ZDRVA, ZSBA
004A	N4	CKEY
004B	N5	CASSBT
004C	B34	DSTAT

```
004D              B10              ATRACT
004E              B12              DRKMSK
004F              B11              COLRSH
0050              B33              TMPCHR
0051              B30              HOLD1
0052              B5               LMARGN
0053              B6               RMARGN
0054-0056         B2               ROWCRS, COLCRS
0057              B35              DINDEX
0058-0059         B36              SAVMSC
005A-005C         B3               OLDROW, OLDCOL
005D              B37              OLDCHR
005E-005F         B38              OLDADR
0060-0062         B19              NEWROW, NEWCOL
0063              B15              LOGCOL
0064-0065         B39              ADRESS
0066-0067         B40              MLTTMP, OPNTMP, TOADR
0068-0069         B41              SAVADR/FRMADR
006A              A4               RAMTOP
006B              B42              BUFCNT
006C-006D         B43                BUFSTR
006E              B50              BITMSK
006F              B29              SHFAMT
0070-0073         B24              ROWAC, COLAC
0074-0075         B25              ENDPT
0076-0078         B22              DELTAR, DELTAC
0079-007A         B21              ROWINC, COLINC
007B              B44              SWPFLG
007C              E7               HOLDCH
007D              B45              INSDAT
007E-007F         B23              COUNTR

0080-00FF         SEE FLOATING POINT VARIABLE LIST AT END.

0100-01FF         6502 STACK

0200-0201         P10              VDSLST
0202-0203         P14              VPRCED
0204-0205         P15              VINTER
0206-0207         P16              VBREAK
0208-0209         P17              VKEYBD
020A-020B         P18              VSERIN
020C-020D         P19              VSEROR
020E-020F         P20              VSEROC
0210-0215         P21              VITMR1, VITMR2, VITMR4
0216-0217         P13              VIMIRQ
0218-0219         P4, H26          CDTMV1
021A-021B         P6               CDTMV2
021C-0221         P8               CDTMV3, CDTMV4, CDTMV5
0222-0223         P11              VVBLKI
0224-0225         P12              VVBLKD
0226-0227         P5, H27          CDTMA1
0228-0229         P7               CDTMA2
022A              P9               CDTMF3
```

022B	E8	SRTIMR
022C	P9	CDTMF4
022D	P28	INTEMP
022E	P9	CDTMF5
022F	P22	SDMCTL
0230-0231	P23	SDLSTL, SDLSTH
0232	H32	SSKCTL
023A	H18	CDEVIC
023B	H19	CCOMND
023C-023D	H20	CAUX1, CAUX2
023E	H23	TEMP
023F	H30	ERRFLG
0240	N8	DFLAGS
0241	N9	DBSECT
0242-0243	N10	BOOTAD
0244	N11	COLDST
0246	C2	DSKTIM
0247-026E	B51	LINBUF
026F	P24	GPRIOR
0270-0277	J3	PADDLO -- PADDL7
0278-027B	J1, J7, J8	STICKO -- STICK3
027C-0283	J4	PTRIGO -- PTRIG7
0284-0287	J2, J7, J9	STRIGO -- STRIG3
0289	D14	WMODE
028A	D10	BLIM
028B-028F	S10	unused
0290-0292	B4	TXTROW, TXTCOL
0293	B49	TINDEX
0294-0295	B52	TXTMSC
0296-029B	B53	TXTOLD
029D	B32	HOLD3
029E	B48	SUBTMP
029F	B31	HOLD2
02A0	B28	DMASK
02A1	B47	TMPLBT
02A2	B26	ESCFLG
02A3-02B1	B13	TABMAP
02B2-02B5	B14	LOGMAP
02B6	E9	INVFLG
02B7	B18	FILFLG
02B8-02BA	B46	TMPROW, TMPCOL
02BB	B9	SCRFLG
02BC	B20	HOLD4
02BE	E6	SHFLOK
02BF	B16	BOTSCR
02C0-02C3	B7, P27	PCOLRO -- PCOLR3
02C4-02C8	B8, P27	PCOLRO -- PCOLR4
02E4	A5	RAMSIZ
02E5-02E6	A2	MEMTOP
02E7-02E8	A1	MEMLO
02EA-02ED	G11	DVSTAT
02EE-02EF	D1	CHBAUDL, CHBAUDH
02F0	B1	CRSINH
02F1	E2	KEYDEL

02F2	E1	CH1
02F3	P25	CHACT
02F4	P26	CHBAS
02FA	B55	CHAR
02FB	B54	ATACHR
02FC	E3	CH
02FD	B17	FILDAT
02FE	B27	DSPFLG
02FF	E4	SSFLAG
0300	H1, H2	DCB/DDEVIC
0301	H3	DUNIT
0302	H4	DCOMND
0303	H5	DSTATS
0304-0305	H6	DBUFLO, DBUFHI
0306	H7	DTIMLO
0308-0309	H8	DBYTLO, DBYTHI
030A-030B	H9	DAUX1, DAUX2
030C-030D	D3	TIMER1
030E	D4	ADDCOR
030F	D8	CASFLG
0310-0311	D3	TIMER2
0312-0313	D5	TEMP1
0315	D6	TEMP3
0316	D7	SAVIO
0317	D2, H25	TIMFLG
0318	H28	STACKP
0319	H29	TSTAT
031A-033F	G12	HATABS
0340	G1, G2	IOCB, ICHID
0341	G3	ICDNO
0342	G4	ICCOM
0343	G5	ICSTA
0344-0345	G6	ICBAL, ICBAH
0346-0347	G7	ICPTL, ICPTH
0348-0349	G8	ICBLL, ICBLH
034A-034B	G9	ICAX1, ICAX2
034C-034F	G10	ICSPR
0350-035F	G2-G10	(IOCB #1)
0360-036F	G2-G10	(IOCB #2)
0370-037F	G2-G10	(IOCB #3)
0380-038F	G2-G10	(IOCB #4)
0390-039F	G2-G10	(IOCB #5)
03A0-03AF	G2-G10	(IOCB #6)
03B0-03BF	G2-G10	(IOCB #7)
03C0-03E7	F1	PRNBUF
03FD-047F	D9	CASBUF
0480-06FF	R2	User Area

FLOATING POINT PACKAGE VARIABLES

00D4-00D9	M1	FR0
00DA-00DF	M2	FRE
00E0-00E5	M3	FR1
00E6-00EB	M4	FR2
00EC	M5	FRX
00ED	M6	EEXP
00EE	M7	NSIGN
00EF	M8	ESIGN
00F0	M9	FCHRFLG
00F1	M10	DIGRT
00F2	M11	CIX
00F3-00F4	M12	INBUFF
00F5-00F6	M13	ZTEMP1
00F7-00F8	M14	ZTEMP4
00F9-00FA	M15	ZTEMP3
00FB	M24	RADFLG/DEGFLG
00FC-00FD	M16	FLPTR
00FE-00FF	M17	FPTR2
057E	M18	LBPR1
057F	M19	LBPR2
0580-05FF	M20	LBFEND, LBUFF
05E0-05E5	M21	PLYARG
05E6-05EB	M22	FPSCR/FSCR
05EC-05F1	M23	FPSCR1/SCR1

INDEX

The subject index contains three forms of references:

Section number, such as '3.'
Appendix, such as 'App B'
Variable ID from Appendix L, such as 'B7'.

ATARI standards	12
ATASCII	B54-55, 5, App D-G
attract mode	B10-12, 6,
bit mapped graphics	B28-B29, 5, App H
blackboard mode	3, N12, 7, 12
BNF	1
boot	3, 4, N3-10, 5, 7, 10
BREAK	E5, 6, 12
cartridge	3, 4, 7, 10
cassette baud rate determine	D1-D7
cassette-boot	3, N3-10, 7, 10
cassette device	D1-D15, 3, 5
Cassette Handler (C)	5
CIO (Central I/O Utility)	G1-25, 5, 9
CIO/user interface	G1-11, 5, App A, App B
CIO/Handler interface	G12-22, 9
CLOSE I/O command	5, 9
coldstart (see 'Power-up')	
color control	B7-8, 5, 6
control characters	B26-27, 5, App D
critical section	P1, 6
cursor	B1-4, 5
database	4
DCB (Device Control Block)	H1-9, 5, 9
DELETE I/O command	5
development system	13
device/filename specification	5
Device Handler	5, 9
device table	2, G12, 5, 7, 9
disk-boot	3, N3-10, 5, 7, 10
disk device	5
Disk File Manager (D)	K1-5, 5
Disk Handler (resident)	C1-2, 5
display device (screen)	B54-55, 5, App E, App H
Display Handler (S)	B1-55, 5
display list	4, P10
DOS (Disk Utilities)	L1, 12
DRAW I/O command	B17-25, 5
driving controller	J8-9
Educational System Format Cassettes 5	
error handling	G5, H5, H11-12, 9, App B-C

OPERATING SYSTEM C016555 -- INDEX