

Atari*System Reference manual

January 25, 2008

* Atari is a registered trademark of Atari Corp.

(c) 1987 By Bob DuHamel

Contents

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)	13
1.1. THE CIO CHANNELS	13
1.2. THE IOCB STRUCTURE	13
1.2.1. ICAX1 through ICAX6	15
1.2.2. OPENNING A CIO CHANNEL	15
1.2.3. USING AN OPEN CHANNEL	18
1.2.4. CLOSING A CHANNEL	21
1.3. THE DEVICE TABLE	21
1.4. SPECIAL COMMANDS	22
2. THE DISK OPERATING SYSTEM (D:)	25
2.1. DOS 2 SECTOR ASSIGNMENTS	26
2.2. DISK FILE STRUCTURE	27
2.3. INSIDE THE COMPUTER	28
2.4. DOS FILE NAME CONVENTIONS	28
2.4.1. WILD CARDS	29
2.5. USING DOS	30
2.5.1. READING THE DIRECTORY	30
2.5.2. USING NOTE AND POINT	31
2.5.3. STATUS REQUEST	32
3. USING THE DOS 2 UTILITIES (DUP.SYS)	33
3.1. THE DOS UTILITIES MENU	33
3.1.1. [A] DIRECTORY	33
3.1.2. [B] RUN CARTRIDGE	33
3.1.3. [C] COPY FILE	34
3.1.4. [D] DELETE FILE(S)	34
3.1.5. [E] RENAME	35
3.1.6. [F] LOCK FILE	35
3.1.7. [G] UNLOCK FILE	35
3.1.8. [H] WRITE DOS FILES	35
3.1.9. [I] FORMAT DISK	36
3.1.10. [J] DUPLICATE DISK	36
3.1.11. [K] BINARY SAVE	36
3.1.12. [L] BINARY LOAD	37
3.1.13. [M] RUN AT ADDRESS	37

Contents

3.1.14. [N] CREATE MEM.SAV	37
3.1.15. [O] DUPLICATE FILE	37
3.1.16. [P] FORMAT SINGLE	38
3.2. DOS 2.5 Utilities	38
3.2.1. RAMDISK.SYS	38
3.2.2. COPY32.COM	38
3.2.3. DISKFIX.COM	38
3.2.4. SETUP.COM	38
3.2.5. AUTORUN.SYS (DOS 2.0 and 2.5)	38
3.3. SPECIAL DOS INFORMATION	38
4. THE CASSETTE HANDLER (C:)	41
5. THE KEYBOARD HANDLER (K:)	43
5.1. KEYBOARD REPEAT DELAY AND RATE CONTROL	44
5.2. KEYBOARD CLICK	44
5.3. NON-HANDLER, NON-CIO KEYS	44
6. THE PRINTER HANDLER (P:)	47
7. SCREEN EDITOR (E:)	49
7.1. Editor control codes	49
8. THE DISPLAY HANDLER (S:)	51
8.1. TEXT MODE 0	51
8.2. TEXT MODES 1 AND 2	53
8.3. GRAPHICS MODES 3 THROUGH 11	54
8.4. USING THE SCREEN HANDLER	56
8.4.1. OPENING A CHANNEL TO THE SCREEN HANDLER	56
8.4.2. USING AN OPEN CHANNEL TO THE SCREEN	58
8.5. THE COLOR REGISTERS	59
8.5.1. The attract mode	61
9. THE RESIDENT DISK HANDLER	63
9.1. DISK HANDLER COMMANDS	63
9.1.1. GET SECTOR	63
9.1.2. PUT SECTOR	64
9.1.3. PUT SECTOR WITH VERIFY	64
9.1.4. GET STATUS	64
9.1.5. FORMAT DISK	65
10. SYSTEM INTERRUPTS	67
10.1. CHIP RESET	67
10.1.1. COLD START	67
10.1.2. WARM START	68

10.2. NON-MASKABLE INTERRUPTS (NMI)	69
10.3. DISPLAY LIST INTERRUPTS (DLIs)	69
10.4. VERTICAL BLANK INTERRUPTS (VBIs)	69
10.5. [SYSTEM RESET] INTERRUPT	70
10.6. INTERRUPT REQUESTS (maskable interrupts (IRQs))	70
10.7. SOFTWARE INTERRUPT (BRK instruction)	70
10.8. SYSTEM TIMERS	70
10.9. HARDWARE INTERRUPT CONTROL	71
10.10.WAIT FOR HORIZONTAL SYNC	72
11. THE FLOATING POINT ARITHMETIC PACKAGE	75
11.1. USE OF THE FLOATING POINT PACKAGE	75
11.2. MAKING THE CALL	76
12. Boot software formats	83
12.1. DISK BOOTED SOFTWARE	83
12.1.1. FORMAT OF A DISK BOOTED PROGRAM	83
12.2. CASSETTE BOOTED SOFTWARE	84
12.3. CARTRIDGE BOOTED SOFTWARE	84
12.3.1. THE CARTRIDGE HARDWARE	85
13. THE SERIAL INPUT/OUTPUT INTERFACE (SIO)	87
13.1. USING THE SIO INTERFACE	88
13.2. THE SERIAL CONNECTOR	90
14. THE HARDWARE CHIPS	93
14.1. THE GTIA CHIP	93
14.2. THE ANTIC CHIP	93
14.3. THE POKEY CHIP	94
14.4. THE PIA CHIP	94
15. DISPLAY LISTS	95
15.1. DISPLAY LIST INSTRUCTIONS	95
15.1.1. ANTIC MODE 2 (Graphics 0)	96
15.1.2. ANTIC MODE 3	96
15.1.3. ANTIC MODE 4 (graphics 12 on XL and XE)	97
15.1.4. ANTIC mode 5	97
15.1.5. ANTIC MODE 6 (Graphics 1)	98
15.1.6. ANTIC MODE 7 (Graphics 2)	98
15.1.7. ANTIC MODE 8 (Graphics 3)	98
15.1.8. ANTIC MODE 9 (Graphics 4)	98
15.1.9. ANTIC MODE A (Graphics 5)	98
15.1.10.ANTIC MODE B (Graphics 6)	98
15.1.11.ANTIC MODE C	99

Contents

15.1.12.ANTIC MODE D (Graphics 7)	99
15.1.13.ANTIC MODE E (Graphics 15 on XL and XE)	99
15.1.14.ANTIC mode F (Graphics 8, 9, 10 and 11)	99
15.2. DISPLAY LIST EXAMPLES	99
15.2.1. SAMPLE DISPLAY LIST	99
15.3. MULTIPLE DISPLAYS	100
15.4. DISPLAY LIST INTERRUPTS	101
15.5. SCROLLING	101
15.5.1. VERTICAL SCROLLING	101
15.5.2. HORIZONTAL SCROLLING	102
16. PLAYER AND MISSILE (PM) GRAPHICS	105
16.1. SETTING UP PM GRAPHICS	106
16.2. P/M PRIORITY	109
16.3. COLLISIONS	111
17. SOUND	115
17.1. GENERATING SOUNDS	115
17.2. ADVANCED SOUND	116
17.2.1. AUDIO TIMER INTERRUPTS	117
17.3. THE CONSOLE SPEAKER	117
18. THE JOYSTICK PORTS	119
18.1. The port control registers	119
18.2. THE GAME PADDLE REGISTERS	121
18.3. THE LIGHT PEN REGISTERS	121
19. MISC HARDWARE REGISTERS AND INFORMATION	123
19.1. VERTICAL LINE COUNTER	123
19.2. SERIAL PORT REGISTERS	123
19.3. SERIAL PORT INPUT AND OUTPUT DATA	124
19.4. HARDWARE CHIP MEMORY ALLOCATION	124
20. THE XL AND XE MODELS	127
20.1. BASIC B BUGS	127
20.2. NEW OPERATING SYSTEM PROBLEMS	127
20.3. 130XE MEMORY MANAGEMENT	127
20.4. THE XL PARALLEL PORT	128
20.5. FINE SCROLLING	129
20.6. OTHER ADDRESSES	130
A. HARDWARE REGISTERS	131
B. OPERATING SYSTEM EQUATES	141

C. MEMORY USE

157

Contents

Author

Bob Duhamel

can be reached at AtariAge Forums (<http://www.atariage.com/>)

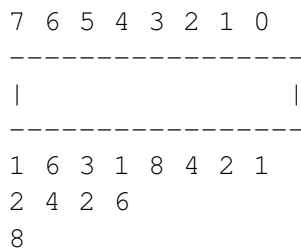
This manual contains highly technical information. Such information is provided for those who know how to use it. To understand the advanced information you are expected to know 6502 assembly language. If you are new to programming, concentrate on the parts which discuss BASIC commands.

Addresses are usually given in both hexadecimal and decimal numbers. The operating system equate names are given in capital letters with the address following in brackets. The decimal address is in parenthesis within the brackets. For example:

```
DOSVEC [$000A, 2 (10)]
name hex dec
```

The ",2" after the hexadecimal number means that this address requires two bytes to hold its' information. Any address called a "vector" uses two bytes whether noted or not. Control registers and some other bytes of memory are shown in the following format

Figure 0.1.: Register format



The numbers on top are the bit numbers. Bit 7 is the Most Significant Bit (MSB) and bit 0 is the Least Significant bit (LSB). The numbers on the bottom are the bit weights. These are useful when changing memory with decimal numbers, as you would in BASIC. For example, to set bit 4 of a register to 1, without changing any other bits you would add 16 to the decimal number already in the register. To reset the same bit to 0, you would subtract 16 from the number in the register. This is exactly what the command GRAPHICS 8+16 does. It sets bits 3 and 4 of a graphics mode control register.

MSB and LSB may also mean Most Significant Byte or Least Significant Byte, depending on context.

Contents

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)

The ATARI computer uses a very easy-to-use input and output system called the Central Input/Output utility, or CIO. Nearly all input or output passes through this utility. CIO uses eight "channels" as paths for I/O. There are not really separate channels for I/O but the computer acts as if there were. Each channel is controlled by a 16 byte block of memory called an Input/Output Control Block or IOCB. The channels are used by putting the proper numbers in the proper IOCB bytes then jumping to the CIO routine. In BASIC, complete I/O operations can be as easy as typing a command such as LPRINT. In this case BASIC does all the work for you.

1.1. THE CIO CHANNELS

There are eight CIO channels, numbered from 0 to 7. In BASIC some channels are reserved for BASIC's use.

BASIC CIO channel assignments

- Channel 0 Permanently assigned to the screen editor
- 6 Used for graphics commands
- 7 Used for the Cassette, disk and printer

Channels 6 and 7 are free to use when they are not being used by BASIC. With machine language, all of the channels are available to the programmer.

1.2. THE IOCB STRUCTURE

The IOCB for channel zero starts at address \$0340 (decimal 832). This is the only address you need to know. Indexing from this address is used to find all of the other bytes. Below are the names and uses of the IOCB bytes. IOCB bytes and uses:

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)

ADDRESS	NAME	EXPLANATION
\$0340	ICHID	handler Identifier
\$0341	ICDNO	device number (disk)
\$0342	ICCOM	command
\$0343	ICSTA	status
\$0344	ICBAL	buffer address (low byte)
\$0345	ICBAH	buffer address (high byte)
\$0346	ICPTL	address of put byte
\$0347	ICPTH	routine (used by BASIC)
\$0348	ICBLL	buffer length (low byte)
\$0349	ICBLH	buffer length (high byte)
\$034A	ICAX1	auxiliary information
\$034B	ICAX2	auxiliary information
\$034C	ICAX3	the remaining auxiliary
\$034D	ICAX4	bytes are rarely used
\$034E	ICAX5	-
\$034F	ICAX6	-

Table 1.1.: IOCB Memory Addresses and usage

ICHID

When a channel is open, the handler I.D. contains an index to the handler table. The handler table (to be discussed later) holds the address of the device handling routines. When the channel is closed ICHID contains \$FF.

ICDNO

The device number is used to distinguish between multiple devices with the same name, such as disk drives.

ICCOM

The command byte tells CIO what operation to perform. (See Table 1.2)

ICSTA

The status byte contains an error code if something goes wrong. If bit 7 is 0 there have been no errors.

Command	HEX	DEC
+Open	\$03	3
+close	\$0C	12
get	\$07	7
put	\$0B	11
input	\$05	5
print	\$09	9
+status request	\$0D	13
+*special	>\$0D	>13

+ command may be made to a closed channel

* device specific commands

Table 1.2.: CIO command codes

ICBAL and ICBAH

Before a channel is opened, the buffer address bytes are set point to the block of memory which contains the name of the device the channel is to be opened to. Before actual input or output these bytes are set to point to the block of memory where the I/O data is stored or is to be stored.

ICPTL and ICPTH

The put routine pointer is set by CIO to point to the handlers' put-byte routine. When the channel is closed the pointer points to the IOCB-closed routine. This pointer is only used by BASIC.

ICBLL and ICBLH

The buffer length bytes show the number of bytes in the block of memory used to store the input or output data. (See ICBAL and ICBAH.) If the amount of data read during an input operation is less than the length of the buffer, the number of bytes actually read will be put in ICBLL and ICBLH by CIO.

1.2.1. ICAX1 through ICAX6

The auxiliary information bytes are used to give CIO or the device any special information needed.

1.2.2. OPENNING A CIO CHANNEL

Before using a CIO channel it must be assigned to an I/O device. In machine language you start by putting the channel number in the four high bits of the 6502 X register (X = \$30 for

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)

channel three). Next you place the necessary codes (parameters) into IOCB 0 indexed by X. The X register will cause the numbers to be offset in memory by 16 times the channel number. This puts the numbers into the correct IOCB instead of IOCB 0. See Figure 1.1 for the parameters used to open a channel.

Figure 1.1.: Channel-open parameters:

```

ICCOM open code
ICBAL address of device name
ICBAH in memory
ICAX1 direction code
ICAX2 zero

```

The direction code byte in ICAX1 takes on the format shown in Figure 1.2.

Figure 1.2.: ICAX1 format for opening a channel

```

      7 6 5 4 3 2 1 0
      -----
ICAX1 |           W R           |
      -----
              8 4 2 1
W 1 = open for output (write)
R 1 = open for input (read)

```

ICAX1 may have the following data

HEX	DEC	Operation
\$04	4	input
\$08	8	output
\$0C	12	input and output (cannot change the length of a disk file)

Table 1.3.: CIO direction codes

ICBAL and ICBAH point to the device name stored in memory. The device and file name must be followed by 0 or \$9B (decimal 155). Once the parameters are set, jumping (JSR) to the CIO vector (CIOV) at address \$E456 (58454) will cause the channel to be opened. In the following example a basic knowledge of assembly language is assumed.

Routine to open channel 1 to the keyboard:

```

ICHID = $0340

```


1.2. THE IOCB STRUCTURE

```
ICCOM = ICHID+2
ICAX1 = ICHID+10
ICAX2 = ICHID+11
IOCB1 = $10 channel in four high bits
CIOV = $E456
OPEN = $03
OREAD = $04 ;open for input
ERROR = (address of error handling routine)
;
START LDX IOCB1
LDA OPEN
STA ICCOM,X
LDA <NAME
STA ICBAL,X
LDA >NAME
STA ICBAH,X
LDA OREAD
STA ICAX1,X
LDA #0
STA ICAX2,X
JSR CIOV
BPL OK
JMP ERROR
;
NAME .BYTE "K:",$9B
OK (program continues here)
```

To open a CIO channel in BASIC the OPEN command is used. BASIC OPEN command format:

```
OPEN #channel,aux1,aux2,device:file name
```

aux1 = direction code

aux2 = special code

To open channel 1 to the keyboard in BASIC Type:

```
OPEN #1,4,0,"K:"
```

The third parameter, aux2, is a rarely used special parameter. One use is to keep the screen from erasing when changing graphics modes. The fourth parameter is the device to open the channel to. It may be either a string in quotes or a string variable.

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)

Name	Device
C	cassette recorder
*D	disk drive
E	screen editor
K	Keyboard
P	printer
*R	RS 232 I/O port
S	screen handler

* Uses a non-resident handler loaded by the device at power-up.

Table 1.4.: CIO device names

CIO device names The device name must usually be followed by a colon. With the disk drive a file name is expected after the device name. The screen handler is used for graphics. The screen editor uses both the keyboard handler and the screen handler to work between the keyboard and screen.

1.2.3. USING AN OPEN CHANNEL

Once a channel is opened to a device you have several options:

INPUT: (ICCOM = \$05) The computer reads data from the device until a carriage-return is read (decimal number 155, hex \$9B) or the end of the file (EOF) is reached. A carriage return is also known as an End-Of-Line or EOL. The IOCB input parameters are shown in Figure 1.3.

```
ICCOM get record code
ICBAL address of buffer to
ICBAH store the data in
ICBLL length of the data
ICALH buffer
```

Figure 1.3.: IOCB input parameters

The following routine demonstrates the input command in assembly language. Some of the equates are in the channel opening example above.

Input routine:

```
GETREC = $05
BUFF = (address to store data at)
BUFLen = (number of bytes available at storage address)
:
```

1.2. THE IOCB STRUCTURE

```
LDX IOCB1
LDA GETREC
STA ICCOM,X
LDA < BUFF
STA ICBAL,X
LDA > BUFF
STA ICBAH,X
LDA < BUFLen
STA ICBLl,X
LDA > BUFLen
STA ICBLH,X
JSR CIOV
BPL OK2
JMP ERROR
:
OK2 (continues if no errors)
```

If the data retrieved is shorter than the prepared buffer, the number of bytes actually read will be put into ICBLl and ICBLH. In BASIC, the INPUT command is used. BASIC INPUT command format:

```
INPUT #channel,string variable
```

or

```
INPUT #channel,arithmetic variable
```

For example:

```
INPUT #1,IN$
```

The above commands will cause the data from the device to be put into the specified buffer (IN\$ in the BASIC example) until an EOL is reached. If the INPUT statement is used again, without closing the channel, the computer will get more data from the device until another EOL is read or the end of the file is reached. The new data will write over the old data in the input string or buffer. If an arithmetic variable is used, only numbers can be input.

PRINT: (ICCOM = \$09) In assembly language the print command is identical to the input command. The only difference is that the PUTREC code (\$09) is stored in ICCOM. Of course the buffer bytes of the IOCB then specify the block of memory to be output from rather than input to. With the print command, EOLs within the string or buffer are ignored but an EOL is placed at the end of the data when sent to the device. In BASIC, the PRINT command is used like INPUT except you want to use a semicolon instead of a comma to separate parameters. For example:

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)

```
PRINT #1;OUT$
```

or

```
PRINT #1;"HELLO"
```

If you use a comma, ten space characters will be sent before the string. If the print command is used again, without closing the channel, the new data will be appended to the end of the data previously sent. Old data will not be written over.

GET: (ICCOM = \$07) In BASIC this command inputs a single byte of data from the device. EOLs are ignored. In BASIC, GET is used like INPUT except an arithmetic variable must be used. For example:

```
GET #1, IN
```

If the get command is used again the next byte from the device will be read. If the end of a file is reached an error will occur. There is no command in BASIC to input an entire file without stopping at each EOL. If you wish to ignore EOLs while reading a file to a string, you must use the GET command. Each byte of data is then put into the string by the program.

EXAMPLE:

```
10 OPEN #1, 4, 0, "D:TEST"
20 TRAP 60:REM GOES TO LINE 60 WHEN END OF FILE ERROR OCCURS
30 GET #1, IN
40 IN$(LEN(IN$)+1)=CHR$(IN)
50 GOTO 30
60 CLOSE #1
```

In assembly language, the get command can be used to get any number of bytes from the device. It works just as INPUT does except EOLs are ignored.

```
ICCOM get-character (single byte) code
ICBAL \
ICBAH same as in input
ICBLL
ICBAH /
```

Figure 1.4.: IOCB get-byte parameters

Other than the ICCOM code (GETCHR = \$07) this command is identical to the input command.

PUT: (ICCOM = \$0B) In BASIC, PUT is the opposite of GET. It outputs a single byte from a variable to the device. PUT is used the same as GET. For example:

```
PUT #1,OUT
```

In assembly language, the command byte of the IOCB is loaded with the put-character code (PUTCHR = \$0B). Otherwise the PUT command is identical to GET.

1.2.4. CLOSING A CHANNEL

Closing a channel frees it for use by another device or for changing parameters. In assembly language the close code is put into the command byte of the IOCB then the CIOV call is made. IOCB close command:

```
CLOSE = $0C
:
LDX IOCB1
LDA CLOSE
STA ICCOM,X
JSR CIOV
```

In BASIC, use the CLOSE command followed by the channel number.

```
CLOSE #1
```

With the disk drive, the file name is not put into the directory until the channel is closed.

1.3. THE DEVICE TABLE

CIO uses a jump table located at \$031A (794). When a CIO call is made, CIO searches the table for the one-byte device name. The two bytes following the device name contain the address of the device handler's vector table. CIO searches the device table from the end, \$033D (829) to the beginning. This way, if a custom handler has been substituted for a resident handler, the custom handler will be found first. (custom handlers cannot be inserted directly in the place of resident handlers in the device table.) Each handler has its' own vector table. This vector table is 16 bytes long. The two-byte vectors point to the various handler routines. The vectors are stored in the vector table in the following order:

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)

```
open
close
get byte
put byte
get stat
special
JMP init code (3 bytes)
```

Figure 1.5.: Handler vector table

The open routine should validate the ICAX parameters and check for illegal commands. The close routine should send any remaining data in the buffer to the device, mark the End-Of-File and update any directories, etc. The get byte routine should get a byte from the device or the handler buffer and put it in the 6502 A register. Handlers with long timeouts must monitor the break key flag and put a \$80 in the 6502 Y register if the [BREAK] key is pressed.

The put byte routine should send the byte in the 6502 A register to the device or handler buffer. If the buffer fills, it should be sent to the device. BASIC can call the put byte routine without using CIO.

The get status routine may get 4 bytes of status information from the device and put them in DVSTAT [\$02EA] to DVSTAT+3.

For special commands the handler must examine the command byte and find the proper routine entry point. In all cases the status (error code) of the operation should be put in the 6502 Y register. To be compatible with all versions of the operating system, the handler must redirect DOSINI [\$000C,2 (12)] for initialization upon reset. This initialization must restore the vectors in the handler vector table and jump to the original DOSINI vector.

1.4. SPECIAL COMMANDS

Some devices have special CIO commands. These are known as device specific commands. In assembly language these commands are executed just as any other CIO command is. In BASIC the XIO command is used. An example of the XIO command is:

```
XIO command code #channel,aux1,aux2,device:file name
```

To open a channel with the XIO command instead of the OPEN command use:

```
XIO 3 #1,4,0,"K:"
```

Note that the above command is identical to the OPEN command except "XIO 3" is used instead of "OPEN". Also note that \$03 is the IOCB open code for ICCOM.

1.4. SPECIAL COMMANDS

Name	Location	Comment
DOSINI	\$000C,2 (12)	initialization vector
BRKKEY	\$0011 (17)	break key flag
ICHID	\$0340 (832)	start of IOCBs
ICDNO	\$0341 (833)	device number
ICCOM	\$0342 (834)	CIO Command
ICSTA	\$0343 (835)	Device Status
ICBAL	\$0344 (836)	Buffer Address (Lowbyte)
ICBAH	\$0345 (837)	Buffer Address Highbyte
ICPTL	\$0346 (838)	
ICPTH	\$0347 (839)	
ICBLL	\$0348 (840)	
ICBLH	\$0349 (841)	
ICAX1	\$034A (842)	Auxillary Byte 1
ICAX2	\$034B (843)	Auxillary Byte 2
HATABS	\$031A,16 (794)	device handler table
CIOV	\$E456 (58454)	CIO entry vector

Table 1.5.: Useful database variables and OS equates

1. THE CENTRAL INPUT/OUTPUT UTILITY, (CIO)

2. THE DISK OPERATING SYSTEM (D:)

The disk operating system program (DOS) is also called the file management system (FMS). DOS is not a permanent part of the computer, it is loaded in upon power-up if a disk drive is attached to the computer.

When the computer is turned on, one of the first things it does is send a request to the disk drive to load DOS into the computer. This startup operation is called booting. The word boot is short for bootstrapping – the start-up process of early computers. The term comes from "lifting one's self by one's boot straps". Anytime the disk boots, the computer tries to read a program starting at sector 1 and continuing in sequence. If the disk has DOS on it, the first three sectors, called the boot record, have a program which loads the DOS.SYS file. If there is no DOS.SYS file on the disk the computer will display a "Boot Error" (See Figure 2.1),

```
-----  
| BOOT ERROR      |  
| BOOT ERROR      |  
| BOOT ERROR      |  
| BOOT ERROR      |  
| BOOT ERROR      |  
| BOOT ERROR      |  
| (etc.)          |  
|                 |  
|                 |  
-----
```

Figure 2.1.: Boot Error

When a disk is formatted, the drive read/write head passes over the entire disk and puts magnetic marks on it. These marks divide the disk into 32 concentric tracks. With DOS 2.0 each track is divided into 18 sectors, each holding 128 bytes of data. With DOS 2.5 there are 32 sectors per track giving a total of 1,024 sectors. Each sector on the disk is marked with a reference number from 1 to 720. Unfortunately, the writers of DOS 2.0 didn't know this so they wrote the DOS to use sectors numbered from 0 to 719. As a result, DOS 2.0 cannot access sector 720. The designers of the disk drive were the guilty party in this case. It is normal to number from 0 in computers. With DOS 2.5, sectors 720 - 1,024 can be accessed normally.

Sector 720 can be accessed using the computer's resident disk handler. Some software writers use sector 720 to hide special information to make their programs difficult to copy.

2. THE DISK OPERATING SYSTEM (D:)

2.1. DOS 2 SECTOR ASSIGNMENTS

Sectors 1 through 3 are called the boot record. They contain a program which loads the DOS.SYS file into memory. Sector 360 is called the Volume Table of Contents or VTOC. The main purpose of the VTOC is to keep track of what sectors are occupied. Bytes 3 and 4 of the VTOC tell how many sectors are available. Starting at byte 10 is the Volume Bit Map. Each byte in the VBM tells the status of eight sectors. If a bit is a 1 the sector is available. If a bit is a 0 the sector is occupied.

Sectors 361 through 368 contain the disk directory. Each directory sector holds eight file names. The first byte of a file name is called the flag byte (Figure 2.2). It tells the status of that file.

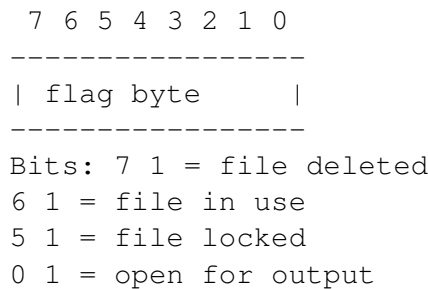


Figure 2.2.: Directory flag byte

The next two bytes tell how many sectors are in the file. The two bytes after them tell the starting sector of the file. The last 11 bytes contain the file name. Sector 720 cannot be accessed with DOS 2.0.

The boot record, VTOC, directory and sector 720 use 13 sectors. This leaves 707 sectors for storing files with DOS 2.0. Each file sector has 125 bytes of data. The last three bytes tell how many bytes of the sector are used, what directory entry the sector belongs to and which sector is next in the file (Figure 2.3).

If the directory number does not match the order of the file name in the directory, an error 167 (file number mismatch) will occur. As a file is written to an empty disk it is put in consecutive sectors, 125 bytes at a time. After the file is written, the VTOC and directory are updated. When new files are written they also use consecutive sectors.

When a file is deleted the status bit of the directory is changed to show that the file has been deleted. DOS then tracks the file, sector by sector, to find what sectors are used. Finally the VTOC is updated to show that the deleted file's sectors are available for new files.

The file is not erased from the disk; only the VTOC and directory are changed. When a file is deleted, an "island" of free sectors may be left on the disk. When a new file is then written to the disk it will first use these new free sectors. When the island is used up, DOS will skip over the occupied sectors to the next free sector. This is the reason for the sector link. A file can end up with its sectors scattered all over a disk. It can be complicated but it's efficient.

2.2. DISK FILE STRUCTURE

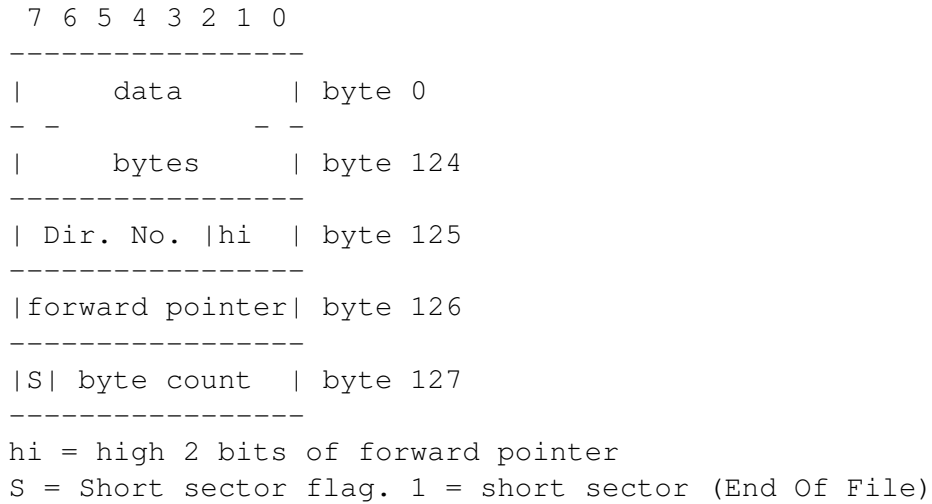


Figure 2.3.: File sector structure

2.2. DISK FILE STRUCTURE

The first few bytes of a file may tell DOS or another program what kind of file it is. These information bytes are called a header. A text file is any file which has no header. A listed BASIC program is a type of text file. A letter from a word processor is another.

A binary load file is a file intended to load to a specific address in memory. The first two bytes of a binary load file are decimal 255. The next two bytes hold the address at which the file is to load. The last two header bytes tell the ending address for the file. If the file is a program and is to run automatically, the initialization and run address are appended to the end of the file.

Decimal		Hexadecimal
255	identifier	FF
255		FF
0	start	00
7		07
15	end	FF
8		08

Table 2.1.: binary load file header

The file shown in Table 2.1 would load at address \$0700 (1792 decimal) and end at address \$08FF (2063). If a binary load file has initialization and run address appended to it they take on the following format (Table 2.2 and 2.3).

2. THE DISK OPERATING SYSTEM (D:)

CHR	Decimal		Hexadecimal
init address format			
[b]	226	identifier	E2
	2		02
[c]	227		E3
	2		02
	n	address	nn
	n		nn

[]=inverse video

Table 2.2.: Run tailer

CHR	Decimal		Hexadecimal
run address format			
[diamond]	224	identifier	E0
	2		02
[a]	225		E1
	2		02
	n	address	nn
	n		nn

[]=inverse video

Table 2.3.: Init tailer

A program which doesn't need special initialization can be run at the init address. Otherwise, an RTS instruction is expected at the end of the initialization section. The computer will then jump to the run address if specified.

2.3. INSIDE THE COMPUTER

DOS uses the computer's CIO utility. When a DOS disk is booted a non-resident handler is loaded into memory. A new handler name, D, is then added to the handler table. When CIO is called with a device name of D: or Dn:, CIO will search the handler table for that device name. If the 'D' is found, the next two bytes in the table point to the DOS entry address.

2.4. DOS FILE NAME CONVENTIONS

DOS is unique among CIO handlers in that it requires an eight character file name to follow the device name. This file name may be followed by a period and then a three character extender.

EXAMPLES:

D:TEST, D2:FIREMAN, D:VENTURE.EXE, D:CHAPTER.001

The D2: is used for drive number two if present. The file name must use upper-case letters or numbers. The first character must always be a letter.

2.4.1. WILD CARDS

The characters * and ? may be used as wild cards. * means any combination of characters and ? means any single character.

EXAMPLES:

D:P*

any file beginning with P and without an extender

D:* .EXE

any file with the extender .EXE

D:*.*

any file

D:F?REMAN

one unknown character, FIREMAN or FOREMAN will match

Wild cards can only be used to load, delete, lock and unlock files. When loading a file using wild cards, only the first matching file will be loaded. When renaming a file, both the new and old names are expected after the device name.

EXAMPLE:

D:OLDNAME .BAS, NEWNAME .BAS

To format a disk, only the device name (D: or Dn:) is needed.

2. THE DISK OPERATING SYSTEM (D:)

```
    7 6 5 4 3 2 1 0
    -----
    |           W R D A |
    -----
D 1 = open to read the directory instead of a file
A 1 = append data to the end of the file
```

Figure 2.4.: ICAX1 for disk open

2.5. USING DOS

When a CIO channel is opened to the disk drive it must actually be opened to a specific file on the disk. The device name in the open command must be followed by a file name. When a channel is opened to the disk, two special parameters may be used in ICAX1 (Figure 2.4).

This gives the following extra ICAX1 options.

HEX	DEC	Description
\$06	6	open to read directory
\$09	9	output, append to the end of an existing file

Table 2.4.: Disk specific ICAX1 options

2.5.1. READING THE DIRECTORY

When the directory is read, each file name is treated as if it were followed by an EOL. A loop must be used to read all of the file names in the directory. The last entry read is the free sector count. After it is read, another read operation will result in an End-Of-File error.

The disk drive has a number of device specific commands other than the regular CIO commands. From BASIC the XIO command is used to access these commands. The XIO command allows you to directly load the IOCBs from BASIC. Each parameter of the XIO command places values in certain bytes of an IOCB.

XIO command format:

```
XIO command channel,aux1,aux2,device:file name
```

Note that the parameters resemble the BASIC OPEN command. The BASIC OPEN command is identical to it's equivalent XIO command.

Command	Hex	Dec
RENAME XIO	\$20	(32)
DELETE XIO	\$21	(33)
LOCK XIO	\$23	(35)
UNLOCK XIO	\$24	(36)
POINT XIO	\$25	(37)
NOTE XIO	\$26	(38)
FORMAT XIO	\$FE	(254)

Table 2.5.: XIO commands specific to the disk drive

EXAMPLES:

```
XIO 33 #1,0,0,"D:JUNK"
```

= delete file named D:JUNK

```
XIO 32 #1,0,0,"D:OLD,NEW"
```

= change name of D:OLD to D:NEW

2.5.2. USING NOTE AND POINT

NOTE and POINT can also be used directly from BASIC. NOTE finds the current position of the read/write head on the disk. POINT moves the read/write head to the desired position. The command format for NOTE and POINT is as follows:

```
NOTE \
      channel,sector,byte
POINT/
```

EXAMPLE:

```
NOTE #1,SECT,BYTE
```

BASIC requires the sector and byte parameters in both commands to be variables. Fixed numbers cannot be used. If you try to do a POINT to a sector outside the file the channel is open to, a point error will occur. Care may need to be taken to be sure the file being accessed is in contiguous sectors on the disk. If it is not, it will be difficult to know where to do points to.

One use of NOTE is to use the command immediately after opening a channel to a disk file. After the NOTE command, the parameter variables contain the coordinates of the first byte of the file. They can then be used as a reference for the POINT command. In assembly language, ICAX3 and ICAX4 are used for the sector number (lsb,msb). ICAX5 is used for the byte number.

2. THE DISK OPERATING SYSTEM (D:)

2.5.3. STATUS REQUEST

If the status request command is used, one of the following values will be found in ICSTA and the 6502 Y register.

HEX	DEC	Description
\$01	1	OK
\$A7	167	file locked
\$AA	170	file not found

Table 2.6.: Status request codes

3. USING THE DOS 2 UTILITIES (DUP.SYS)

If you boot a DOS 2.x disk with no cartridge in the slot or with BASIC disabled (by holding the OPTION key), DOS will try to load the file named DUP.SYS. This is the disk utility file. When using BASIC, typing DOS [RETURN] will load the DUP.SYS file. When the utilities are loaded the menu will appear on the screen.

3.1. THE DOS UTILITIES MENU

```
DISK OPERATING SYSTEM II VERSION 2.0S
COPYRIGHT 1980 ATARI
A. DISK DIRECTORY      I. FORMAT DISK
B. RUN CARTRIDGE      J. DUPLICATE DISK
C. COPY FILE          K. BINARY SAVE
D. DELETE FILE(S)    L. BINARY LOAD
E. RENAME FILE        M. RUN AT ADDRESS
F. LOCK FILE          N. CREATE MEM.SAV
G. UNLOCK FILE        O. DUPLICATE DISK
H. WRITE DOS FILES
SELECT ITEM OR [RETURN] FOR MENU
```

3.1.1. [A] DIRECTORY

After pressing [A] [RETURN] you will get the prompt:

```
DIRECTORY--SEARCH SPEC, LIST FILE?
```

If you want to see the entire directory just press [RETURN] again. If you wish, you may type in a specific file name (D: is optional) or wild cards to search for. If you specify a search spec only matching files will be displayed. If you want, you can have the directory sent to another device. To do this type a comma and the device name. For example, if you type ,P: the directory will be sent to the printer.

3.1.2. [B] RUN CARTRIDGE

If a cartridge was inserted or BASIC was not disabled when the computer was turned on, [B] [RETURN] will run that cartridge or BASIC.

3. USING THE DOS 2 UTILITIES (DUP.SYS)

3.1.3. [C] COPY FILE

This option will copy a file to another part of the same disk (with a different file name) or copy from one disk drive to another. When you press [C] [RETURN] you will be given the prompt:

```
COPY--FROM, TO
```

Type the devices and file names separated by a comma. EXAMPLES:

```
FOREMAN, FIREMAN
```

or

```
D1:TEST, D2:TEST
```

The first example will copy to the same disk. The second example will copy from disk drive one to disk drive two. If you want to have the first file appended to the end of the second file type /A after the file names. EXAMPLE:

```
RUNMENU.EXE, AUTORUN.SYS/A
```

If the files are binary load files, this will cause both files to be saved as one file. When the load command is used they will both be loaded and run.

3.1.4. [D] DELETE FILE(S)

After pressing [D] [RETURN] you will get the prompt:

```
DELETE FILE SPEC
```

After typing the file name you will be asked to confirm the file to delete.

```
DELETE FILE SPEC  
DELETE-D1:JUNK ARE YOU SURE?
```

Press [Y] if the correct file is displayed. If you use wild cards you will be asked to confirm each matching file.

3.1.5. [E] RENAME

Upon typing [E] [RETURN] you will be given the prompt:

```
RENAME-GIVE OLD NAME, NEW
```

Type the file name you want to change and the new name separated by a comma. EXAMPLE:

```
COLT, HORSE
```

WARNING! Do not rename a file to a name which already exists on the disk. You will end up with a duplicate file name and will not be able to access one of them. Attempting to rename or delete one of them will rename or delete both. The only way to fix a duplicate file name is with a sector editor or other special utility.

3.1.6. [F] LOCK FILE

A locked file cannot be written to, renamed or deleted. To lock a file type [F] [RETURN]. You will get the prompt:

```
WHAT FILE TO LOCK?
```

Type the file name you want to lock. Wild cards will cause all matching files to be locked.

3.1.7. [G] UNLOCK FILE

Used the same as lock.

3.1.8. [H] WRITE DOS FILES

This option will write the DOS.SYS and DUP.SYS files to a formatted disk. When you type [H] [RETURN] you will receive the prompt:

```
DRIVE TO WRITE DOS FILES TO?
```

Type the number of the drive. If the drive contains a formatted disk the dos files will be written to it.

3. USING THE DOS 2 UTILITIES (DUP.SYS)

3.1.9. [I] FORMAT DISK

This option formats a new disk or erases a disk with files on it. Typing [I] [RETURN] will get you the prompt:

```
WHICH DRIVE TO FORMAT
```

Be sure you have the correct disk in the proper drive then type the drive number. It is impossible to recover files on a disk formatted by accident.

While the disk is being formatted the drive will check to be sure the disk is formatted correctly. If not, the drive attempt to format the disk again. If the disk is defective the drive will not finish the formatting process.

3.1.10. [J] DUPLICATE DISK

This option will copy an entire disk except for sectors listed as free in the VTOC. Some programs are copy-protected by changing the VTOC to show that some occupied sectors are empty. For such disks, a program which copies the entire disk is needed. When you press [J] [RETURN] you will be given the prompt:

```
DUP DISK--SOURCE,DEST DRIVES?
```

If you are using only one disk drive, type 1,1. If you have only one drive you will be told when to swap disks.

3.1.11. [K] BINARY SAVE

This option saves a block of memory as a binary load file. When you type [K] [RETURN] you will be given the prompt:

```
SAVE-GIVE FILE, START, END (, INIT, RUN)
```

Type the desired file name and a comma. Now type the start and end addresses of the memory block to be saved, in hexadecimal numbers, separated by commas. If the file is a program which is to automatically run when loaded, give the initialization address, if needed, then the run address. EXAMPLE:

```
CHASE.EXE,0700,09FF,,0700
```

This will save the block of memory from address 0700 to 09FF. The program is not initialized before running so there is no address typed after the third comma. When the program is loaded the computer will jump to address 0700, as specified in the last parameter, to run the program.

3.1.12. [L] BINARY LOAD

To load a binary file type [L] [RETURN]. You will get the Prompt:

```
LOAD FROM WHAT FILE?
```

Type the file name and the file will be loaded. If wild cards are used, only the first matching file will be loaded.

3.1.13. [M] RUN AT ADDRESS

Typing [M] [RETURN] will get the prompt:

```
RUN FROM WHAT ADDRESS?
```

Type the hexadecimal address of the program you want to run.

3.1.14. [N] CREATE MEM.SAV

A MEM.SAV file is used by BASIC and some other programs to save the part of memory which the DUP.SYS file loads into. If there is no MEM.SAV file on the disk when you go to the DOS utilities, you will lose that part of memory. With BASIC you will lose your program. When you type [N] [RETURN] you will get the prompt:

```
TYPE Y TO CREATE MEM.SAV
```

Typing [Y] [RETURN] will create a MEM.SAV file on the disk in drive one.

3.1.15. [O] DUPLICATE FILE

This option is used to copy a file from one disk to another, using only one disk drive. When you type [O] [RETURN] you will get the prompt:

```
NAME OF FILE TO MOVE?
```

If you use wild cards you will be asked to swap disks for each matching file.

DOS 2.5 also has option:

3. USING THE DOS 2 UTILITIES (DUP.SYS)

3.1.16. [P] FORMAT SINGLE

DOS 2.5 normally formats disks to use "enhanced" density. This option will format a disk in single density for use with the 810 drive.

3.2. DOS 2.5 Utilities

DOS 2.5 also has some special utilities on the master disk. Use the binary load option to run them.

3.2.1. RAMDISK.SYS

This program will cause the extra bank of memory in the 130XE to act like a disk drive (called D8:). If this program is on the disk it will automatically run. It need not be renamed to AUTORUN.SYS.

3.2.2. COPY32.COM

Copies DOS 3 files to DOS 2.

3.2.3. DISKFIX.COM

Can make certain "repairs" to a disk, such as restoring deleted files.

3.2.4. SETUP.COM

Used to change the default configuration of DOS.

3.2.5. AUTORUN.SYS (DOS 2.0 and 2.5)

This program is needed to operate the RS-232 ports on the 850 interface. If you don't want this program to automatically load when you boot with the master disk, rename the file to RS232.

3.3. SPECIAL DOS INFORMATION

When DOS is in memory, changes can be made to the DOS program. These changes can be made by poking the changes into memory. If you want to make the changes permanent, you can

3.3. SPECIAL DOS INFORMATION

type DOS [RETURN] to load the utilities. From the utilities menu you can use the write DOS files option to save the changes on disk. Some of the useful changes you can make follow.

POKE 1913,80 – This turns off the write verify and speeds up disk writing.

POKE 1913,87 – This turns write verify on

POKE 5903,42

POKE 5904,46

POKE 5905,82

POKE 5906,85

POKE 5907,78

POKE 5908,155 – This causes any binary file with the extender .RUN to be loaded automatically when the computer is turned on.

POKE 5903,65

POKE 5904,85

POKE 5905,84

POKE 5906,79

POKE 5907,82

POKE 5908,85 – This returns the DOS to normal, Automatically loading files named AUTORUN.SYS.

DOS 2.0

POKE 3772,255

POKE 3818,64

POKE 3822,123

DOS 2.5

POKE 3774,64

POKE 3778,123 – This will cause DOS to accept lower-case as well as upper-case letters in file names. It will also now accept @,[\,],^ and _ .

DOS 2.0

POKE 3772,223

POKE 3818,65

POKE 3822,91

DOS 2.5

POKE 3774,65

POKE 3778,91 – This will change DOS back to normal, accepting only upper-case letters and numbers.

3. USING THE DOS 2 UTILITIES (DUP.SYS)

4. THE CASSETTE HANDLER (C:)

The cassette handler sends data to the cassette recorder in blocks of 128 bytes each. The blocks are sent in the following format:

```
-----  
|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  | handled by SIO  
-----
```

Figure 4.1.: Cassette record format

The control byte may have one of the following values.

HEX	DEC	Description
\$FC	(252)	record is full
\$FA	(250)	partly full, next record is EOF
\$FE	(254)	EOF record, data section is all zeroes

Table 4.1.: Cassette handler control byte

The cassette handler has two modes of operation. The first mode uses only a short gap between records. It is called the no IRG (interrecord gaps) mode. The second mode uses longer gaps between records and is called the IRG mode. In the IRG mode the computer may stop the cassette recorder between records for processing data. When a channel is opened to the cassette recorder, bit 7 of ICAUX2 may be set to 1 (ICAX2 = \$80 (128)). This will cause the cassette to use the no IRG mode.

4. THE CASSETTE HANDLER (C:)

A cassette file starts with a 20 second mark tone. This tone is followed by the file records with 128 data bytes each. The final record is an End-Of-File record. The cassette is a straight-forward read/write device. There are no special functions other than those common to other CIO devices. The cassette motor is controlled by one of the controller port control registers. If bit 3 of PACTL [\$D302 (54018)] is 0 then the cassette motor is on. The following BASIC commands will turn the cassette motor on and off.

Basic Command	Description
POKE 54018,PEEK(54018)-8	motor on
POKE 54018,PEEK(54018)+8	motor off

Table 4.2.: Cassette motor control

Useful data base variables and OS equates

PACTL \$D302 (54018): port A control register, bit 4 controls cassette motor

5. THE KEYBOARD HANDLER (K:)

The keyboard is a read only device and therefore the keyboard handler has no output functions. The keyboard handler reads the keys as ATASCII codes. Each key is represented by one byte of data. Therefore, each time a key is pressed the data is treated as a byte of data just as data from any other device is. The only difference is that the computer must wait for the operator to press the keys as it reads the data.

Whenever a key is pressed an IRQ interrupt is generated by the keyboard reading hardware. The internal code (not ATASCII) for the key just pressed is then stored in CH [\$02FC (764)]. The code is then compared with the prior key code in CH1 [\$02F2 (754)]. If the code in CH1 is different from the code in CH, the key is accepted. The code is then converted to ATASCII, and placed in the database variable ATACHR [\$02FB (763)]. On XL and XE models, KEYDEF [\$0079,2 (121)] points to the key-code-to-ATASCII conversion table. (This address is used by the the screen handler in 400/800 models).

If the code in CH1 is the same as the code in CH, the new key code will not be accepted unless the key debounce timer, KEYDEL [\$02F1 (753)] is 0. When CIO is told to do an input operation from the keyboard, CH is checked to see if a key has been pressed. If CIO finds \$FF (255) in CH, it waits until a key is pressed. If CH is not \$FF, a key has been pressed and the ATASCII code for that key is taken from ATACHR. CH is then set to \$FF.

The data in CH is in the following format.

```

  7 6 5 4 3 2 1 0
-----
|C|S| key code  |
-----
C 1 = [CTRL] key is pressed
S 1 = [SHIFT] key is pressed
```

Figure 5.1.: Key code format

Anytime a key is pressed, CH is loaded with the key code. CH will hold the code until the computer is commanded to read the keyboard. Sometimes the computer will read a key which was pressed long ago. If you want to prevent this, load CH with \$FF before reading the keyboard. (In BASIC use POKE 764,255.) This will clear out any old key pressings.

Special function keys

[CTRL][1] screen output start/stop

5. THE KEYBOARD HANDLER (K:)

[CTRL][2] BELL

[CTRL][3] Generates End-Of-File status

[/\] or [/] inverse video toggle

[CAPS LOWER] sets lower case

[CTRL][CAPS] sets CTRL lock

[SHIFT][CAPS] sets caps lock

5.1. KEYBOARD REPEAT DELAY AND RATE CONTROL

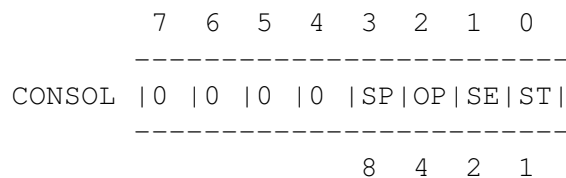
On the XL and XE, KRPDEL [\$02D9 (729)] determines the delay before the key repeat begins. The value of this byte is the number of vertical blanks (1/60th second each) to delay. KEYREP [\$02DA (730)] determines the repeat rate in vertical blanks.

5.2. KEYBOARD CLICK

The keyboard click of the XL/XE is heard through the TV speaker. The click may be turned off by putting \$FF in NOCLIK [\$02DB (731)].

5.3. NON-HANDLER, NON-CIO KEYS

The [OPTION], [SELECT] and [START] keys are read from the console switch register, CONSOL [\$D01F (53279)].



ST 0 = [START]

SE 0 = [SELECT]

OP 0 = [OPTION]

SP Console speaker. set to 1 during vertical blank. toggling this bi

Figure 5.2.: The console switch register

The [HELP] key on XL and XE models is read from HELPPFG, [\$02DC (732)]. This address is latched and must be reset to zero after being read.

5.3. NON-HANDLER, NON-CIO KEYS

```

      7 6 5 4 3 2 1 0
      -----
HELPFG | C S 0 H 0 0 0 H |
      -----
      1 6 3 1 8 4 2 1
      2 4 2 6
      8
H 1 = [HELP] (bits 0 and 4)
S 1 = [SHIFT]
C 1 = [CONTROL]

```

Table 5.1.: The [HELP] key register

Label	Hex	Dec	Description
KEYDEF	\$0079,2	121	key code conversion table vector (XL/XE)
KRPDEL	\$02D9	729	delay before key repeat (XL/XE)
KEYREP	\$02DA	730	key repeat rate (XL/XE)
NOCLIK	\$02DB	731	\$FF turns off key click (XL/XE)
HELPFG	\$02DC	732	[HELP] key (XL/XE)
ATACHR	\$02FB	763	ATASCII Code for last key
CH	\$02FC	764	keycode, \$FF if no key has been pressed
BRKKEY	\$0011	17	break key flag, 0 = break key pressed
SRTIMR	\$022B	555	Key delay and repeat timer
SHFLOK	\$02BE	702	SHIFT/CTRL lock flag \$00 = lower case \$40 (64) = upper case lock \$80 (128) = CTRL lock
INVFLG	\$02B6	694	inverse video flag, non-zero = inverse
CONSOL	\$D01F	53279	start, select and option keys
IRQEN	\$D20E	53774	IRQ interrupt enable bit 7 enables [BREAK] bit 6 enables other keys
shadow registers			
POKMSK	\$0010	16	IRQEN shadow

Table 5.3.: Useful database variables and OS equates

5. *THE KEYBOARD HANDLER (K:)*

6. THE PRINTER HANDLER (P:)

The printer is a write only device so the printer handler has no input functions. The printer handler has no special functions other than the CIO functions common to all other devices. Although many printers have special functions, the printer handler has no control over them. See your printer manual for information on special functions.

6. *THE PRINTER HANDLER (P:)*

7. SCREEN EDITOR (E:)

The screen editor uses both the keyboard handler and the screen handler to provide interactive control of the computer. In fact, the keyboard handler, the screen handler and the screen editor are contained in a single section of code and are therefore very closely related.

The editor works with one line of characters at a time. The lines it works with are called logical lines and are up to three screen lines long. The screen editor inputs data from the keyboard and then prints the data on the screen. When the [RETURN] key is pressed, the editor inputs all of the data on the present logical line for processing by CIO.

If characters are typed on the screen, and then the cursor is moved off the line, then back on the line, and new characters are typed, only the characters to the right of the reentry point of the cursor are input when [RETURN] is pressed. However, if the cursor is moved off the line again, then moved back on, all characters on that logical line are input.

If bit 0 of ICAX1 is 1, the editor will act as if the [RETURN] key is being held down. This bit may be changed at any time.

7.1. Editor control codes

The screen editor treats certain ATASCII codes as special control codes.

7. SCREEN EDITOR (E:)

Key	Hex	Dec	Function
[RETURN]	\$9B	155	carriage return or EOL
[CLEAR]	\$7D	125	Clear screen,put cursor in upper left
[UP ARROW]	\$1C	28	Move cursor up one screen line
[DOWN]	\$1D	29	down one line
[LEFT]	\$1E	30	left one character
[RIGHT]	\$1F	31	right one character
[BACK SPACE]	\$7E	126	Back-space operation
[SET TAB]	\$9F	159	sets tab stop at cursor
[CLEAR TAB]	\$9E	158	Clear tab stop at cursor
[TAB]	\$7F	127	move to next tab stop
[SHIFT][INSERT]	\$9D	157	Make space for a new line
[SHIFT][DELETE]	\$9C	156	delete the logical line at the cursor
[CTRL][INSERT]	\$FF	255	make room for a character
[CTRL][DELETE]	\$FE	254	delete character at cursor
[ESCAPE]	\$1B	27	causes next non-EOL code to be displayed as an ATASCII character, even if it is an editor control code
[CTRL][1]			screen print start/stop
	\$FD		

Table 7.1.: Screen editor control codes

8. THE DISPLAY HANDLER (S:)

The display handler manages the computer's video display. Although no data ever leaves the computer through it, the display is treated like any other CIO device. Data sent to the screen may be displayed as either characters or point by point graphics. Although it is only visible in the 40 column text mode, mode 0, there is a cursor on the screen in all of the text or graphics modes. Whenever a character or graphics point is put on the screen, the cursor moves just as in mode 0.

The display is capable of both input and output. Information can be put on the screen with any of the CIO output commands. An input command will find whatever is on the screen at the position of the cursor.

When text or graphics is sent to the screen it is actually stored in an area of memory called the display buffer. What you see on the screen is the computer's interpretation of the data stored there. This will be explained further as each mode is covered.

DISPLAY HANDLER SPECIAL FUNCTIONS DRAW

FILL

SPECIAL ERROR STATUSES: \$84 (132) Invalid special command.

\$8D (141) Cursor out of range.

\$91 (145) Nonexistent screen mode.

\$93 (147) Insufficient ram for screen mode.

8.1. TEXT MODE 0

In graphics mode 0, data passes through CIO, and is stored in the display buffer in the following format.

```
  7 6 5 4 3 2 1 0
-----
|I|   Data   |
-----
I 1 = displays character in inverse video
```

Figure 8.1.: Text Mode 0 Display Buffer Format

8. THE DISPLAY HANDLER (S:)

Bits 0 through 6 select one of the 128 characters in the ATASCII set. If bit seven = 1, the character is displayed in inverse video. Converting the above byte to decimal will give the BASIC ASC(x) equivalent.

The characters displayed in the text modes are determined by the ATASCII character set. This is a bit by bit representation of how the characters appear on the screen. The character set starts \$E000 (57344) in the operating system ROM. From there, for 1K of memory, each eight bytes holds a "bit map" of a particular character. See Figure 8.2 how the letter A is stored in the character set.

	7	6	5	4	3	2	1	0		
\$E208		0	0	0	0	0	0	0		
		0	0	0	1	1	0	0		* *
		0	0	1	1	1	1	0		* * * *
		0	1	1	0	0	1	1		* * * *
		0	1	1	0	0	1	1		* * * *
		0	1	1	1	1	1	1		* * * * * *
		0	1	1	0	0	1	1		* * * *
\$E20F		0	0	0	0	0	0	0		

Figure 8.2.: Letter A as represented in the C-set

XL and XE models have an international character set starting at \$CC00 (55224). In this character set the graphics characters are replaced by international characters. Custom characters sets may be loaded at any free address which is a multiple of 1,024 (\$0400, or 1K). The database variable CHBAS [\$02F4 (756)] stores the most significant byte (MSB) of the address of the active C-set. Since the LSB of the C-set address is always \$00, no LSB is needed to find it.

The data stored in the display buffer does not use the ATASCII code. A special code needed by the ANTIC chip is used.

ATASCII	display
\$00 - \$1F (0 - 31)	\$40 - \$5F (64 - 95)
\$20 - \$5F (32 - 95)	\$00 - \$3F (0 - 63)
\$60 - \$7F (96 - 127)	unchanged

Table 8.1.: DISPLAY CODE / ATASCII CODE CONVERSION

The codes for inverse video (the above codes with bit 7 set (= 1) or the above codes + 128 in decimal) are treated likewise.

When you first turn on the computer, BASIC opens channel 0 to the screen editor (E:). The screen editor uses both the keyboard handler and the screen handler, in mode 0, to display characters when they are typed in.

8.2. TEXT MODES 1 AND 2

Graphics modes 1 and 2 offer a split screen configuration if desired. The split screen has four lines of mode 0 at the bottom of the screen. In mode 1 the screen holds 20 characters horizontally and 24 characters vertically. In mode 2 the characters are twice as tall so the screen holds 12 vertically.

In BASIC, characters are sent to the screen with the PRINT command. Since BASIC uses channel 6 for graphics you must specify channel 6 in the command. For example:

```
? #6;"HELLO"
```

If you use a comma in place of the semicolon, ten spaces will print before the "HELLO". You can also use the PLOT and DRAWTO commands. In this case the COLOR command determines the character, as well as the color to be displayed.

Data passes through CIO in the following form:

```
  7 6 5 4 3 2 1 0
-----
|  C  |   D   |
-----
```

C determines the color (Figure 8.2)

D is a 5 bit ATASCII code which selects the character to be displayed.

8. THE DISPLAY HANDLER (S:)

C	Default Color	Color Register	Shadow Register
0	green	COLPF1	COLOR1
1	gold	COLPF0	COLOR0
2	gold	COLPF0	COLOR0
3	green	COLPF1	COLOR1
4	red	COLPF3	COLOR3
5	blue	COLPF2	COLOR2
6	blue	COLPF2	COLOR2
7	red	COLPF3	COLOR3

Table 8.2.: CIO Color

The database variable CHBAS selects between upper case (CHBAS=\$E0 (224)) and lower case (CHBAS=\$E2 (226)).

8.3. GRAPHICS MODES 3 THROUGH 11

Modes 3 through 8 offer a split screen mode. In modes 9 through 11 special programming is required for split screens. These modes use dot by dot (pixel by pixel) graphics instead of character sets. Before explaining how graphics are sent to the screen through CIO, I will describe how the data in the display buffer is interpreted by the ANTIC chip.

Mode 8 is the simplest of the graphics modes. Each byte of the display buffer controls eight pixels horizontally. The first 40 bytes of the display buffer control the first horizontal line of graphics. This makes a total of 320 pixels horizontally. If one of the eight bits of a byte is a 1 then the pixel it controls is on. If a bit is a 0 then it's pixel is off. For example, if a particular byte is equal to \$9B (binary 10011011) then its' part of the screen would look like...

```
*   **  **  
(10011011)
```

In reality the pixels are assigned to different color registers. A color register is a byte of memory which controls the color of all pixels assigned to it. In mode 8, if a bit is = 0 it's pixel is assigned to the register called COLBK. If a bit is one, it's pixel is assigned to COLPF0. See COLORS below for more information on the color registers.

You may notice a close similarity between mode 0 and mode 8. The major difference between these modes is where the dot by dot information comes from. In mode 8 this information comes from the display buffer. In mode 0 the display buffer contains codes telling what characters to display. The actual dot by dot information comes for the character set at \$E000.

In mode 7 each pixel is controlled by two bits. Therefore each byte only controls four pixels. There are also only 1/4 as many pixels on the screen as in mode 8. See mode 3 below for an

8.3. GRAPHICS MODES 3 THROUGH 11

explanation of how the each byte affects the pixels. In a graphics mode, when CIO sends a byte of data to the screen handler, that byte has information for only one pixel. Do not confuse a byte which CIO sends to the screen handler with the bytes in the display buffer.

CIO sends data to or retrieves data from the screen in the following forms.

```

 7 6 5 4 3 2 1 0
-----
|0 0 0 0 0 0| D | Modes 3,5,7 -- D = color
-----
|0 0 0 0 0 0 0|D| Modes 4,6,8 -- D = Color
-----
|0 0 0 0|   D   | Modes 9,10,11 -- D = data
-----

```

Mode 3 uses a screen which is 40 pixels horizontally and 24 vertically. Each pixel is a square the size of a mode 0 character. It requires 273 bytes of RAM where each byte controls 4 pixels. Each pair of bits controls which of the four color registers their pixel is assigned to.

```

 7 6 5 4 3 2 1 0
-----
| D | D | D | D |
-----
 P1  P2  P3  P4
Pixel/color register assignments:
D = 00 COLBK  (COLOR4)
    01 COLPF0 (COLOR0)
    10 COLPF1 (COLOR1)
    11 COLPF2 (COLOR2)

```

Figure 8.3.: display buffer byte for mode 3

Mode 4 uses a screen of 80 columns by 48 rows. Each pixel is half the size of those in mode 3. Mode 4 requires 537 bytes of RAM where each byte controls 8 pixels. This mode is very similar to mode 8 except there are fewer but larger pixels. Mode 5 uses a screen of 80 columns by 48 rows. The pixels are the same size as in mode 4. Mode 5 requires 1,017 bytes of RAM where each byte controls 4 pixels in the same manner as in mode 3. Mode 6 uses a screen of 160 columns by 96 rows. It requires 2,025 bytes of RAM where each byte controls 8 pixels as in mode 4. Mode 7 uses a screen of 160 columns by 96 rows. It requires 3,945 bytes of RAM where each byte controls 4 pixels as in modes 3 and 5.

8. THE DISPLAY HANDLER (S:)

Modes 8 through 11 (and 15 on XL and XE models) each require 7,900 bytes of RAM and are very similar in display set up. The main differences between these modes is the interpretation of data in the display buffer. Mode 15 (sometimes called mode 7.5) uses a screen of 160 columns by 192 rows. Each byte controls 4 pixels as in mode 7. The main difference between mode 15 and its related modes is bit 0 of each instruction byte in the display list (the program which the ANTIC chip uses). If this bit is 0 the screen is interpreted as mode 15. If the bit is 1 the screen is interpreted as modes 8 through 11. Modes 8 through 11 are set up identically in memory, including the display list. The only difference is the data in the PRIOR register of the GTIA chip. The shadow register for PRIOR is GPRIOR [\$026F (623)].

Mode 8 (PRIOR = \$00 - \$3F (0 - 63)), uses a screen of 320 columns by 192 rows. Each byte controls 8 pixels as in modes 4 and 6.

Mode 9 (PRIOR = \$40 - \$7F (64 - 127)) uses a screen of 80 columns by 192 rows. Each byte controls 2 pixels. The pixels are all of the same color, controlled by COLBK. Each half of a byte in the display buffer controls the luminance of the assigned pixel. The format of each byte is as follows.

```

    7 6 5 4 3 2 1 0
    -----
    | data | data |
    -----
    pixel 1 | pixel 2
```

Mode 10 (PRIOR = \$80 - \$BF (128 - 191), is the same as mode 9 except 9 color luminance combinations are available. The data in each half byte chooses one of the 9 color registers for the assigned pixel. Mode 11 (PRIOR = \$C0 - FF (192 - 255), is the same as mode 9 except there is one brightness but 16 colors. The pixel data chooses one of the 16 available colors. The luminance is that of the background (COLBK).

8.4. USING THE SCREEN HANDLER

8.4.1. OPENING A CHANNEL TO THE SCREEN HANDLER

When a channel is opened to the screen handler the following actions take place:

The area of memory to be used for the screen data is cleared. A display list (program for the ANTIC chip) is set up for the proper graphics mode. The top-of-free-memory pointer, MEMTOP [\$02E5,2 (741)], is set to point to the last free byte before the display list. Before opening a channel to the screen handler, the pointer to the highest memory address needed by the program, APPMHI [\$000E,2 (14)], should be properly set. This will prevent the screen handler from erasing part of the program when it sets-up the screen data region. When the channel is opened, two special options can be sent with the direction parameter (ICAX1).

Before the open command, the graphics mode number is placed into ICAX2.


```

 7 6 5 4 3 2 1 0
-----
|   C S W R   |
-----
 1 6 3 1 8 4 2 1
 2 4 2 6
 8
C 1 = don't clear the screen
S 1 = split screen
R 1 = input
W 1 = output

```

Figure 8.4.: ICAX1 for screen open

ICAX2 for screen open

```

 7 6 5 4 3 2 1 0
-----
|           : mode |
-----
mode = $00 through $0B (0 - 11 (0 - 15 on XL/XE))

```

To open a channel to the screen in BASIC use the GRAPHICS command. BASIC screen open format

```
GRAPHICS mode
```

For Example:

```
GRAPHICS 8
```

This will set up a mode 8 graphics screen and open channel 6 to it. If the graphics mode is 1 - 8, a split screen will be set up. For example, GRAPHICS 8 will set up a mode 8 screen with a four line text window at the bottom.

If 16 is added to the mode number, a full screen will be set-up. For example, GRAPHICS 8+16 or GRAPHICS 24 will set up a mode 8 screen, with no text window, a full 192 pixels high. If the number 32 is added to the mode number, the screen will not clear when the channel opens.

If you want to use a channel other than #6, you will have to use the open command. It is used in the following format. screen open without GRAPHICS command

```
OPEN #channel,direction/special,mode,"S:"
```

8. THE DISPLAY HANDLER (S:)

For example:

```
OPEN #1,8,7,"S:"
```

This will open channel 1 to a mode 7 screen for output only. For use of special parameters, see ICAX1 above.

8.4.2. USING AN OPEN CHANNEL TO THE SCREEN

Once a channel is opened to the screen it is used like any other input or output device. In other words, data is placed on the screen by the PRINT and PUT commands. Data is retrieved from the screen with the INPUT and GET commands. The part of the screen which the data will be put in or taken from is determined by the X,Y coordinants in the database variables COLCRS [\$0055,2 (85)] and ROWCRS [\$0054 (84)].

What appears on the screen depends on what graphics mode the computer is in. Before sending data to the screen in BASIC, a color register must be assigned to the data. Once a point is plotted on the screen, it's color will be determined by the color register it was assigned to. To assign a color to a plotted point, the COLOR command us used as follows.

COLOR command format

```
COLOR register
```

For example,

```
COLOR 1
```

After using the above command, all points plotted will be controlled by color register 1. To change color registers, use the COLOR command again.

In assembly language, the color is determined by the data sent to the screen. See the above section on graphics modes for color information. In BASIC the PLOT command is used to put data on the screen. The PLOT command is used as follows.

The BASIC PLOT command

```
PLOT x,y
```

x and y are the horizontal and vertical coordinates for the plotted point.

In modes 3 through 11 a single point will be plotted. In modes 1 and 2 a text character will be printed on the screen by the PLOT command. The PRINT and PUT commands can also be used in basic. What appears on the screen depends on the graphics mode. In modes 1 and 2

8.5. THE COLOR REGISTERS

the ATASCII characters sent to the screen will be printed just as in mode 0. See the paragraph on modes 1 and 2 above for more information. In the other modes what appears depends on how the ANTIC chip interprets the data bytes sent to the screen. For example, in mode 8, even numbered characters will be single pixels in color 1. Odd numbered characters will be in color 0 (background).

There are two special commands for the screen handler, DRAW and FILL

DRAW (ICCOM = \$11 (17)) The draw command works exactly like the plot command except a straight line is drawn from the previous pixel to the new one. In BASIC it is used in the following format.

the BASIC DRAW command

```
DRAWTO x,y
```

FILL (ICCOM = \$12 (18)) Fill works like draw except the area to the right of the drawn line will be filled with the color in FILDAT [\$02FD (765)]. The fill command expects to find a boundary to the right. If no boundary is found, the entire horizontal screen between the ends of the line is filled. To use the fill command in BASIC the XIO command must be used in the following format.

```
POSITION x,y  
XIO 18 #6,0,0,"E:"
```

Note that the cursor is first moved by the POSITION command. Below is an example of how to prepare for and use the fill command.

using the fill command

```
2nd DRAWTO   ._____. DRAWTO here  
              |       |  
              |       |  
              |       |  
fill to here !   ! PLOT here
```

This will draw and fill a box on the screen.

8.5. THE COLOR REGISTERS

There are nine bytes of memory which control the colors on the screen. These bytes are called color registers. The color registers have the following names and relationships.

8. THE DISPLAY HANDLER (S:)

Color registers and relationships

Register name	Register address		modes					
			0 & 8	1 & 2	3 5 7	4 & 6	9 & 11	10
	HEX	DEC	COLOR numbers					
PCOLR0	\$02C0	704						0
PCOLR1	\$02C1	705						1
PCOLR2	\$02C2	706						2
PCOLR3	\$02C3	707						3
COLOR0	\$02C4	708		0 - 63	1	1		4
COLOR1	\$02C5	709	1 - 255	64 -127	2			5
COLOR2	\$02C6	710	0	128-191	3			6
COLOR3	\$02C7	711		192-255				7
COLOR4	\$02C8	712	border	backgnd	0	backgnd	backgnd	8

The color numbers are in decimal. These are actually shadow registers. See the O.S. equates below for relationships. In modes 0 - 3 the COLOR number actually determines the character printed. The register to which a pixel/character is assigned to is determined by the data byte sent to the screen through CIO. The data in the color registers is in the following format.

```

  7 6 5 4 3 2 1 0
  -----
  | color |bright |
  -----
  color = one of 16 possible colors
  bright = one of 8 possible brightnesses
  (even numbers, 0 - E)

```

Figure 8.5.: Color register data format

In basic, the COLOR command is used to assign color registers. The corresponding registers depends on the graphics mode. For example, COLOR 0 is COLOR2 in mode 8. In most other modes COLOR 0 is COLOR4. See the above chart for the register relationships. To change the contents of the color registers in BASIC, the SETCOLOR command is used. In all modes except mode 10, the SETCOLOR command refers to the registers COLOR0 to COLOR4.

SETCOLOR 0	COLPF0	(COLOR0)
SETCOLOR 1	COLPF1	(COLOR1)
SETCOLOR 2	COLPF2	(COLOR2)
SETCOLOR 3	COLPF3	(COLOR3)
SETCOLOR 4	COLBK	(COLOR4)

Table 8.3.: SETCOLOR/register relationships

8.5. THE COLOR REGISTERS

The format for the SETCOLOR command is...

SETCOLOR command format

```
SETCOLOR register,hue,brightness
register = 0 - 4 (0 - 8 in mode 10)
hue = 0 - 15 (16 colors)
brightness = 0 - 16 (even numbers only (8 brightnesses))
```

The following chart gives the colors represented by the hue number.

hue	color	hue	color
0	grey	8	blue
1	gold	9	cyan
2	gold-orange	10	blue-green
3	red-orange	11	blue-green
4	orange	12	green
5	magenta	13	yellow-green
6	purple-blue	14	yellow
7	blue	15	yellow-red

Table 8.4.: colors represented by hue numbers

8.5.1. The attract mode

If a key is not pressed for more than 9 minutes the computer will enter the attract mode. This mode is used to prevent burning of the TV phosphors by lowering the brightness and constantly changing the colors. The attract mode timer, ATTRACT [\$004D (77)], is set to 254 (\$FE) when the the attract mode is entered. To force the computer out of the attract mode, poke a number less than 127 into ATTRACT.

Useful database variables and OS equates

8. THE DISPLAY HANDLER (S:)

Label	Hex	Dec	Description
APPMHI	\$000E,2	14	lower limit for screen region
ATTRACT	\$004D	77	attract mode timer and flag
LMARGN	\$0052	82	left margin
RMARGN	\$0053	83	right margin
ROWCRS	\$0054	84	horizontal cursor position
COLCRS	\$0055,2	85	vertical cursor position
DINDEX	\$0057	87	current graphics mode
SAVMSC	\$0058,2	88	starting address of display buffer
OLDROW	\$005A	90	previous cursor position (row)
OLDCOL	\$005B,2	91	previous cursor position (column)
OLDCHR	\$005D	93	character currently at the text cursor
OLDADR	\$005E,2	94	memory address of cursor
RAMTOP	\$006A	106	end-of-RAM + 1 (MSB only)
SDLSTL	\$0230,2	560	shadow register of display list address
TXTROW	\$0290	656	text window cursor position (row)
TXTCOL	\$0291,2	657	text window cursor position (column)
TXTMSC	\$0294,2	660	starting address of text window data buffer
RAMSIZ	\$02E4	740	permanent end-of-RAM + 1 (MSB only)
CRSINH	\$02F0	752	cursor inhibit, 1 = no cursor
FILDAT	\$02FD	765	color data for fill
DSPFLG	\$02FE	766	if >0 screen control codes are displayed as ATASCII characters (EOL is unaffected)
SSFLAG	\$02FF	767	> 0 = stop screen print
actual color registers loaded from shadow registers during vertical blank			
COLPM0	\$D012	53266	Color Player / Missile 0
COLPM1	\$D013	53267	Color Player / Missile 1
COLPM2	\$D014	53268	Color Player / Missile 2
COLPM3	\$D015	53269	Color Player / Missile 3
COLPF0	\$D016	53270	Color Playfield 0
COLPF1	\$D017	53271	Color Playfield 1
COLPF2	\$D018	53272	Color Playfield 2
COLPF3	\$D019	53273	Color Playfield 3
COLBK	\$D020	53274	Color Background
OS shadow registers			
PCOLR0	\$02C0	704	Shadow for COLPM0
PCOLR1	\$02C1	705	Shadow for COLPM1
PCOLR2	\$02C2	706	Shadow for COLPM2
PCOLR3	\$02C3	707	Shadow for COLPM3
COLOR0	\$02C4	708	Shadow for COLPF0
COLOR1	\$02C5	709	Shadow for COLPF1
COLOR2	\$02C6	710	Shadow for COLPF2
COLOR3	\$02C7	711	Shadow for COLPF3
COLOR4	\$02C8	712	Shadow for COLBK

9. THE RESIDENT DISK HANDLER

The resident disk handler is separate from DOS and is part of the permanent operating system ROM. The disk handler does not use CIO. The resident disk handler works with one sector at a time. It is used by setting the drive number, sector number, and operation code in the device control block. The program then jumps (JSR) to the handler entry vector, DSKINV [\$E453 (58451)].

Label	Hex	Dec	Description
DDEVIC	\$0300	768	Serial bus I.D. Set by handler
DUNIT	\$0301	769	Drive/Device number
DCOMND	\$0302	770	Command
DSTATS	\$0303	771	status byte
DBUF	\$0304,2	772	Pointer to 128 byte memory block for data storage
DTIMLO	0306	774	Timeout value (response time limit) in seconds
DBYT	\$0308,2	777	number of bytes transferred, set by handler
DAUX1	\$030A	778	sector number (lowbyte)
DAUX1	\$030B	779	sector number (highbyte)

Table 9.1.: Device control block (for resident disk handler)

9.1. DISK HANDLER COMMANDS

9.1.1. GET SECTOR

Before the JSR to DSKINV is made the following parameters are set.

GET SECTOR parameters

```
DCOMND = $52 (82)
DUNIT = (1 - 4)
DBUFHI and DBUFLO = address of 128 byte buffer
DAUX1 and DAUX2 = Sector number (LSB,MSB)
```

This operation will read the specified sector and put the data into the specified buffer.

9. THE RESIDENT DISK HANDLER

9.1.2. PUT SECTOR

PUT SECTOR is used the same as GET SECTOR except for DCOMND.

PUT SECTOR parameters

```
DCOMND = $50 (80)
DUNIT = (1 - 4)
DBUFHI and DBUFLO = address of 128 byte buffer
DAUX1 and DAUX2 = Sector number (LSB,MSB)
```

This operation sends the data in the specified buffer to the specified disk sector.

9.1.3. PUT SECTOR WITH VERIFY

PUT SECTOR WITH VERIFY is used the same as PUT SECTOR except for DCOMND.

PUT SECTOR WITH VERIFY parameters

```
DCOMND = $57 (87)
DUNIT = (1 - 4)
DBUFHI and DBUFLO = address of 128 byte buffer
DAUX1 and DAUX2 = Sector number (LSB,MSB)
```

This operation sends the data in the specified buffer to the specified disk sector then checks for errors.

9.1.4. GET STATUS

Only the DUNIT and DCOMND need to be set

GET STATUS parameters

```
DCOMND = $53 (83)
DUNIT = (1 - 4)
```

The status information will be put in three bytes starting at DVSTAT [\$02EA (746)].

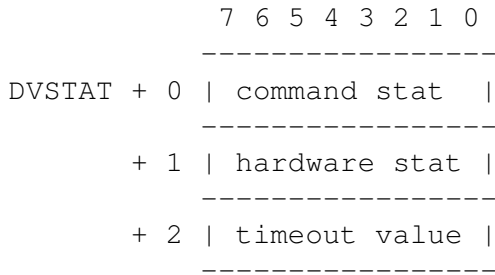


Figure 9.1.: DVSTAT Status format

The command status byte gives the following information.

```

Bit
0 1 = invalid command frame received
1 1 = invalid data frame received
2 1 = unsuccessful PUT operation
3 1 = disk is write protected
4 1 = active/standby

```

The hardware status byte contains the status register of the ISN1771-1 disk controller chip. The timeout byte contains the maximum allowable response time for the drive in seconds.

9.1.5. FORMAT DISK

The handler will format then verify the the disk. The numbers of all bad sectors (up to 63) will be put into the specified buffer followed by two bytes of \$FF. The following parameters are set before the call.

FORMAT parameters

```

DCOMND = $21 (33)
DUNIT = (1 - 4)
DBUFLO and DBUFHI = address of bad sector list (buffer)

```

After the operation the status byte is set. Also, DBYTLO and DBYTHI will contain the number of bytes of bad sector information (not including the two \$FF bytes).

9. THE RESIDENT DISK HANDLER

Label	Hex	Dec	Description
DVSTAT	\$02EA,3	746	device status block, 3 bytes
DDEVIC	\$0300	768	serial bus I.D.
DUNIT	\$0301	769	device number
DCOMND	\$0302	770	command byte
DSTATS	\$0303	771	status byte
DBUFLO	\$0304	772	data buffer pointer
DBUFHI	\$0305	773	data buffer pointer
DTIMLO	\$0306	774	timeout value
DBYTLO	\$0308	776	number of bytes transfered
DBYTHI	\$0309	777	number of bytes transfered
DAUX1	\$030A	778	sector number
DAUX2	\$030B	779	sector number
DSKINV	\$E453	58451	disk handler entry vector

10. SYSTEM INTERRUPTS

There are four types of interrupts which can occur with the 6502 microprocessor:

1. chip reset
2. IRQ, interrupt request (maskable)
3. MNI (non-maskable interrupt)
4. software interrupt (BRK instruction)

Figure 10.1.: 6502 interrupts

10.1. CHIP RESET

On the 400/800 the chip reset occurs only upon power-up and causes the computer to do a cold start. On later models, pressing [SYSTEM RESET] will cause a chip reset but the computer then does a warm start. On the 400/800, the [SYSTEM RESET] key generates a NMI interrupt.

10.1.1. COLD START

This is a synopsis of the cold start routine.

1. The warm start flag [\$0008] is set to 0 (false)
2. If a cartridge slot contains a diagnostic cartridge, control is handed to the cartridge.
3. The end of RAM is determined by trying to complement the first byte of each 4K block of memory.
4. Hardware registers at \$D000 - \$D4FF (except \$D100 - \$D1FF) are cleared.
5. RAM is cleared from \$0008 to the top of ram.
6. The user program jump vector, DOSVEC [\$000A] is set to point to the black board mode (Atari logo display mode / Self-Test in XL/XE models).
7. The screen margins are set to 2 and 39
8. Interrupt vectors are initialized.

10. SYSTEM INTERRUPTS

9. Bottom of free RAM pointer, MEMLO [\$02E7], is set to point to \$0700.
10. Resident CIO handlers are initialized.
11. If the [START] key is pressed the cassette boot request flag, CKEY [\$004A], is set.
12. The CIO device table is initialized.
13. If a cartridge is present it is initialized.
14. Channel 0 is opened to the screen editor. The top-of-free-RAM pointer, MEMTOP [\$02E5], is set to point below the screen region. The computer then waits for the screen to be established before continuing.
15. If the cassette boot flag is set the cassette is booted.
16. If there is no cartridge present or a cartridge doesn't prevent it, the disk is booted.
17. The cold start flag is reset.
18. If there is a cartridge present, the computer jumps to the cartridge's run vector.
19. If there is no cartridge present the computer jumps through the vector DOSVEC [\$000A (10)]. DOSVEC will point to either a booted program, the memo pad routine (400/800) or the logo display routine / selftest (XL/XE).

10.1.2. WARM START

1. The warm start flag is set to \$7F (true).
2. cold start steps 2 - 4 are executed.
3. RAM is cleared from \$0010 - \$007F and \$0200 - \$03FF.
4. Cold start steps 7 - 14 are executed.
5. If cassette booted software is present the computer JSRs through CASINI [\$0002].
6. If disk booted software is present the computer JSRs through DOSINI [\$000C (12)].

The difference between cold start and warm start is the condition of the warm start flag, WARMST, [\$0008]. If this flag is 0 a complete cold start is executed. If the flag is anything other than 0 then only the warm start part of the warm start/cold start code is executed.

10.2. NON-MASKABLE INTERRUPTS (NMI)

NMI interrupts are generated by the following conditions:

1. Display list interrupt, generated by the ANTIC chip.
2. TV vertical blank interrupt, generated by the ANTIC chip.
3. [SYSTEM RESET] key (400/800).

When an NMI interrupt occurs, the hardware register NMIST [\$D40F] is examined to determine what type of interrupt occurred. The computer is then directed through the proper ram vector to service the interrupt.

10.3. DISPLAY LIST INTERRUPTS (DLIs)

The computer makes no use of DLIs. The ram vector points to an RTI instruction.

10.4. VERTICAL BLANK INTERRUPTS (VBIs)

There are two stages to the VBI service routine. The second stage is only done if a critical function was not interrupted.

Stage 1 (immediate VBI)

1. The real time clock, RTCLOK [\$0012 - \$0014], is incremented.
2. The attract mode variables are processed.
3. System timer 1 is decremented. If it goes to zero the computer JSRs through system time-out vector 1.

Stage 2 (deferred VBI)

1. The hardware registers are loaded with the data in their shadow registers.
2. System timer 2 is decremented. If it goes to zero the computer JSRs through the system time-out vector 2.
3. System timers 3, 4, and 5 are decremented. If a timer goes to zero the computer sets system timer flags 3, 4, and/or 5.

10. SYSTEM INTERRUPTS

4. If auto-repeat is active, the auto-repeat process is done.
5. The keyboard debounce timer is decremented if not 0.
6. Information at the controller port registers is read, processed and placed in the proper shadow registers.

10.5. [SYSTEM RESET] INTERRUPT

If a [SYSTEM RESET] interrupt is generated on the 400/800 the computer jumps to the warm start routine.

10.6. INTERRUPT REQUESTS (maskable interrupts (IRQs))

When an IRQ interrupt occurs the hardware register IRQST [\$D20E], the PIA status registers, PACTL [\$D302] and PBCTL [\$D303] are examined to determine what caused the interrupt. For each interrupt, the 6502 accumulator is pushed to the stack. The computer is then directed to the proper ram vector to service the interrupt.

10.7. SOFTWARE INTERRUPT (BRK instruction)

The operating system doesn't use software interrupts. The software interrupt vector points to a PLA followed by an RTI.

10.8. SYSTEM TIMERS

The following timers are updated during vertical blank (VBI) as noted above. If a timer is decremented to 0 the computer jumps through it's associated vector or sets it's associated flag.

Label	Hex	Dec	flag/vector
RTCLOK	\$0012,3	18	3 byte clock (\$0012 = MSB)
CDTMV1	\$0218		CDTMA1 \$0226 vector (SIO time-out)
CDTMV2	\$021A		CDTMA2 \$0228 vector
CDTMV3	\$021C		CDTMF3 \$022A flag
CDTMV4	\$021E		CDTMF4 \$022C flag
CDTMV5	\$0220		CDTMF5 \$022E flag

Label	Hex	Dec	Type	default function
VDSLST	\$0200	512	NMI	DLI Points to an RTI
VVBLKI	\$0222	546	NMI	stage 1 VBI
VVBLKD	\$0224	548	NMI	return-from-interrupt routine (RTI)
CDTMA1	\$0226	550	NMI	time-out 1 (used by SIO)
CDTMA2	\$0228	552	NMI	time-out 2 (not used by OS)
VPRCED	\$0202	514	IRQ	not used (points to PLA,RTI)
VINTER	\$0204	516	IRQ	not used (PLA,RTI)
VKEYBD	\$0208	520	IRQ	keyboard interrupt
VSERIN	\$020A	522	IRQ	used by Serial I/O routine
VSEROR	\$020C	524	IRQ	used by SIO
VSEROC	\$020E	526	IRQ	used by SIO
VTIMR1	\$0210	528	IRQ	not used by OS (PLA,RTI)
VTIMR2	\$0212	530	IRQ	not used by OS (PLA,RTI)
VTIMR4	\$0214	532	IRQ	?
VIMIRQ	\$0216	534	IRQ	main IRQ code
VBREAK	\$0206	518	BRK	unused by OS (PLA,RTI)

Table 10.1.: Interrupt vectors

10.9. HARDWARE INTERRUPT CONTROL

There are two registers on the antic chip which control interrupts. These registers can be used to disable interrupts if necessary. There are also two associated interrupt status registers. The IRQ enable and status registers use the same address. The result is that reading the register does not reveal the enabled interrupts but the interrupts pending. IRQ interrupt enable data should usually be written to the OS shadow first. Reading the OS shadow tells which interrupts are enabled.

Non maskable interrupt enable

```

NMIEN $D40E
 7 6 5 4 3 2 1 0
-----
| | | not used |
-----
bit 7 1 = DLI enabled
6 1 = VBI enabled

```

Non maskable interrupt status

```

NMIST $D40F
 7 6 5 4 3 2 1 0
-----

```

10. SYSTEM INTERRUPTS

```
| | | | not used|
-----
bit 7 1 = DLI pending
6 1 = VBI pending
5 1 = [SYSTEM RESET] key pending
```

Interrupt request enable

```
IRQEN $D20E
7 6 5 4 3 2 1 0
-----
| | | | | | | |
-----
bit 7 1 = [BREAK] key interrupt enable
6 1 = keyboard interrupt enable
5 1 = serial input interrupt enable
4 1 = serial output interrupt enable
3 1 = serial output-finished interrupt enable
2 1 = timer 4 interrupt enable
1 1 = timer 2 interrupt enable
0 1 = timer 1 interrupt enable
```

IRQEN has a shadow register, POKMSK [\$0010 (A)].

Interrupt request status

```
IRQST $D20E
7 6 5 4 3 2 1 0
-----
| | | | | | | |
-----
bit 7 1 = [BREAK] key interrupt pending
6 1 = keyboard interrupt pending
5 1 = serial input interrupt pending
4 1 = serial output interrupt pending
3 1 = serial output-finished interrupt pending
2 1 = timer 4 interrupt pending
1 1 = timer 2 interrupt pending
0 1 = timer 1 interrupt pending
```

10.10. WAIT FOR HORIZONTAL SYNC

Writing any number to WSYNC [\$D40A (54282)] will cause the computer to stop and wait for the next TV horizontal sync. It is wise to use DLIs one TV line before needed then writing to

10.10. WAIT FOR HORIZONTAL SYNC

WSYNC. This will keep other interrupts from causing DLIs to be serviced late. This can cause a DLI to change something in the middle of a scan line.

Useful database variables and OS equates

POKMSK \$0010 (16): IRQEN shadow

IRQEN \$D20E (53774): enables IRQs when written to

IRQST \$D20E (53774); gives IRQs waiting when read

PACTL \$D302 (54018): bit 7 (read) peripheral A interrupt status

bit 0 (write) peripheral A interrupt enable

PBCTL \$D303 (54019): bit 7 (read) peripheral B interrupt status

bit 0 (write) peripheral B interrupt enable

WSYNC \$D40A (54282): wait for horizontal sync

NMIEN \$D40E (54286): NMI enable

NMIST \$D40F (54287): NMI status

10. SYSTEM INTERRUPTS

11. THE FLOATING POINT ARITHMETIC PACKAGE

The routines which do floating point arithmetic are a part of the operating system ROM. The Atari computer uses the 6502's decimal math mode. This mode uses numbers represented in packed Binary Coded Decimal (BCD). This means that each byte of a floating point number holds two decimal digits. The actual method of representing a full number is complicated and probably not very important to a programmer. However, for those with the knowledge to use it, the format is given below.

```
byte 0 xx excess 64 exponent + sign
      xx \
      xx \
      xx > 10 BCD digits
      xx /
byte 7 xx /
```

Figure 11.1.: Floating point number representation

The decimal point is shifted to left of the MSD and the exponent is adjusted accordingly. Therefore, the decimal point doesn't need to be represented.

For programming purposes, floating point numbers can be in ASCII code. It takes up to 14 bytes to store a floating point number in this manner. The floating point package has a routine to convert numbers between ASCII and floating point.

11.1. USE OF THE FLOATING POINT PACKAGE

The floating point package has several routines to convert between ASCII and FP and to do the arithmetic functions. These are the important data base variables.

Floating point data base variables

FR0 \$00D4,6 (212): 6 byte buffer for floating point number

FR1 \$00E0,6 (224): 6 byte buffer for floating point number

CIX \$00F2 (242): index for INBUFF address

11. THE FLOATING POINT ARITHMETIC PACKAGE

INBUFF \$00F3,2 (243): 2 byte pointer to ASCII floating point number

FLPTR \$00FC,2 (252): 2 byte pointer to user buffer for floating point number

LBUFF \$0580,? (1408): result buffer for FASC routine

11.2. MAKING THE CALL

To do a floating point function, first set the proper pointers and JSR to the operation entry point. Below is a list of the entry points and parameters.

ASCII to floating point Converts ASCII representation pointed to by INBUFF to FP in FR0.

```
AFP      = $D800
INBUFF  = address of ASCII number
CIX     = buffer offset if any
JSR AFP
```

FLOATING POINT TO ASCII Converts floating Point number in FR0 to ASCII. The result will be in LBUFF. INBUFF will point to the ASCII number which will have the bit 7 of the last byte set to 1.

```
FASC = $D8E6
JSR FASC
```

INTEGER TO FLOATING POINT CONVERSION. Converts a 2 byte unsigned integer (0 to 65535) in FR0 to floating point in FR0.

```
IFP = $D9AA
JSR IFP
```

FLOATING POINT TO INTEGER CONVERSION. Converts floating point number in FR0 to 2 byte integer in FR0.

```
FPI = $D9D2
JSR FPI
BCS overflow
```

ADDITION Adds floating point numbers in FR0 and FR1 with result in FR0.

```
FADD = $DA66
JSR FADD
BCS out of range
```

SUBTRACTION subtracts FR1 from FR0 with the result in FR0.

```
FSUB = $DA60
JSR FSUB
BCS out of range
```

MULTIPLICATION Multiplies FR0 by FR1 with the result in FR0.

```
FMUL = $DADB
JSR FMUL
BCS out of range
```

DIVISION Divides FR0 by FR1 with result in FR0.

```
FDIV = $DB28
JSR FDIV
BCS out of range or divisor is 0
```

LOGARITHMS Puts logarithm of FR0 in FR0

```
LOG = $DECD
LOG10 = $DED1
JSR LOG ;for natural log.
```

or

```
JSR LOG10 ;for base 10 log.
BCS negative number or overflow
```

EXPONENTIATION Put exponentiation of FR0 in FR0

```
EXP = $DDC0
EXP10 = $DDCC
JSR EXP ;for e ** Z
```

or

```
JSR EXP10 ;for 10 ** Z
```

11. THE FLOATING POINT ARITHMETIC PACKAGE

POLYNOMIAL EVALUATION Puts the result of an n degree polynomial evaluation of FR0 in FR0.

```
PLYEVL = $DD40
LDX LSB of pointer to list of floating point coefficients, ordered hi
LDY MSB of above
LDA number of coefficients in list
JSR PLYEVL
BCS overflow
```

CLEAR FR0 Sets FR0 to all zeroes

```
ZFR0 = $DA44
JSR ZFR0
```

CLEAR ZERO PAGE FLOATING POINT NUMBER Clears user floating point number in page zero.

```
ZF1 = $DA46
LDX address of zero page FP buffer
JSR ZF1
```

LOAD FR0 WITH FLOATING POINT NUMBER Loads FR0 with user FP number in buffer pointed to by 6502 X and Y registers or by FLPTR. After either operation below, FLPTR will point to the user FP buffer.

```
FLD0R = $DD89
LDX lsb of pointer
LDY msb
JSR FLD0R
```

or

```
FLD0P = $DD8D
FLPTR = address of FP number
JSR FLD0P
```

LOAD FR1 WITH FLOATING POINT NUMBER Loads FR1 with user FP number in buffer pointed to by 6502 X and Y registers or by FLPTR. After either operation below, FLPTR will point to the user FP buffer.

```
FLD1R = $DD98
LDX lsb of pointer
LDY msb
JSR FLD1R
```

or

```
FLD1P = $DD9C
FLPTR = address of FP number
JSR FLD1P
```

STORE FR0 IN USER BUFFER stores the contents of FR0 in user FP buffer pointed to by 6502 X and Y registers or by FLPTR. After either operation below, FLPTR will point to the user FP buffer.

```
FST0R = $DDA7
LDX lsb of pointer
LDY msb
JSR FST0R
```

or

```
FST0P = $DDAB
FLPTR = address of FP number
JSR FST0P
```

MOVE FR0 TO FR1 Moves the contents of FR0 to FR1

```
FMOVE = $DDB6
JSR FMOVE
```

The usual use sequence of the floating point package might be to: load FR0 and FR1 with FP numbers from user specified buffers do the math then store FR0 in a user buffer. An alternative might be to: convert an ASCII representation to FP (the result is automatically in FR0), move FR0 to FR1, convert the second ASCII number, do the math. Convert FR0 back to ASCII. Store the number back into a user buffer.

11. THE FLOATING POINT ARITHMETIC PACKAGE

\$00D4 - \$00FF
\$057E - \$05FF

Figure 11.2.: RAM used by floating point package

The floating point package uses the following blocks of RAM.

If the floating point package is not used the above ram is free.

Useful data base variables and OS equates

FR0 \$00D4,6 (212): system FP buffer

FR1 \$00E0,6 (224): system FP buffer

CIX \$00F2 (242): INBUFF index

INBUFF \$00F3,2 (243): pointer to ASCII FP buffer

FLPTR \$00FC,2 (252): pointer to user FP buffer

LBUFF \$0580 (1408): result buffer for FP to ASCII

AFP \$D800 (55296): ASCII to FP

FASC \$D8E6 (55526): FP to ASCII

IFP \$D9AA (55722): integer to FP

FPI \$D9D2 (55762): FP to integer

ZFR0 \$DA44 (55876): clear FR0

ZF1 \$DA46 (55878): clear zero page FP buffer

FSUB \$DA60 (55904): FR0 - FR1

FADD \$DA66 (55910): FR0 + FR1

FMUL \$DADB (56027): FR0 * FR1

FDIV \$DB28 (56104): FR0 / FR1

FLD0R \$DD89 (56713): load FR0 by X,Y pointer

FLD0P \$DD8D (56717): load FR0 by FLPTR pointer

FLD1R \$DD98 (56728): load FR1 by X,Y pointer

FLD1P \$DD9C (56732): load FR1 by FLPTR pointer

FST0R \$DDA7 (56743): store FR0 at buffer by X,Y pointer

FST1P \$DDAB (56747): store FR0 at buffer by FLPTR pointer

FMOVE \$DDB6 (56758): move FR0 to FR1

EXP \$DDC0 (56768): e exponentiation

11.2. MAKING THE CALL

EXP10 \$DDCC (56780): base 10 exponentiation

PLYEVL \$DD40 (56640): polynomial evaluation

LOG \$DECD (57037): natural log of FR0

LOG10 \$DED1 (57041): base 10 log of FR0

11. THE FLOATING POINT ARITHMETIC PACKAGE

12. Boot software formats

There are three ways which programs may be booted (loaded automatically upon power-up):

- From the disk drive
- From the cassette recorder
- From a ROM cartridge

12.1. DISK BOOTED SOFTWARE

The disk drive is the primary source for programs (other than the BASIC interpreter in the computer ROM). A program booted from disk must be a machine language program. Secondly, the program is arranged on disk in a different manner from the DOS files.

When the computer is first turned on, it will attempt to read a program starting at sector one in disk drive one. The exceptions are, if a cartridge prevents the disk boot process or the [START] key is pressed. The program is expected to use all 128 bytes of each sector.

12.1.1. FORMAT OF A DISK BOOTED PROGRAM

A disk booted program begins at sector one on the disk and continues in sequence. The first six bytes of the first sector contain program information. The rest of the bytes contain the program itself.

Disk boot program header

1st byte \$00 flags, stored in DFLAGS [\$0240]

\$xx number of sectors used by program

\$xx address to start load

\$xx

\$xx initialization address

6th byte \$xx

7th byte \$xx start of program

12. Boot software formats

The flags byte is usually unused and should be zero. The load address is stored in BOOTAD [\$0242,2 (578)]. The initialization address is stored in DOSINI [\$000C,2 (12)]. After the program is completely loaded the computer will JSR to the address stored in DOSINI for initialization. It will then jump to the address stored in DOSVEC to run the program.

The initialization part of the program should set the bottom-of-free-RAM pointer, MEMLO [\$02E7,2 (743)], to point to the end of the program + 1. This will protect the program from the computer and other programs. The top-of-user-RAM pointer, APPMHI [\$000E,2 (14)], is also usually set to point to the same address. This will protect the program from the video hardware. It must also set DOSVEC [\$000A,2 (10)] to actually point to the run address of the program.

The initialization should of course end with and RTS. With DOSINI and DOSVEC properly set, the program will restart up pressing the [SYSTEM RESET] key. Remember that the load address of the program should be six bytes before where you want the program to reside in memory. The six byte header will load at the specified start address followed by the program.

12.2. CASSETTE BOOTED SOFTWARE

The cassette boot process is nearly identical to the disk boot process. The processes are so similar that cassette boot programs can usually be transferred directly to disk and vice-versa. The two differences are:

1. The cassette is booted instead of the disk if the [START] key is pressed when the power is turned on.
2. A bug in early operating systems requires the booted program to turn off the cassette motor with the following command.

```
LDA #$3C
STA PACTL [$D302]
```

12.3. CARTRIDGE BOOTED SOFTWARE

The Atari 800 has two cartridge slots. All other models have only one. The second cartridge slot, slot B on the 800, resides from \$8000 to \$9FFF. The first slot, slot A, resides from \$A000 to BFFF. If a cartridge is inserted in a slot it will disable any RAM in the same area.

Slot A, which is present in all models, can reside at the entire 16K used by both cartridges in the 800 (\$8000 to \$BFFF). Cartridges use the last six bytes for boot information. In cartridge A these bytes are from \$BFFA to \$BFFF. In cartridge B they are from \$9FFA to 9FFF. last six bytes of a cartridge

\$9FFA or \$BFFA xx start address

xx

00

xx flag byte

xx init address

\$9FFF or \$BFFF xx

Flag byte

bit 0 1 = allow disk boot

bit 2 0 = do not start cartridge after init

bit 7 1 = cartridge takes control before OS is initialized

The initialization process for the cartridge should be similar to that for disk and cassette. A minimum of an RTS instruction is required. The third byte of the cartridge tailer is used by the OS to check for the presence of a cartridge. This byte must be zero.

A 16K cartridge will use both cartridge areas and the cartridge B tailer area can be used for program code.

12.3.1. THE CARTRIDGE HARDWARE

Most cartridges consist of two ROM chips on a single circuit board. Moreover, both chip sockets have identical pin assignments. In other words, the chips can be switched to opposite sockets and the cartridge will still work. The difference is in the chips themselves. On one chip, the A12 pin acts as an active-low chip select. On the other the A12 pin acts as an active-high chip select. Therefore the state of the A12 pin selects between the two chips.

Cartridge slot pin assignments

BACK

111111

543210987654321

SRPNMLKJHFEDCBA

FRONT

1 1 = 16K A A13 (16K only)

2 A3 B GND

3 A2 C A4

4 A1 D A5

12. Boot software formats

5 A0 E A6

6 D4 F A7

7 D5 H A8

8 D8 J A9 ___

9 D1 K A12 (CS)/(CS)

10 D0 L D3

11 D6 ___ M D7

12 (CS) N A11

13 +Vcc P A10

14 +Vcc R NC

15 NC S NC

The BASIC interpreter resides in the memory used by cartridge A. In 400, 800 and 1200XL models, a BASIC cartridge is required to run BASIC programs. On other XL and XE models, inserting a cartridge into the slot or pressing the [OPTION] key upon power-up will disable the internal BASIC ROM. If BASIC is disabled without inserting another cartridge, the area from \$A000 to \$BFFF will contain RAM.

Useful data base variables and OS equates

APPMHI \$000E,2 (14): low limit of screen region

DOSVEC \$000A,2 (10): run and program reset vector

DOSINI \$000C,2 (12): init and reset init

CARTB \$8000 (32768): start of cartridge B

CARTA \$A000 (40960): start of cartridge A

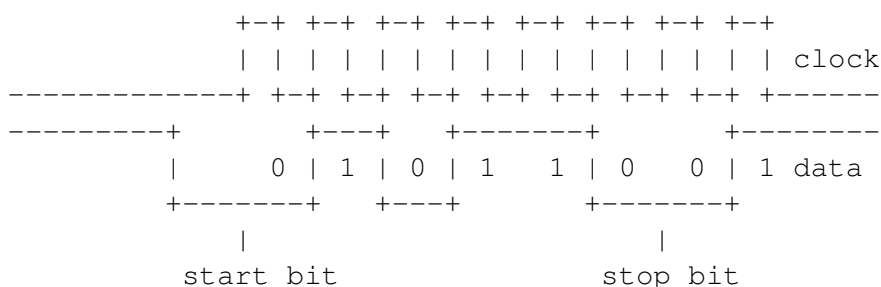
PACTL \$D302 (54018): port A control register Bit 3 controls the cassette motor

13. THE SERIAL INPUT/OUTPUT INTERFACE (SIO)

Most input and output with the Atari computer passes through the serial I/O bus. The SIO interface is rather complicated but you are unlikely to need to use it directly. CIO usually handles SIO for you. However, if you want to design your own I/O device and its associated handler, you need to know how to use the SIO. SIO transfers data at a rate of 19,200 baud on separate input and output lines. The data is sent one byte at a time, LSB first, in an asynchronous format. There are also clock-in and clock-out lines.

There is a signal on the clock-out line but it is not used by any present devices. The clock-in line is available for synchronous transfer but is not used by the OS. The signal on the clock-out line goes high at the leading edge of each bit and goes low in the middle of each bit.

One byte of SIO data



The SIO interface is used much like the resident disk handler. In fact, it uses the same device control block as the resident disk handler. After the control block parameters are set, a JSR is made to the SIO entry vector, SIOV, at \$E459 (58457).

Device control block (for SIO)

DDEVIC [\$0300 (768)]

Serial bus I.D. Set by handler or program.

DUNIT [\$0301 (769)]

Device number if more than one.

DCOMND [\$0302 (770)]

Device command byte.

13. THE SERIAL INPUT/OUTPUT INTERFACE (SIO)

DSTATS [\$0303 (771)]

Before the SIO call, this byte tells whether the operation is read, write or that there is no data transfer associated with the command. After the call this byte will hold the status (error/no error code) of the operation.

DSTATS format before command

```
    7 6 5 4 3 2 1 0
-----
|W|R| not used |
-----
```

If both W and R are 0, there is no data transfer.

DBUFLO [\$0304 (772)]

DBUFHI [\$0305 (773)]

Points to the data buffer for either input or output.

DTIMLO [\$0306 (774)]

Timeout value (response time limit) in 64/60ths of a second to be set by handler or program.

DBYTLO [\$0308 (776)]

DBYTHI [\$0309 (777)]

Number of bytes to be transferred, set by handler or program. This parameter is not required if the DSTATS specifies no data transfer.

DAUX1 [\$030A (778)]

DAUX2 [\$030B (779)]

These parameters are sent to the device as part of the command frame.

13.1. USING THE SIO INTERFACE

All commands on the serial bus must originate from the computer. The peripherals will present data on the bus only when commanded to do so.

Any operation on the serial bus begins with a five byte command frame. While the command frame is being sent, the command line of the serial connector is 0.

Command frame format

\$xx DDEVIC

\$xx DCOMND

\$xx DAUX1

\$xx DAUX2

\$xx checksum

The first four bytes of the command frame come from the device control block. the checksum is the sum of the other four bytes with the carry added back after each addition.

If both R and W of the DSTATS are 0, no data is sent to, or expected from the peripheral, after a command frame is sent. However, the device is usually expected to send an ACK byte (\$41) after the command frame is sent. If the command frame is invalid, an NAK byte (\$4E) should be sent.

If the operation is output (W = 1) the computer will send a data frame after it receives the ACK of the command frame. It then expects an ACK after the data frame is sent. If the operation is an input (R = 1) the computer expects a data frame from the peripheral after the ACK. With either input or output, a "complete" code (\$43) should be sent to the computer when the operation is finished. The "complete" code would follow the ACK of the data frame with an output operation.

If the operation is not completed for some reason, the peripheral should send an error code (\$45) instead of "complete".

SIO data frame

```
byte 1 $xx\  
           > data bytes  
byte n $xx/  
byte n+1 $xx checksum
```

SIO commands

READ \$52

WRITE \$57

STATUS \$53

PUT \$50

FORMAT \$21

DOWNLOAD \$20

READADDR \$54

READ SPIN \$51

MOTOR ON \$55

VERIFY

13. THE SERIAL INPUT/OUTPUT INTERFACE (SIO)

SECTOR \$56

Present SIO device I.D.s

DISK \$31 - \$34 (D1 - D4)

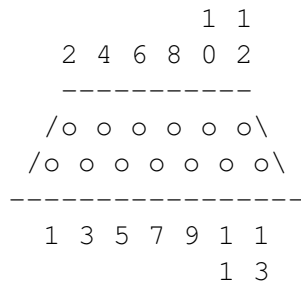
PRINTER \$40

RS-232-C \$50 - \$53 (R1 - R4)

13.2. THE SERIAL CONNECTOR

The serial connectors on the computer and all peripherals are identical. Nearly all peripherals have two serial connectors. Either connector may be used for any connection. The serial bus is designed so that peripherals can be daisy-chained together. The following is a diagram of the serial connector.

The serial connector pin-out



1. clock in (to computer)
2. clock out
3. data in
4. GND
5. data out
6. GND
7. command (active low)
8. cassette motor control
9. proceed (active low)
10. +5V/ready
11. audio in

13.2. THE SERIAL CONNECTOR

12. +12V (400/800)

13. interrupt (active low)

Proceed goes to pin 40 (CA1) of the PIA. It is not used by present peripherals.

Interrupt goes to pin 18 (CB1) of the PIA. It is not used by present peripherals.

Pin 10 doubles as a 50mA +5V peripheral power supply and a computer ready signal.

Useful database variables and OS equates

SIOV \$E459 (58457): serial port handler entry

DDEVIC \$0300 (768): device ID

DUNIT \$0301 (769): device number

DCOMND \$0302 (770): command byte

DSTATS \$0303 (771): status byte

DBUFLO \$0304 (772): data buffer pointer

DBUFHI \$0305 (773):

DTIMLO \$0306 (774): timeout value

DBYTLO \$0308 (776): number of bytes to transfer

DBYTHI \$0309 (777):

DAUX1 \$030A (778): sent to device

DAUX2 \$030B (779): sent to device

13. THE SERIAL INPUT/OUTPUT INTERFACE (SIO)

14. THE HARDWARE CHIPS

The previous chapters described the operating system of the computer. The following chapters will examine the hardware which supports the 6502 and the hardware's associated software.

14.1. THE GTIA CHIP

The GTIA (George's Television Interface Adapter) is the main video circuit in the computer. It controls the following functions.

GTIA functions

- Priority of overlapping objects
- Color and brightness, including information from the antic chip.
- Player/missile control.
- console switches and game control triggers.

14.2. THE ANTIC CHIP

The main job of the ANTIC chip is interpreting the display buffer for the GTIA chip. The ANTIC chip is somewhat of a processor in its own right. The program which runs it is called the display list and usually resides just before the display buffer in memory. The ANTIC chip operates independent of the 6502. It operates by direct memory access (DMA). The ANTIC chip gives a HALT signal the 6502, causing the 6502 to give up control of the address bus. The ANTIC chip can then read any data it needs to from memory.

ANTIC chip functions

- DMA (Direct Memory Access) control.
- NMI (Non-Maskable Interrupt) control.
- LIGHT PEN READING
- WSYNC (wait for horizontal sync)

14.3. THE POKEY CHIP

The most important jobs of the POKEY chip are reading the keyboard and operating the serial port. It also has the following functions.

POKEY chip functions

- Keyboard reading.
- Serial port.
- Pot (game paddles) reading.
- Sound generation.
- System timers.
- IRQ (maskable interrupt) control.
- Random number generator.

14.4. THE PIA CHIP

The PIA (Parallel Interface Adapter) is a commonly used I/O chip. It consists of two 8 bit parallel ports with hand shaking lines. In the Atari, it has the following functions.

- Game controller port control (bi-directional).
- Peripheral control and interrupt lines.

Registers in the hardware chips are treated as memory addresses. Many of the registers are write only. These registers cannot be read from after they are written to. Other registers control one function when written to and give the status of an entirely different function when read from. Still other registers are strobes. Any command which causes the address of one of these registers to appear on the address bus will cause their functions to be performed.

The write only registers have shadow registers in RAM. Data to be put in the registers is usually put into the shadow registers. The data in the shadow registers is automatically moved to the operating registers during vertical blank.

For register use and address, see the previous chapters on the associated functions.

15. DISPLAY LISTS

The program which runs the ANTIC chip is called the display list. Much like a microprocessor, the ANTIC chip has a program counter, called the display list counter. The display list counter is a 16 bit register. However, the six most significant bits are semi-fixed.

These bits can only be changed with a jump instruction. The result of this is that the display list counter cannot cross a 1K memory boundary (i.e. \$A3FF to \$A400) without using a jump instruction.

The ANTIC chip holds the address of the display list in DLISTL [\$D402 (54274)] and DLISTH [\$D403 (54275)]. SDLSTL [\$0230 (560)] and SDLSTH [\$0231 (561)] are the shadow registers for DLISTL and DLISTH. The ANTIC chip also has a memory scan counter. This register scans the display buffer for data to be interpreted and displayed. Once loaded, the memory scan counter's 4 most significant bits are fixed. The result is that the memory scan counter cannot cross a 4K memory boundary (i.e. \$AFFF to \$B000) without being reloaded.

15.1. DISPLAY LIST INSTRUCTIONS

There are three basic instructions in the display list. The type of instruction is determined by bits 0,1,2 and 3 of an instruction byte. The other four bits give auxiliary parameters for the instruction. Bit 7 always enables a display list interrupt (DLIs).

Display list instruction format

```

 7 6 5 4 3 2 1 0
-----
|I|n|n|n|0|0|0|0|
-----
\ / \ /
---
| |
| 0 = display blank lines
|
0-7 = number of blank lines (1-8)
 7 6 5 4 3 2 1 0
-----
|I|W| | |0|0|0|1|
```

15. DISPLAY LISTS

```
-----  
| \ /  
| -----  
| |  
| 1 = jump (3 byte instruction)  
|  
0 = jump and display one blank line  
1 = jump and wait for vertical blank  
7 6 5 4 3 2 1 0  
-----  
|I|R|H|V|M|M|M|M|  
-----  
| | | | \ /  
| | | | -----  
| | | | |  
| | | | 2-F = display one line of graphics in  
| | | | ANTIC mode 2-F  
| | | 1 = horizontal scroll enabled  
| | |  
| | 1 = vertical scroll enabled  
| |  
| 1 = reload memory scan counter with next two bytes  
|  
1 = display list interrupt, all instructions
```

In the display instruction, the ANTIC mode is different from the CIO graphics mode. However, each CIO graphics mode uses a particular ANTIC mode. Below are descriptions of the ANTIC modes with their associated graphics (CIO) modes.

15.1.1. ANTIC MODE 2 (Graphics 0)

Uses 8 pixel by 8 pixel characters, 40 characters horizontal, 8 TV scan lines vertical. Only one color can be displayed at a time.

15.1.2. ANTIC MODE 3

8 X 10 pixel, Graphics 0 type characters. This mode requires a custom character set. The advantage is that it allows true decenders. The custom C-set is still 8 X 8 pixels. Lower-case letters with decenders have the bottom row of pixels put on the top row. Lower-case "y" for ANTIC mode 3

C-set Display


```

-----
|  XXXXX  | | |
|  |  |  |
|  |  |  |
|  XX XX  | | XX XX  |
|  XX XX  | | XX XX  |
|  XX XX  | | XX XX  |
|  XXXXX  | | XXXXX  |
|  XX  | | XX  |
----- |  XXXXX  |
|  |
-----

```

15.1.3. ANTIC MODE 4 (graphics 12 on XL and XE)

This mode has characters the same size as graphics 0. However, the characters are only 4 X 8 pixels. This gives only half the horizontal resolution of graphics 0. The advantage is that up to four colors of "graphics 0" characters can be displayed at once. This mode also requires a custom C-set. Below is a comparison of the normal C-set to one which works with the ANTIC 4 mode.

Upper-case "A" for ANTIC modes 2 and 4 mode 2 mode 4

```

-----
|  |  |  |
|  XX  | | yy  |
|  XXXX  | | yy  |
|  XX XX  | |xx zz  |
|  XX XX  | |xx zz  |
|  XXXXXX  | |xxyyzz  |
|  XX XX  | |xx zz  |
|  |  |  |
-----

```

xx, yy and zz represent two bit binary numbers, controlling one pixel each. These numbers determine which color register a pixel is assigned to: (COLOR0, COLOR1, COLOR2 or COLOR3).

15.1.4. ANTIC mode 5

Antic mode five is identical to ANTIC mode 4 except the characters are displayed twice as tall. This makes only 12 lines on the screen.

15. DISPLAY LISTS

15.1.5. ANTIC MODE 6 (Graphics 1)

This mode uses 8 X 8 pixel characters except they are displayed twice as wide as in ANTIC mode 2. There are 3 colors available at once but only one case (upper or lower) can be displayed at a time. The data base variable CHBAS [\$02F4 (756)] controls the character, [\$E0 (224) = upper-case, \$E2 (226) = lower-case] The color/character is controlled by either the color statement or the ATASCII number of the character printed. Control characters are controlled by COLOR0, upper-case characters by COLOR1 and lower-case characters by COLOR2. Remember that all characters print as upper-case alpha characters, but of different colors.

15.1.6. ANTIC MODE 7 (Graphics 2)

This mode is identical to mode 6 except the characters are displayed twice as tall. This results in only 12 lines possible on the screen.

15.1.7. ANTIC MODE 8 (Graphics 3)

This is the first graphics (non-character) mode. This mode, as other non-character graphics modes do, uses data in the display buffer as a bit map to be displayed. A command to display in mode 8 will cause the ANTIC chip to read the next 10 bytes in the display buffer. Each pair of bits will control one pixel as in mode 4. However, the pixels are blocks the same size as a Graphics 0 (ANTIC 2) characters.

15.1.8. ANTIC MODE 9 (Graphics 4)

This is similar to ANTIC mode 8 except each byte controls 8 pixels (instead of 4) and only one color can be displayed at a time. The pixels are also half the size of those in ANTIC mode 8.

15.1.9. ANTIC MODE A (Graphics 5)

This mode uses 20 bytes per line/command. As in ANTIC mode 8, each pair of bits controls one pixel. The result is that the pixels are the same size as in ANTIC mode 9 but four colors can be displayed at once.

15.1.10. ANTIC MODE B (Graphics 6)

As in mode A, there are 8 pixels per byte and only one color. The pixels are half the size as in mode A.

15.1.11. ANTIC MODE C

Like mode B except the pixels are half as tall (only one T.V. line).

15.1.12. ANTIC MODE D (Graphics 7)

40 Bytes per line, each byte controls 4 pixels. The pixels are 1/4 as large as in ANTIC mode 8 (Graphics 3).

15.1.13. ANTIC MODE E (Graphics 15 on XL and XE)

Like mode D except the pixels are half as tall (one T.V. line). Antic mode E is sometimes called Graphics 7.5

15.1.14. ANTIC mode F (Graphics 8, 9, 10 and 11)

This is the highest resolution mode. Pixels are 1/8 the size of ANTIC mode 8 or mode 2 characters. It uses 40 bytes per line, each byte controlling 8 pixels, unless the GTIA chip intervenes. Only one color can be displayed at a time.

15.2. DISPLAY LIST EXAMPLES

When CIO opens a channel to the screen, it sets up the proper display list for the ANTIC chip. The following are the things CIO must handle when setting up the display list.

Display list duties as used by CIO display a certain number of blank lines at the top of the screen. Load the memory scan counter with the address of the display data buffer. Display the required number of lines in the required ANTIC mode. Set up a jump instruction if the display list crosses a 1K memory boundary.

Set up a reload-memory-scan-counter instruction if the display data buffer crosses a 4K memory boundary. CIO assumes that the display data buffer will butt against an 8K memory boundary. If a program causes the display buffer to cross a 4K boundary (by changing RAMTOP [\$006A (106)] to point to an address which is not at an 8K boundary) the screen will be scrambled. This is not usually a problem if the graphics mode doesn't require a large block of memory.

15.2.1. SAMPLE DISPLAY LIST

Below is an example of a Graphics 0 display list as CIO would set it up. Display list for Graphics 0 assuming BASIC starts at \$A000.

15. DISPLAY LISTS

address	instruction	explanation
	Dec.	Hex.
\$9C20	112	\$70 \
	112	\$70 >---- 24 blank lines (8 each command)
	112	\$70 /
	66	\$42 ----- load memory scan counter with
\$9C24	64	\$40 __ next two bytes and display one line
	156	\$9C / \ of ANTIC 2 characters
	2	\$02 -\
	2	\$02 \- address of display data buffer
	2	\$02
	2	\$02 \--- 2nd ANTIC 2 instruction
	-	---
	2	\$02 ----- 24th ANTIC 2 instruction
	65	\$41 \
	32	\$20 >---- jump back to start of list
	156	\$9C /
\$9C40	???	?? first byte of display data buffer
	---	--
\$9FFF	???	?? last byte of buffer
\$A000		start of ROM

A display list for a higher resolution graphics mode would require more instructions and might cross a 1K boundary. It would then include a jump instruction to cross the boundary.

15.3. MULTIPLE DISPLAYS

It is possible to set up multiple displays and use one at a time. The technique of changing from one display to another is called page flipping. Below is the simplest way to set up two displays.

setting up two displays

1. Call a graphics mode through CIO or by using a BASIC GRAPHICS command.
2. Store the display list pointers, SDLSTL and SDLSTH, and the CIO screen pointer, SAVMSC [\$0058,2 (88)].
3. Move the start-of-ROM pointer, RAMTOP [\$006A (106)] to below the current display list. RAMTOP is a one byte pointer so it changes in increments of one page (256 bytes).
4. make another graphics call as in the first step.
5. store the new display list pointer and CIO screen pointer.

This will set up two displays, each with its own display list. If the displays are in the same graphics mode, or you will not make any changes in the displays with CIO commands, (PLOT, PRINT, etc.) you can flip between the two simply by changing the display list pointer.

If the screens are in the same graphics mode and you want to change which one to do CIO commands to, Change the CIO screen pointer, SAVMSC [\$0058,2 (88)]. This way, you can display one screen while drawing on the other.

If you want to do CIO commands to screens of different graphics modes, you will have to move RAMTOP and do a graphics call to change screens. If your manipulation of RAMTOP causes the display data buffer to cross a 4K boundary, the screen may be scrambled.

15.4. DISPLAY LIST INTERRUPTS

DLIs are not used by the operating system. However, other programs can initiate and use them. Use the following steps to set up display list interrupts.

Setting up DLIs

1. Set bit 7 of the display list instruction for the line before you want the interrupt to occur. (The interrupt routine should set WSYNC and wait for the next line to execute.)
2. Set bit 7 of NMIEN [\$D40E (54286)] to enable DLIs.
3. Set the DLI routine vector, VDSLST [\$0200,2 (512)] to point to your machine language DLI routine.

Your DLI routine should set WSYNC [\$D40A (54282)]. STA WSYNC will do. This will cause the 6502 to wait for the next horizontal sync. This will keep the DLI routine from changing something in the middle of a T.V. line. The DLI routine must end with an RTI instruction.

15.5. SCROLLING

Scrolling is controlled by a combination of scroll position registers, and changing the memory scan counter. Basically, course scrolling is done by reloading the memory scan counter and fine scrolling is done by changing the scroll registers.

15.5.1. VERTICAL SCROLLING

Vertical scrolling is very simple. Follow the steps below to set up vertical scrolling of graphics.

15. DISPLAY LISTS

Steps to use vertical scrolling

1. Set bit 4 of the first byte of the display list instruction for each line to be scrolled.
2. Put the number of T.V. lines to offset the graphics vertically in the vertical scroll register, VSCROL [\$D405 (54277)]

The vertical scroll register can offset the graphics upward by 0 - 7 T.V. lines in the 24 line graphics modes (ANTIC modes 2 and 4). In 12 line graphics modes (ANTIC modes 5 and 7) it can vertically offset the graphics by 0 - 15 T.V. lines. To offset the graphics an 8th (or 16th) line, the scroll register is reset to 0 and the memory scan counter is reloaded with the address of the next line of graphics in the display data buffer. If the entire screen is being scrolled, the load-memory-scan-counter command (near the beginning of the display list) is changed to point to the address of the second line of graphics.

15.5.2. HORIZONTAL SCROLLING

Horizontal scrolling works much like vertical scrolling. It is enabled by setting bit 5 of the instruction for each line to be scrolled. The horizontal scroll register, HSCROL [\$D404 (54276)], sets the offset. The small difference is that graphics are moved twice as far per change (two graphics 8 pixels instead of one). Also, when HSCROL = 0 the graphics are offset beyond the left edge of the screen by 16 color clocks (32 Graphics 8 pixels). When HSCROL = 15, the graphics line is shifted one color clock (2 Graphics 8 pixels) to the left of the screen.

The big difference is that the memory scan counter gets messed up. This means that you must use a reload-memory-scan-counter command for each line of graphics. This is a major modification of the display list. It will require you to move and build the list yourself.

The advantage of this is that you can have a scrolling window in a large graphics map. The technique is to move the window by reloading the memory scan counter, then fine scrolling to the invisible bytes beyond the edges of the screen.

useful data base variables and OS equates SAVMSC \$0058,2 (88): pointer to current screen for CIO commands

RAMTOP \$006A (106): start-of-ROM pointer (MSB only)

VDSLST \$0200,2 (512): DLI vector

RAMSIZ \$02E4 (740): permanent start-of-ROM pointer (MSB only)

DLISTL \$D402 (54274): display list pointer low byte

DLISTH \$D403 (54275): " high byte

HSCROL \$D404 (54276): horizontal scroll register

VSCROL \$D405 (54277): vertical scroll register

15.5. SCROLLING

NMIEN \$D40E (54286): NMI enable (DLIs)

Shadow registers

SDLSTL \$0230 (560): DLISTL

SDLSTH \$0231 (561): DLISTH

15. DISPLAY LISTS

16. PLAYER AND MISSILE (PM) GRAPHICS

Players and missiles (called sprites on some computers) are movable objects which are independent of the normal graphics. Player and missile graphics are fairly straight forward. Once the computer is set-up for PM graphics, five 8-pixel-wide columns can be displayed on the screen. The horizontal resolution (width of each pixel) and the vertical resolution (number of scan lines per pixel) are variable. The horizontal position of each column is determined by its horizontal position register. Each column is simply a representation of a bit map in a certain block of memory. If you want to draw an object on the screen, you simply put a bit map representing it in the proper memory block. The vertical position of an object is determined by the location of its bit map in memory. For example, if you want to draw a happy face in the middle of the screen, you put a happy face bit map in the middle of one of the memory blocks controlling one of the columns.

One column (player) displayed on the screen

```

----- first byte of a block
|           |
|           |
-----
| |         |           |
| |         |           |
| |         |           |
| |         |           |
| |         |           |
| |         |           |
| |         |           |
| |  +++++  |           |
| |  +   +  |           |
| | + + + + |           |
| | +       + |           |
| | ++      ++ |           |
| | + + + + + | --object |
| | +   +   | bit map   |
| |  +++++  |           |
| |         |           |
| |         |           |
| |         |           |

```

16. PLAYER AND MISSILE (PM) GRAPHICS

```
-----  
|           |  
|           |  
----- last byte of a block
```

Horizontal positions

```
    $00 $30                               $CE  $FF  
    (0) (48)                             (206) (255)  
    |   |                               |   |  
    |   Left edge   right edge         |   |  
    |               |                 |   |  
    Far left       far right
```

To move the happy face vertically you would move the entire bit map in memory. To move the happy face horizontally you change the number in the horizontal position register for the proper player. One of the players can be (and often is) split into four columns of two pixels wide each. These columns are then called missiles. In this case, each missile has its own horizontal position register.

16.1. SETTING UP PM GRAPHICS

PM graphics are enabled by the direct memory access control register, DMACTL [$\$D400$ (54272)]. The program using PM graphics will usually use the shadow register, SDMCTL [$\$022F$ (559)].

DMACTL (SDMCTL)

```
    7 6 5 4 3 2 1 0  
-----  
|0|0| control  |  
-----  
bits  
5      1 = enable display list reading  
4      0 = one line player resolution  
        1 = two line player resolution  
3      1 = enable four players  
2      1 = enable fifth player or missiles  
1 & 0 00 = no background  
        01 = narrow background (128 color clocks, 1 color clock equals  
        10 = normal background (160 color clocks)  
        11 = wide background (192 color clocks)
```

16.1. SETTING UP PM GRAPHICS

Normally, bits 5 and 1 are set to 1. Bits 4, 3 and 2 are used to enable players and/or missiles accordingly. Once DMACTL is set up for the type of PM graphics to enable, the graphics control register, GRACTL [5D01D (53277)], is used to actually enable the PM graphics.

```
GRACTL
 7 6 5 4 3 2 1 0
-----
|not used | | | |
-----

Bits
2 1 = latch paddle triggers
1 1 = enable four players
0 1 = enable fifth player or missiles
```

If only DMACTL is set up, the ANTIC chip will access memory for PM graphics but will not display them. Next, the memory area used for the PM bit maps must be set. This block must start on a 2K (8 page) boundary if single line resolution is used and a 1K (4 page) boundary for two line resolution. The page number where the bit map starts is stored in the PM base register, PMBASE [5D407 (54279)]. For one line resolution this number will be a multiple of 8. For two line resolution it will be a multiple of 4. PMBASE holds the MSB of the address of the PM bit map. The LSB will always be 0 so it need not be specified.

```
The PM bit maps
2 line resolution
128 bytes (1/2 page)
per player
----- start + 0
|                |\
+-----+ 1-1/2 page
|                | (384 bytes)
+=====+ unused
|                | /
+-----+ +$180 (384)
|M3 |M2 |M1 |M0 | fifth player or missiles
+=====+ +$200 (512)
| player 0 map |
+-----+ +$280 (640)
| player 1 map |
+=====+ +$300 (768)
| player 2 map |
+-----+ +$380 (896)
| player 3 map |
+=====+ +$400 (1024)
```

16. PLAYER AND MISSILE (PM) GRAPHICS

```

1 line resolution
256 bytes (1 page)
per player
----- start + 0
|                                     |\
+                                     +
|                                     |
+=====+
|                                     | 768 bytes
+                                     +
|                                     | (3 pages)
+=====+
|                                     | unused
+                                     +
|                                     | /
+=====+ +$300 (768)
|   |   |   |   | fifth player
+M3 |M2 |M1 |M0 | or missiles
|   |   |   |   |
+=====+ +$400 (1024)
|                                     |
+ player 0 map +
|                                     |
+=====+ +$500 (1280)
|                                     |
+ player 1 map +
|                                     |
+=====+ +$600 (1536)
|                                     |
+ player 2 map +
|                                     |
+=====+ +$700 (1792)
|                                     |
+ player 3 map +
|                                     |
+=====+ +$800 (2048)

```

Example of using P/M graphics in BASIC

```

0 REM ---LABEL REGISTERS ETC
10 LINES=2
20 VERT=120
22 IF LINES=2 THEN VERT=VERT/2

```

```

30 PM0=1024
32 IF LINES=2 THEN PM0=PM0/2
40 HORIZ=120
50 PCOLR0=704
60 SDMCTL=559
70 SIZEP0=53256
80 HPOSP0=53248
90 SDMCTL=559
100 PMRAM=PEEK(106)-16
110 PMBASE=54279
120 GRCTL=53277
130 PMSTART=PMRAM*256+PM0
200 REM ---SET REGISTERS
210 POKE SDMCTL,62
212 IF LINES=2 THEN POKE SDMCTL,46
220 POKE SIZEP0,1
230 POKE HPOSP0,HORIZ
240 POKE PCOLR0,88
250 POKE PMBASE,PMRAM
260 POKE GRCTL,3
300 REM ---DRAW PLAYER
310 POKE PMSTART+VERT,60
320 POKE PMSTART+VERT+1,66
330 POKE PMSTART+VERT+2,165
340 POKE PMSTART+VERT+3,129
350 POKE PMSTART+VERT+4,195
360 POKE PMSTART+VERT+5,189
370 POKE PMSTART+VERT+6,66
380 POKE PMSTART+VERT+7,60

```

The above program will draw a happy face in about the middle of the screen using player 0. To move the player horizontally, poke a different number into HPOSP0. To draw the player in a different vertical position, change VERT. To use a different player or missile, use the memory maps above to find the starting address of the player you want to use. For example, to use player 1 change line 40 to PM1=1280. Then change line 130 to PMSTART=PMRAM*256+PM1. The variable "LINES" determines the vertical resolution. The number poked into SIZEP0 determines the width.

16.2. P/M PRIORITY

The priorities of players, missiles and non-P/M graphics can be controlled by the PRIOR register [\$D10B (53275)] and its shadow register, GPRIOR [\$26F (623)]. Objects with higher priority will appear to move in front of lower priority objects. The format of PRIOR is as follows:

16. PLAYER AND MISSILE (PM) GRAPHICS

PRIOR bit assignment

7 6 5 4 3 2 1 0

 | | | | | | | |

1 6 3 1 8 4 2 1

2 4 2 6

8

Bits Use

7-6

Control the GTIA graphics modes.

00 = normal

01 = mode 9

10 = mode 10

11 = mode 11

5

1 = multiple color player enable. Permits overlapping of p

4

1 = fifth player enable. All missiles will assume the color controlled by COLOR3 [\$2C7 (711)]. missiles are positioned together to make the fifth player.

3-0

Controls the priorities of players, missiles and other grap

The following chart may need some clarification. In the chart:

PM0 = player 0 and missile 0

C0 = COLOR0, plotted graphics controlled by color register 0 in the command.

P5 = all four missiles when combined into one player.

BAK = the background, known as COLOR4 or color register 4 in the SETC

Etc.

Bits 0-3 of PRIOR and P/M priorities

Bit 3=1 2=1 1=1 0=1

C0 C0 PM0 PM0 highest

C1 C1 PM1 PM1 priority

PM0 C2 C0 PM2

PM1 C3+P5 C1 PM3

```

PM2 PM0 C2 C0
PM3 PM1 C3+P5 C1
C2 PM2 PM2 C2
C3+P5 PM3 PM3 C3+P5 lowest
BAK BAK BAK BAK priority

```

Only one priority bit can be set at a time. If more than one priority bit is 1, overlapping areas of conflicting priorities will turn black.

16.3. COLLISIONS

Each player or missile has a register showing overlap (collisions) with other objects. Each player has two registers assigned to it; one to detect collisions with other players and one to detect collisions with plotted objects. Likewise each missile has two registers; one to detect collisions with players and one to detect collisions with plotted objects. Careful use of these 16 registers can detect any type of collision.

Each register uses only the lower 4 bits. The bits which equal 1 tell what the associated object has collided with. For example, to detect collisions of player 1 to other players examine P1PL [\$D00D (53261)].

```

P1PL, player 1 to player collisions
    7 6 5 4 3 2 1 0
    -----
P1PL |unused | | | | |
    -----
                8 4 2 1
3 = 1 collision with player 3
2 = 1 collision with player 2
1 = 1 invalid
0 = 1 collision with player 0

```

When looking for collisions with plotted objects, the bit number tells what color register is assigned to the object the collision was with. For example, to detect collisions between player 1 and plotted objects (officially called the play field), P1PF [\$D005 (53253)] is used.

P1PF, player 1 to plotted object collisions

```

    7 6 5 4 3 2 1 0
    -----
P1PF |unused | | | | |
    -----
                8 4 2 1

```

16. PLAYER AND MISSILE (PM) GRAPHICS

```
3 = 1 collision with COLOR3
2 = 1 " COLOR2
1 = 1 " COLOR1
0 = 1 " COLOR0
```

Once a collision occurs it remains indicated in its collision register. To clear out all collision registers, write anything to HITCLR [\$D01E (53278)]. STA HITCLR or POKE 53278,0 will do.

Useful database variables and OS equates HPOSP0 \$D000 (53248): write: horizontal position of player 0

M0PF " " : read: missile 0 to plotted graphics collisions

HPOSP1 \$D001 (53249): write: horizontal position of player 1

M1PF " " : read: missile 1 to plotted graphics collisions

HPOSP2 \$D002 (53250): write: horizontal position of player 2

M2PF " " : read: missile 2 to plotted graphics collisions

HPOSP3 \$D003 (53251): write: horizontal position of player 3

M3PF " " : read: missile 3 to plotted graphics collisions

HPOSM0 \$D004 (53252): write: horizontal position of missile 0

P0PF " " : read: Player 0 to plotted graphics collisions

HPOSM1 \$D005 (53253): write: horizontal position of missile 1

P1PF " " : read: Player 1 to plotted graphics collisions

HPOSM2 \$D006 (53254): write: horizontal position of missile 2

P2PF " " : read: Player 2 to plotted graphics collisions

HPOSM3 \$D007 (53255): write: horizontal position of missile 3

P3PF " " : read: Player 3 to plotted graphics collisions

SIZEP0 \$D008 (53256): write: size of player 0

M0PL " " : read: missile 0 to player collisions

SIZEP1 \$D009 (53257): write: size of player 1

M1PL " " : read: missile 1 to player collisions

SIZEP2 \$D00A (53258): write: size of player 2

M2PL " " : read: missile 2 to player collisions

SIZEP3 \$D00B (53259): write: size of player 3

M3PL " " : read: missile 3 to player collisions
 SIZEM \$D00C (53260): write: widths for all missiles
 POPL " " : read: player 0 to other player collisions
 GRAFP0 \$D00D (53261): write: player 0 graphics (used by OS)
 P1PL " " : read: player 1 to other player collisions
 GRAPF1 \$D00E (53262): write: player 1 graphics
 P2PL " " : read: player 2 to other player collisions
 GRAFP2 \$D00F (53263): write: player 2 graphics
 P3PL " " : read: player 3 to other player collisions
 GRAPF3 \$D010 (53264): write: player 3 graphics
 GRAFM \$D011 (53265): write: missile graphics (used by OS)
 COLPM0 \$D012 (53266): color for player/missile 0
 COLPM1 \$D013 (53267): color for player/missile 1
 COLPM2 \$D014 (53268): color for player/missile 2
 COLPM3 \$D015 (53269): color for player/missile 3
 COLPF0 \$D016 (53270): color register 0
 COLPF1 \$D017 (53271): color register 1
 COLPF2 \$D018 (53272): color register 2
 COLPF3 \$D019 (53273): color register 3
 COLBK \$D01A (53274): background color (register 4)
 PRIOR \$D01B (53275): priority select, GTIA modes
 GRACTL \$D01D (53277): graphics control
 HITCLR \$D01E (53278): writing anything clears all collision bits
 DMACTL \$D400 (54272): direct memory access (DMA) control
 PMBASE \$D407 (54279): start of P/M memory
 Shadow registers
 SDMCTL \$022F (559): DMACTL
 GPRIOR \$026F (623): PRIOR
 PCOLR0 \$02C0 (704): COLPM0
 PCOLR1 \$02C1 (705): COLPM1
 PCOLR2 \$02C2 (706): COLPM2

16. PLAYER AND MISSILE (PM) GRAPHICS

PCOLR3 \$02C3 (707): COLPM3

COLOR0 \$02C4 (708): COLPF0

COLOR1 \$02C5 (709): COLPF1

COLOR2 \$02C6 (710): COLPF2

COLOR3 \$02C7 (711): COLPF3

COLOR4 \$02C8 (712): COLBK

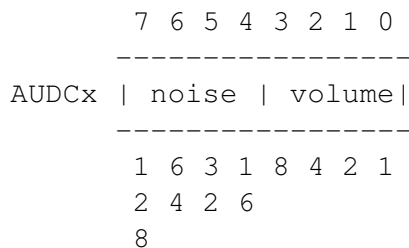
17. SOUND

Generating sound can be very simple. For simple sounds there are four audio channels, each controlled by two control registers.

17.1. GENERATING SOUNDS

To generate a sound in channel 1, put the frequency and volume codes into the frequency and control registers. The frequency register for channel 1, AUDF1 [\$D200 (53760)] can have any number from 0 to \$FF (255). 0 causes the highest frequency; 255 causes the lowest. The volume/noise (control) register for channel 1, AUDC1 [\$D201 (53761)] is more complicated.

Audio channel control (volume/noise) register



The noise bits can have various values. The best way to learn to use them is by experimentation. The technical details of the polynomial counters which generate the noise has little bearing on what is heard.

The two special values of interest are: \$1 (volume+16 in decimal), which causes a DC voltage proportional to the volume bits and; \$A (volume+160), which causes a pure tone (square wave). The volume bits select the relative volume, 0=off. Therefore, the number, \$A8 (168 [8+160]) in AUDC1, will cause the frequency selected by AUDF1 to be a pure tone of medium volume.

In BASIC the dirty work is done for you. The SOUND command will do all the calculations for you. The Sound command format is shown below.

The BASIC sound command format

```
SOUND channel, frequency, noise, volume
```

17. SOUND

The channel numbers is 0 to 3 instead of 1 to 4. The frequency, 0 to 255, is put into the frequency register. The noise is put into the high bits of the channel control register with volume in the low bits.

Therefore...

```
SOUND 0,125,10,8
```

will produce a pure tone of medium frequency and volume in channel 0 (called channel 1 in assembly language).

17.2. ADVANCED SOUND

The Audio Control register, AUDCTL [D208 (53768)], (not to be confused with the four audio channel control registers), adds more control for assembly language programmers. Again, to go into technical details will be less productive than experimentation.

The audio control register. (AUDCTL)

	7	6	5	4	3	2	1	0
AUDCTL								
	1	6	3	1	8	4	2	1
		2	4	2	6			
			8					
7	0	= 17 bit polynomial noise						
	1	= 9 bit below polynomial noise						
6	0	= clock channel 1 with 64 KHz						
	1	= clock channel 1 with 1.79 MHz						
5	0	= clock channel 3 with 64 KHz						
	1	= clock channel 3 with 1.79 MHz						
4	0	= clock channel 2 with 64 KHz						
	1	= clock channel 2 with channel 1						
3	0	= clock channel 4 with 64 KHz						
	1	= clock channel 4 with channel 3						
2	1	= insert logical high-pass filter in channel 1, clocked by channel 1						
1	1	= insert logical high-pass filter in channel 2, clocked by channel 2						
0	0	= 64 KHz main clock						
	1	= 16 KHz main clock						

All bits of AUDCTL are normally zero. The BASIC sound command causes it to be reset to zero. By clocking one channel with another, the range can be increased. This essentially allows

two channels with twice the range as each of the four normal channels. This is called 16 bit sound. To calculate exact frequencies, use the following formulas. The exact clock frequencies are also given if more accuracy is needed. The clock frequencies are acquired by dividing the signal from the TV color-burst crystal. This crystal has a frequency of 3.579545 MHz.

Clock frequencies:

```
1.7897725 MHz (color-burst/2)
63.920446 Khz (color-burst/56)
15.699759 KHz (color-burst/228)
```

Formulas:

For 1.79 MHz

Algorithm 1 16 bit sound

$$f = \frac{clock}{2(AUDFn+7)}$$

Algorithm 2 8 bit sound

$$f = \frac{clock}{2(AUDFn+4)}$$

AUDFn is the number in the audio frequency register.

For 16 KHz and 64 KHz

Algorithm 3 16 KHz and 64 KHz Sound

$$f = \frac{clock}{2(AUDFn+1)}$$

17.2.1. AUDIO TIMER INTERRUPTS

When the audio timers count down to zero they generate IRQ interrupts (if enabled). The timers can be reset by writing any number to STIMER [D209 (53769)].

17.3. THE CONSOLE SPEAKER

The console speaker is where key clicks and the cassette signals come from. On XL and XE models this speaker is heard through the TV speaker. It is operated by toggling bit 3 of CONSOL [SD01F (53279)]. This bit always reads 0 but it is actually set to 1 during vertical blank.

17. SOUND

Useful data base variables and OS equates CONSOL \$D01F (53279): bit 3 controls console speaker

AUDF1 \$D200 (53760): Audio frequency 1

AUDC1 \$D201 (53761): audio control 1

AUDF2 \$D202 (53762): Audio frequency 2

AUDC2 \$D203 (53763): audio control 2

AUDF3 \$D204 (53764): Audio frequency 3

AUDC3 \$D205 (53765): audio control 3

AUDF4 \$D206 (53766): Audio frequency 4

AUDC4 \$D207 (53767): audio control 4

AUDCTL \$D208 (53768): general audio control

STIMER \$D209 (53769): audio timer reset

18. THE JOYSTICK PORTS

The joystick ports are the I/O ports of the PIA chip. This means that they are bidirectional, capable of output as well as input. The joystick ports are usually set up for input. To read them, simply read the port registers. PORTA [\$D300 (53016)] will read joystick ports 1 and 2. PORTB [\$D301 (54017)] will read joystick ports 3 and 4. Joystick ports 3 and 4 are used for memory control on the XL/XE models and don't have external connectors.

Each bit of each port can be configured independently for input or output. To reconfigure a port, the port control registers, PACTL and PBCTL [\$D302 (54018) and \$D303 (54019)], are used. The port control registers also control some lines on the serial I/O connector.

18.1. The port control registers

```

          7 6 5 4 3 2 1 0
PACTL  -----
or      |n 0 1 1 n n 0 n|
PBCTL  -----
        1 6 3 1 8 4 2 1
        2 4 2 6
        8
        bits
```

PACTL 7 Peripheral A interrupt status. Set by peripheral interrupt; reset by reading PORTA.

3 Cassette motor control (0 = on: 1 = off).

2 0 = PORTA is now port A direction control. Writing to PORTA will now set bits for input or output. 0 sets bit for input; 1 sets bit for output.

1 = PORTA operational

1 1 = peripheral A interrupt enabled.

PBCTL 7 Peripheral B interrupt status. Set by peripheral interrupt; reset by reading PORTB.

3 Serial connector command line.

2 0 = PORTB is now port B direction control. Writing to PORTB will now set bits for input or output. 0 sets bit for input; 1 sets bit for output.

18. THE JOYSTICK PORTS

1 = PORTB operational

1 1 = peripheral B interrupt enabled.

The electronic configuration of the controller ports is as follows.

```

-----
\0 1 2 3 R/ \4 5 6 7 R/
 \t + - L/  \t + - L/
-----

```

0 through 7 are the binary data bits for port A or port B. + and - are +5 volts and ground respectively. R and L are the left and right game paddles. t is the joystick trigger line. The data bits in the joystick ports are used as follows for the joysticks and game paddles.

The joysticks and the port registers

```

          7 6 5 4 3 2 1 0
          -----
PORTA |U|D|L|R|U|D|L|R|
          -----
          1 6 3 1 8 4 2 1
          2 4 2 6
          8
paddle | | | |
triggers 3 2 1 0
PORTB -----
(400/800 |U|D|L|R|U|D|L|R|
only) -----
paddle | | | |
triggers 7 6 5 4
U = up
D = down
L = left
R = right

```

The joysticks may be read either directly from the port registers or from the joystick shadow registers. During vertical blank, the data in the port registers is separated and put into the shadow registers. These registers are, STICK0 [\$0278 (632)], STICK1 [\$0279 (633)], STICK2 [\$027A (634)] and STICK3 [\$027B (635)]. The triggers may be read from the joystick trigger registers, TRIG0 - TRIG3 [\$D010 - \$D013 (53264 - 53267)]. These register have shadow registers, STRIG0 - STRIG3 [\$0284 - 0287 (644 -647)]. If these registers read zero the associated triggers are pressed. The paddle triggers may be read from their shadow registers also. They are, PTRIG0 - PTRIG 7, [\$027C - \$0283 (236 - 643)].

18.2. THE GAME PADDLE REGISTERS

Although the game paddles are plugged into the joystick ports, they are not read from the port registers. The game paddles are read by first writing any number to the start-pot-scan register, POTGO [\$D20B (53771)]. This turns off the capacitor dump transistors and allows the pot reading capacitors to begin charging. It also sets the TV scan line counter to zero. As each capacitor crosses a certain trigger voltage, the number of TV lines scanned is put in the respective pot value register. When the scan counter reaches 228, the capacitor dump transistors are turned on and the number 228 is put into any pot value registers which are still empty.

Before reading the pot value registers, ALLPOT [\$D208 (53768)] should be checked. In this register, each bit corresponds to the validity of a pot value register. If a bit is zero, its' associated pot value register is valid. If bit 2 of SKCTL, [\$D20F (53775)], is 1, the pots go into the fast scan mode. In this mode the paddles are read in only 2 TV scan lines. They can also be read without regard to POTGO or ALLPOT.

The pot value registers contain the number of TV scan lines it last took for the paddle reading capacitors to charge (up to 228). These registers are POT0 - POT7 [\$D200 - \$D207 (53760 - 53767)]. Their shadow registers are PADDL0 - PADDL7 [\$0270 - \$0277 (624 - 631)].

18.3. THE LIGHT PEN REGISTERS

Whenever a joystick trigger is pressed, the light pen registers, PENH and PENV are updated. PENH [\$D40C (54284)] takes a value based on a color clock counter. The value can be from 0 to 227. PENV [\$D40D (54285)] takes the 8 highest bits of the vertical line counter. A light pen is simply a photo transistor connected to a joystick trigger line and focused on the TV screen. When the electron beam strikes the part of the screen the light pen is focused on, the transistor turns on pulling the trigger line low. The light pen registers then contain numbers relative to where the light pen was pointing. The shadow register for PENH and PENV are LPENH [\$0234 (564)] and LPENV [\$0235 (566)].

Useful operating system equates TRIG0 \$D010 (53264): joystick triggers

|

TRIG3 \$D013 (53268):

POT0 \$D200 (53760): paddle value

|

POT7 \$D207 (53767):

ALLPOT \$D208 (53768): reads validity of pot values

POTGO \$D20B (53771): starts paddle read

SKCTL \$D20F (53775): bit 2 enables fast pot scan

18. THE JOYSTICK PORTS

PORTA \$D300 (53016): port A data

PORTB \$D301 (53017): port B data

PACTL \$D302 (54018): port A control

PBCTL \$D303 (54019): port B control

PENH \$D40C (54284): light pen horizontal value

PENV \$D40D (54285): light pen vertical value

Shadow registers

LPENH \$0234 (564): light pen horizontal value

LPENV \$0235 (566): light pen vertical value

PADDL0 \$0270 (624): game paddle values

|

PADDL7 \$0277 (631)

STICK0 \$0278 (632): joystick registers

|

STICK0 \$027B (635):

PRTIG0 \$027C (636): paddle triggers

|

PTRIG7 \$0283 (643):

STRIG0 \$0284 (644): joystick triggers

|

STRIG3 \$0287 (647):

19. MISC HARDWARE REGISTERS AND INFORMATION

19.1. VERTICAL LINE COUNTER

The ANTIC chip has a vertical line counter at \$0D4B (54283). This counter shows the high 8 bits of a 9 bit counter. This gives two line resolution. The value of this counter is placed into PENV [\$D40D (54285)] when a joystick trigger is pressed.

19.2. SERIAL PORT REGISTERS

The POKEY chip has some registers which control the serial port. The serial port control register, SKCTL [\$D20F (53775)], controls the serial port configuration and the game paddle scan mode. and some keyboard circuitry.

The serial port control register

```
      7 6 5 4 3 2 1 0
      -----
SKCTL | | | | | | | |
      -----
      1 6 3 1 8 4 2 1
      2 4 2 6
      8
```

bits

0 1 = enable keyboard debounce

1 1 = enable keyboard scan

both 0 = set initialization mode.

2 1 = fast pot scan

3 1 = serial output is two tone (for cassette)
instead of logical true/false

4\

5 >- serial port mode control

6/

7 1 = forced logical 0 on output

19. MISC HARDWARE REGISTERS AND INFORMATION

If the serial port control register is read from it gives the serial port status. The register is then called SKSTAT

Serial port status register

```
7 6 5 4 3 2 1 0
-----
| | | | | | | |1|
-----
```

```
1 6 3 1 8 4 2 1
```

```
2 4 2 6
```

```
8
```

bits

0 not used, reads 1

1 0 = serial input shift register busy

2 0 = last key is still pressed

3 0 = shift key pressed

4 0 = direct from serial input port

5 0 = keyboard over-run

6 0 = serial data input over-run

7 1 = serial data input frame error

The serial port status is latched and must be reset by writing any number to its' reset register, SKRES [\$D20A (53770)].

19.3. SERIAL PORT INPUT AND OUTPUT DATA

When a full byte of serial input data has been received, it is read from the serial input data register, SERIN [\$D20D (53773)]. Serial output data is written to the same register, which is then called the serial output data register, SEROUT. This register is usually written to in response to a serial output data interrupt (bit 4 of IRQST).

19.4. HARDWARE CHIP MEMORY ALLOCATION

The addresses for the hardware chips are not completely decoded. For example, the PIA needs only four bytes of memory but is active from \$D300 - D3FF. Enough room for 64 PIA chips. A second pair of parallel ports could be added by accessing the address bus and further decoding the address for a second PIA. (This would also require a small modification of the computer's circuit board to disable the original PIA when the new one is active.) Similarly, there is room for 15 more POKEY or ANTIC chips and 7 gtia chips, should you ever need them. (GTIA uses \$D000 - D0FF, POKEY uses \$D200 - \$D2FF and ANTIC uses \$D400 - \$D4FF.)

19.4. HARDWARE CHIP MEMORY ALLOCATION

Useful data base variables and OS equates SKRES \$D20A (53770): serial port status reset

SEROUT \$D20D (53773): serial output data

SERIN \$D20D (53773): serial input data

SKCTL \$D20F (53775): serial port control

SKSTAT \$D20F (53775): serial port status

VCOUNT \$D40B (54283): vertical line counter

Os shadow registers

SSKCTL \$0232 (562): SKCTL

19. MISC HARDWARE REGISTERS AND INFORMATION

20. THE XL AND XE MODELS

20.1. BASIC B BUGS

Most of the Atari 600XL and 800XL models were supplied with the "debugged" version B of Atari BASIC. This new BASIC got rid of the minor bugs of BASIC A and introduced some new major bugs of it's own. Each time a program is saved, 16 extra bytes are tagged onto the end of the program. After many saves and reloads, as when developing a long program, the program becomes too large for the memory.

The computer may lock up unpredictably. Program line links may get messed up, leaving garbage in the listing and the program unrunable. Large LISTed programs may not run unless SAVed and reLOADed. If the length of a listed program is a multiple of a certain number of bytes, it will not run unless the length is somehow changed. BASIC version B has been replaced by version C. All of the XE models have this truly debugged version of BASIC.

20.2. NEW OPERATING SYSTEM PROBLEMS

I have heard of only one bug in the operating system in XL and XE models. This is a mishandling of the printer timeout. The computer cannot tell if there is a printer attached or not. This may have been fixed in the XE models. However, many programs, some even formerly sold by Atari, do not jump through published jump vectors when using the operating system. These programs will not run on XL/XE models. (Some of these programs are Atari Word Processor (not Atariwriter) and LJKs Letter Perfect and Data Perfect.) Since the operating system ROM can be switched to RAM, a "translator" can be used to load the 800 operating system into an XL or XE model.

20.3. 130XE MEMORY MANAGEMENT

The 130XE has an extra 64K bank of memory. It is divided into four blocks of 16K each. Each block can be switched to replace part of the main bank of RAM from \$4000 (16384) to \$7FFF (32767). Furthermore, it can be switched in such a way that only the 6502, or the ANTIC chip can see the extra memory.

Port B (formerly the two extra joystick ports of the 400/800) is used to manage the memory.

Port B and memory management

20. THE XL AND XE MODELS

```
7 6 5 4 3 2 1 0
-----
PORTB |T|U|A|C|S S|B|R|
-----
1 6 3 1 8 4 2 1
2 4 2 6
8
R 1 = OS replaced by RAM
B 0 = BASIC enabled
S S bank select bits
C 0 = CPU sees switched RAM at $4000
A 0 = ANTIC sees switched RAM
U unused
T 0 = self test
```

Bits 2 and 3 of PORTB select which block of the extra bank of memory is switched in.

```
Bank select bits
bits block
2 3 address
-----
0 0 $0000 - $3FFF
0 1 $4000 - $7FFF
1 0 $8000 - $BFFF
1 1 $C000 - $FFFF
Bits 4 and 5 select which chip sees the switched in RAM at $4000 -
$7FFF
Chip select bits
bits ANTIC 6502
4 5
-----
0 0 Ext. Ext.
0 1 Ext. Main
1 0 Main Ext.
1 1 Main Main
```

20.4. THE XL PARALLEL PORT

Pin out of the parallel port top from rear

```
111112222233333444445
2468024680246802468024680
```



```

-----
-----
11111222223333344444
1357913579135791357913579
1 2 GND
3 A1 4 A0
5 A3 6 A2
7 A5 8 A4
9 GND 10 A6
11 A8 12 A7
13 A10 14 A9
15 A12 16 A11
17 A14 18 A13
19 A15 20 GND
21 D1 22 D0
23 D3 24 D2
25 D5 26 D4
27 D7 28 D6
29 GND 30 GND
31 GND 32 phase 2 clock
33 RESET 34
35 RDY 36 IRQ
37 37
39 40
41 GND 42
43 RAS 44
45 R/W 46 GND
47 +5V 48 +5V
49 GND 50

```

The phase 2 clock runs at 1.8 MHz. When the clock is high, the address and R/W lines are valid. The clock goes from high to low, when the data lines are also valid. All lines then become invalid. The 130XE doesn't have the parallel port. However, it has a cartridge slot expansion. This is a small cartridge-slot-like connector with the necessary connector to use parallel expansion.

20.5. FINE SCROLLING

If address \$026E (622) is \$FF, graphics 0 will be in the fine scroll mode.

20.6. OTHER ADDRESSES

DSCTLN [\$0D25,2 (725)] is the disk sector size. should be \$80 (128).

DMASAV [\$02DD (735)] is a copy of the DMA control register, SDMCTL [\$022F (559)]. It is set up when a channel is opened to the screen. The value is moved to SDMCTL whenever a key is pressed. It is used to restore the display if DMA is disabled.

PUPBT [\$033D,3 (829-831)] is used to test memory integrity when [RESET] is pressed. If these bytes are not \$5C, \$93 and \$25, the computer will do a cold start when [RESET] is pressed.

The self-test ROM is from \$D000 to \$D7FF, the same addresses as the hardware registers. This part of the operating system ROM is disabled when not used. When The computer is put into the self-test mode, This part of ROM is copied to \$5000 to \$57FF and run from there.

GINTLK [\$03FA (1018)] is a logical 1 if a cartridge is installed (built-in BASIC is considered a cartridge). BASIC can be disabled by poking 1018 with a non-zero number. If [RESET] is then pressed, the computer will attempt to load the DUP.SYS file and basic will be completely disabled.

A. HARDWARE REGISTERS

Register Shadow

Name Description Address Name Address

ALLPOT game paddle ready indicators \$D208 53768
AUDC1 Audio channel 1 control \$D201 53761
AUDC2 Audio channel 2 control \$D203 53763
AUDC3 Audio channel 3 control \$D205 53765
AUDC4 Audio channel 1 control \$D207 53767
AUDCTL general audio control \$D208 53768
AUDF1 Audio frequency 1 control \$D200 53760
AUDF2 Audio frequency 2 control \$D202 53762
AUDF3 Audio frequency 3 control \$D204 53764
AUDF4 Audio frequency 4 control \$D206 53766
CHACTL character control \$D401 54273 CHART \$02F3 755
CHBASE Address of character set / 256 \$D409 54281 CHBAS \$02F4 756
COLBK color/brightness of setcolor 4 \$D01A 53274 COLOR4 \$02C8 712
COLPF0 Color/brightness of setcolor 0 \$D016 53270 COLOR0 \$02C4 708
COLPF1 color/brightness of setcolor 1 \$D017 53271 COLOR1 \$02C5 709
COLPF2 color/brightness of setcolor 2 \$D018 53272 COLOR2 \$02C6 710
COLPF3 color/brightness of setcolor 3 \$D019 53273 COLOR3 \$02C7 711
COLPM0 color/brightness, player/missile 0 \$D012 53266 PCOLR0 \$02C0 704
COLPM1 color/brightness, player/missile 1 \$D013 53267 PCOLR1 \$02C1 705
COLPM2 color/brightness, player/missile 2 \$D014 53268 PCOLR2 \$02C2 706
COLPM3 color/brightness, player/missile 3 \$D015 53269 PCOLR3 \$02C3 707
CONSOL [START], [SELECT], [OPT.], speaker \$D01F 53279

A. *HARDWARE REGISTERS*

DLISTH display list pointer high byte \$D403 54275 SDLSTH \$0231 561
DLISTL display list pointer low byte \$D402 54274 SDLSTL \$0230 560
DMACTL Direct Memory access control (DMA) \$D400 54272 SDMCTL \$022F 559
GRACTL graphics control \$D01D 53277
GRAFM missile graphics \$D011 53265
GRAFP0 player 0 graphics \$D00D 53261
GRAFP1 player 1 graphics \$D00E 53262
GRAFP2 player 2 graphics \$D00F 53263
GRAFP3 player 3 graphics \$D010 53264
HITCLR clear collisions \$D01E 54278
HPOSM0 horizontal position of missile 0 \$D004 53252
HPOSM1 horizontal position of missile 1 \$D005 53253
HPOSM2 horizontal position of missile 2 \$D006 53254
HOPSM3 horizontal position of missile 3 \$D007 53255
HPOSP0 horizontal position of player 0 \$D000 53248
HPOSP1 horizontal position of player 1 \$D001 53249
HPOSP2 horizontal position of player 2 \$D002 53250
HPOSP3 horizontal position of player 3 \$D003 53251
HSCROL horizontal scroll \$D404 54276
IRQEN interrupt request enable (IRQ) \$D20E 53774 POKMSK \$0010 16
IRQST IRQ status \$D20E 53774
KBCODE keyboard code \$D209 53769 CH \$02FC 764
M0PF missile 0 to graphics collisions \$D000 53248
M0PL missile 0 to player collisions \$D008 53256
M1PF missile 1 to graphics collisions \$D001 53249
M1PL missile 1 to player collisions \$D009 53257
M2PF missile 2 to graphics collisions \$D002 53250
M2PL missile 2 to player collisions \$D00A 53258
M3PF missile 3 to graphics collisions \$D003 53251
M3PL missile 3 to player collisions \$D00B 53259
NMIEN non-maskable interrupt enable (NMI)\$D40E 54286

NMIRES NMI reset \$D40F 54287
NMIST NMI status \$D40F 54287
POPF player 0 to graphics collisions \$D004 53252
POPL player 0 to player collisions \$D00C 53260
P1PF player 1 to graphics collisions \$D005 53253
P1PL player 1 to player collisions \$D00D 53261
P2PF player 2 to graphics collisions \$D006 53254
P2PL player 2 to player collisions \$D00E 53262
P3PF player 3 to graphics collisions \$D007 53255
P3PL player 3 to player collisions \$D00F 53263
PACTL port A control \$D302 54018
PAL Europe/North America TV indicator \$D014 53268
PBCLT port B control \$D303 54019
PENH light pen horizontal position \$D40C 54284 LPENH \$0234 564
PENV light pen vertical position \$D40D 54285 LPENV \$0235 565
PMBASE player/missile address / 256 \$D407 54279
PORTA port A \$D300 54016 STICK0 \$0278 632
STICK1 \$0279 634
PORTB port B \$D301 54017 STICK2 \$027A 634
STICK3 \$027B 635
POT0 game paddle 0 \$D200 53760 PADDL0 \$0270 624
POT1 game paddle 1 \$D201 53761 PADDL1 \$0271 625
POT2 game paddle 2 \$D202 53762 PADDL2 \$0272 626
POT3 game paddle 3 \$D203 53763 PADDL3 \$0273 627
POT4 game paddle 4 \$D204 53764 PADDL4 \$0274 628
POT5 game paddle 5 \$D205 53765 PADDL5 \$0275 629
POT6 game paddle 6 \$D206 53766 PADDL6 \$0276 630
POT7 game paddle 7 \$D207 53767 PADDL7 \$0277 631
POTGO start pot scan sequence \$D20B 53771
PRIOR p/m priority and GTIA mode \$D21B 53275 GPRIOR \$026F 623
RANDOM random number generator \$D20A 53770

A. HARDWARE REGISTERS

SERIN serial port input \$D20D 53774
SEROUT serial port output \$D20D 53773
SIZEM missile size \$D00C 53260
SIZEP0 player 0 size \$D008 53256
SIZEP1 player 1 size \$D009 53257
SIZEP2 player 2 size \$D00A 53258
SIZEP3 player 3 size \$D00B 53259
SKCTL serial port control \$D20F 53775 SSKCTL \$0232 563
SKREST reset serial port status \$D20A 53770
SKSTAT serial port status \$D20F 53775
STIMER start timer \$D209 53769
TRIG0 joystick trigger 0 \$D010 53264 STRIG0 \$0284 644
TRIG1 joystick trigger 1 \$D011 53265 STRIG1 \$0285 645
TRIG2 joystick trigger 2 \$D012 53266 STRIG2 \$0286 646
TRIG3 joystick trigger 3 \$D013 53267 STRIG3 \$0287 647
VCOUNT vertical line counter \$D40B 54283
VDELAY vertical delay \$D01C 54276
VSCROL vertical scroll \$D405 54277
WSYNC wait for horizontal sync \$D40A 54282

NUMERICAL ORDER

Registers sharing addresses are listed first when written to, then when read from

Register Shadow

Name Description Address Name Address

HPOSP0 horizontal position of player 0 \$D000 53248
M0PF missile 0 to graphics collisions \$D000 53248
HPOSP1 horizontal position of player 1 \$D001 53249
M1PF missile 1 to graphics collisions \$D001 53249
HPOSP2 horizontal position of player 2 \$D002 53250
M2PF missile 2 to graphics collisions \$D002 53250

HPOSP3 horizontal position of player 3 \$D003 53251
M3PF missile 3 to graphics collisions \$D003 53251
HPOSM0 horizontal position of missile 0 \$D004 53252
P0PF player 0 to graphics collisions \$D004 53252
HPOSM1 horizontal position of missile 1 \$D005 53253
P1PF player 1 to graphics collisions \$D005 53253
HPOSM2 horizontal position of missile 2 \$D006 53254
P2PF player 2 to graphics collisions \$D006 53254
HOPSM3 horizontal position of missile 3 \$D007 53255
P3PF player 3 to graphics collisions \$D007 53255
SIZEP0 player 0 size \$D008 53256
M0PL missile 0 to player collisions \$D008 53256
SIZEP1 player 1 size \$D009 53257
M1PL missile 1 to player collisions \$D009 53257
SIZEP2 player 2 size \$D00A 53258
M2PL missile 2 to player collisions \$D00A 53258
SIZEP3 player 3 size \$D00B 53259
M3PL missile 3 to player collisions \$D00B 53259
SIZEM missile size \$D00C 53260
P0PL player 0 to player collisions \$D00C 53260
GRAFP0 player 0 graphics \$D00D 53261
P1PL player 1 to player collisions \$D00D 53261
GRAFP1 player 1 graphics \$D00E 53262
P2PL player 2 to player collisions \$D00E 53262
GRAFP2 player 2 graphics \$D00F 53263
P3PL player 3 to player collisions \$D00F 53263
GRAFP3 player 3 graphics \$D010 53264
TRIG0 joystick trigger 0 \$D010 53264 STRIG0 \$0284 644
GRAFM missile graphics \$D011 53265
TRIG1 joystick trigger 1 \$D011 53265 STRIG1 \$0285 645
COLPM0 color/brightness, player/missile 0 \$D012 53266 PCOLR0 \$02C0 704

A. *HARDWARE REGISTERS*

TRIG2 joystick trigger 2 \$D012 53266 STRIG2 \$0286 646
COLPM1 color/brightness, player/missile 1 \$D013 53267 PCOLR1 \$02C1 705
TRIG3 joystick trigger 3 \$D013 53267 STRIG3 \$0287 647
COLPM2 color/brightness, player/missile 2 \$D014 53268 PCOLR2 \$02C2 706
PAL Europe/North America TV indicator \$D014 53268
COLPM3 color/brightness, player/missile 3 \$D015 53269 PCOLR3 \$02C3 707
COLPF0 Color/brightness of setcolor 0 \$D016 53270 COLOR0 \$02C4 708
COLPF1 color/brightness of setcolor 1 \$D017 53271 COLOR1 \$02C5 709
COLPF2 color/brightness of setcolor 2 \$D018 53272 COLOR2 \$02C6 710
COLPF3 color/brightness of setcolor 3 \$D019 53273 COLOR3 \$02C7 711
COLBK color/brightness of setcolor 4 \$D01A 53274 COLOR4 \$02C8 712
VDELAY vertical delay \$D01C 54276
GRCTL graphics control \$D01D 53277
HITCLR clear collisions \$D01E 54278
CONSOL [START], [SELECT], [OPT.], speaker \$D01F 53279
AUDF1 Audio frequency 1 control \$D200 53760
POT0 game paddle 0 \$D200 53760 PADDL0 \$0270 624
AUDC1 Audio channel 1 control \$D201 53761
POT1 game paddle 1 \$D201 53761 PADDL1 \$0271 625
AUDF2 Audio frequency 2 control \$D202 53762
POT2 game paddle 2 \$D202 53762 PADDL2 \$0272 626
AUDC2 Audio channel 2 control \$D203 53763
POT3 game paddle 3 \$D203 53763 PADDL3 \$0273 627
AUDF3 Audio frequency 3 control \$D204 53764
POT4 game paddle 4 \$D204 53764 PADDL4 \$0274 628
AUDC3 Audio channel 3 control \$D205 53765
POT5 game paddle 5 \$D205 53765 PADDL5 \$0275 629
AUDF4 Audio frequency 4 control \$D206 53766
POT6 game paddle 6 \$D206 53766 PADDL6 \$0276 630
AUDC4 Audio channel 1 control \$D207 53767
POT7 game paddle 7 \$D207 53767 PADDL7 \$0277 631

ALLPOT game paddle ready indicators \$D208 53768
AUDCTL general audio control \$D208 53768
KBCODE keyboard code \$D209 53769 CH \$O2FC 764
STIMER start timer \$D209 53769
RANDOM random number generator \$D20A 53770
SKREST reset serial port status \$D20A 53770
POTGO start pot scan sequence \$D20B 53771
SEROUT serial port output \$D20D 53773
SERIN serial port input \$D20D 53774
IRQEN interrupt request enable (IRQ) \$D20E 53774 POKMSK \$0010 16
IRQST IRQ status \$D20E 53774
SKCTL serial port control \$D20F 53775 SSKCTL \$0232 563
SKSTAT serial port status \$D20F 53775
PRIOR p/m priority and GTIA mode \$D21B 53275 GPRIOR \$026F 623
PORTA port A \$D300 54016 STICK0 \$0278 632
STICK1 \$0279 633
PORTB port B \$D301 54017 STICK2 \$027A 634
STICK3 \$027B 635
PACTL port A control \$D302 54018
PBCTL port B control \$D303 54019
DMACTL Direct Memory access control (DMA) \$D400 54272 SDMCTL \$022F 559
CHACTL character control \$D401 54273 CHART \$02F3 755
DLISTL display list pointer low byte \$D402 54274 SDLSTL \$0230 560
DLISTH display list pointer high byte \$D403 54275 SDLSTH \$0231 561
HSCROL horizontal scroll \$D404 54276
VSCROL vertical scroll \$D405 54277
PMBASE player/missile address / 256 \$D407 54279
CHBASE Address of character set / 256 \$D409 54281 CHBAS \$02F4 756
WSYNC wait for horizontal sync \$D40A 54282
VCOUNT vertical line counter \$D40B 54283
PENH light pen horizontal position \$D40C 54284 LPENH \$0234 564

A. HARDWARE REGISTERS

PENV light pen vertical position \$D40D 54285 LPENV \$0235 565

NMIEN non-maskable interrupt enable (NMI)\$D40E 54286

NMIRES NMI reset \$D40F 54287

NMIST NMI status \$D40F 54287

SHADOW REGISTER ORDER

ALPHEBETICAL ORDER

Register Shadow

Name Description Address Name Address

KBCODE keyboard code \$D209 53769 CH \$02FC 764

CHACTL character control \$D401 54273 CHART \$02F3 755

CHBASE Address of character set / 256 \$D409 54281 CHBAS \$02F4 756

COLBK color/brightness of setcolor 4 \$D01A 53274 COLOR4 \$02C8 712

COLPF0 Color/brightness of setcolor 0 \$D016 53270 COLOR0 \$02C4 708

COLPF1 color/brightness of setcolor 1 \$D017 53271 COLOR1 \$02C5 709

COLPF2 color/brightness of setcolor 2 \$D018 53272 COLOR2 \$02C6 710

COLPF3 color/brightness of setcolor 3 \$D019 53273 COLOR3 \$02C7 711

PRIOR p/m priority and GTIA mode \$D21B 53275 GPRIOR \$026F 623

PENH light pen horizontal position \$D40C 54284 LPENH \$0234 564

PENV light pen vertical position \$D40D 54285 LPENV \$0235 565

POT0 game paddle 0 \$D200 53760 PADDL0 \$0270 624

POT1 game paddle 1 \$D201 53761 PADDL1 \$0271 625

POT2 game paddle 2 \$D202 53762 PADDL2 \$0272 626

POT3 game paddle 3 \$D203 53763 PADDL3 \$0273 627

POT4 game paddle 4 \$D204 53764 PADDL4 \$0274 628

POT5 game paddle 5 \$D205 53765 PADDL5 \$0275 629

POT6 game paddle 6 \$D206 53766 PADDL6 \$0276 630

POT7 game paddle 7 \$D207 53767 PADDL7 \$0277 631

COLPM0 color/brightness, player/missile 0 \$D012 53266 PCOLR0 \$02C0 704

COLPM1 color/brightness, player/missile 1 \$D013 53267 PCOLR1 \$02C1 705

COLPM2 color/brightness, player/missile 2 \$D014 53268 PCOLR2 \$02C2 706

COLPM3 color/brightness, player/missile 3 \$D015 53269 PCOLR3 \$02C3 707

IRQEN interrupt request enable (IRQ) \$D20E 53774 POKMSK \$0010 16

DLISTH display list pointer high byte \$D403 54275 SDLSTH \$0231 561

DLISTL display list pointer low byte \$D402 54274 SDLSTL \$0230 560

DMACTL Direct Memory access control (DMA) \$D400 54272 SDMCTL \$022F 559

SKCTL serial port control \$D20F 53775 SSKCTL \$0232 563

PORTA port A \$D300 54016 STICK0 \$0278 632

STICK1 \$0279 633

PORTB port B \$D301 54017 STICK2 \$027A 634

STICK3 \$027B 635

TRIG0 joystick trigger 0 \$D010 53264 STRIG0 \$0284 644

TRIG1 joystick trigger 1 \$D011 53265 STRIG1 \$0285 645

TRIG2 joystick trigger 2 \$D012 53266 STRIG2 \$0286 646

TRIG3 joystick trigger 3 \$D013 53267 STRIG3 \$0287 647

NUMERICAL ORDER

IRQEN interrupt request enable (IRQ) \$D20E 53774 POKMSK \$0010 16

DMACTL Direct Memory access control (DMA) \$D400 54272 SDMCTL \$022F 559

DLISTL display list pointer low byte \$D402 54274 SDLSTL \$0230 560

DLISTH display list pointer high byte \$D403 54275 SDLSTH \$0231 561

SKCTL serial port control \$D20F 53775 SSKCTL \$0232 563

PENH light pen horizontal position \$D40C 54284 LPENH \$0234 564

PENV light pen vertical position \$D40D 54285 LPENV \$0235 565

PRIOR p/m priority and GTIA mode \$D21B 53275 GPRIOR \$026F 623

POT0 game paddle 0 \$D200 53760 PADDL0 \$0270 624

POT1 game paddle 1 \$D201 53761 PADDL1 \$0271 625

POT2 game paddle 2 \$D202 53762 PADDL2 \$0272 626

POT3 game paddle 3 \$D203 53763 PADDL3 \$0273 627

POT4 game paddle 4 \$D204 53764 PADDL4 \$0274 628

POT5 game paddle 5 \$D205 53765 PADDL5 \$0275 629

POT6 game paddle 6 \$D206 53766 PADDL6 \$0276 630

POT7 game paddle 7 \$D207 53767 PADDL7 \$0277 631

A. *HARDWARE REGISTERS*

PORTA port A \$D300 54016 STICK0 \$0278 632

STICK1 \$0279 633

PORTB port B \$D301 54017 STICK2 \$027A 634

STICK3 \$027B 635

TRIG0 joystick trigger 0 \$D010 53264 STRIG0 \$0284 644

TRIG1 joystick trigger 1 \$D011 53265 STRIG1 \$0285 645

TRIG2 joystick trigger 2 \$D012 53266 STRIG2 \$0286 646

TRIG3 joystick trigger 3 \$D013 53267 STRIG3 \$0287 647

COLPM0 color/brightness, player/missile 0 \$D012 53266 PCOLR0 \$02C0 704

COLPM1 color/brightness, player/missile 1 \$D013 53267 PCOLR1 \$02C1 705

COLPM2 color/brightness, player/missile 2 \$D014 53268 PCOLR2 \$02C2 706

COLPM3 color/brightness, player/missile 3 \$D015 53269 PCOLR3 \$02C3 707

COLPF0 Color/brightness of setcolor 0 \$D016 53270 COLOR0 \$02C4 708

COLPF1 color/brightness of setcolor 1 \$D017 53271 COLOR1 \$02C5 709

COLPF2 color/brightness of setcolor 2 \$D018 53272 COLOR2 \$02C6 710

COLPF3 color/brightness of setcolor 3 \$D019 53273 COLOR3 \$02C7 711

COLBK color/brightness of setcolor 4 \$D01A 53274 COLOR4 \$02C8 712

CHACTL character control \$D401 54273 CHART \$02F3 755

CHBASE Address of character set / 256 \$D409 54281 CHBAS \$02F4 756

KBCODE keyboard code \$D209 53769 CH \$02FC 764

B. OPERATING SYSTEM EQUATES

```
0100 ;
0101 ; ATARI 800 EQUATE LISTING
0102 ;
0103 ;
0104 ;
0105 ;This listing is based on the original release of Operating System,
0106 ;version A. The vectors shown here were not changed in version B.
0107 ;New equates for XL and XE models are included and noted. Changes
0108 ;from version B to XL/XE are also noted.
0109 ;
0110 ;Most of the equate names given below are the official Atari
0111 ;names. They are in common use but are not mandatory.
0112 ;
0113 ;
0114 ; DEVICE NAMES
0115 ;
0116 ;
0117 ;SCREDT = "E" SCREEN EDITOR
0118 ;KBD = "K" KEYBOARD
0119 ;DISPLY = "S" DISPLAY
0120 ;PRINTR = "P" PRINTER
0121 ;CASSET = "C" CASSETTE
0122 ;DISK = "D" DISK DRIVE
0123 ;
0124 ;
0125 ;
0126 ; STATUS CODES
0127 ;
0128 ;
0129 SUCCES = $01 1
0130 BRKABT = $80 128 BREAK KEY ABORT
0131 PRVOPN = $82 130 IOCB ALREADY OPEN
0132 NONDEV = $82 130 NONEXISTANT DEVICE
0133 WRONLY = $83 131 OPENED FOR WRITE ONLY
0134 NVALID = $84 132 INVALID COMMAND
0135 NOTOPN = $85 133 DEVICE OR FILE NOT OPEN
```

B. OPERATING SYSTEM EQUATES

```
0136 BADIOC = $86 134 INVALID IOCB NUMBER
0137 RDONLY = $87 135 OPENED FOR READ ONLY
0138 EOFERR = $88 136 END OF FILE
0139 TRNRCD = $89 137 TRUNCATED RECORD
0140 TIMEOUT = $8A 138 PERIPHERAL TIME OUT
0141 DNACK = $8B 139 DEVICE DOES NOT ACKNOWLEDGE
0142 FRMERR = $8C 140 SERIAL BUS FRAMING ERROR
0143 CRSROR = $8D 141 CURSOR OUT OF RANGE
0144 OVRRUN = $8E 142 SERIAL BUS DATA OVERRUN
0145 CHKERR = $8F 143 SERIAL BUS CHECKSUM ERROR
0146 DERROR = $90 144 PERIPHERAL DEVICE ERROR
0147 BADMOD = $91 145 NON EXISTANT SCREEN MODE
0148 FNCNOT = $92 146 FUNCTION NOT IMPLEMENTED
0149 SCRMEM = $93 147 NOT ENOUGH MEMORY FOR SCREEN MODE
0150 ;
0151 ;
0152 ;
0153 ;
0154 ; COMMAND CODES FOR CIO
0155 ;
0156 ;
0157 OPEN = $03 3
0158 OPREAD = $04 4 OPEN FOR INPUT
0159 GETREC = $05 5 GET RECORD
0160 OPDIR = $06 6 OPEN TO DISK DIRECTORY
0161 GETCHR = $07 7 GET BYTE
0162 OWRITE = $08 8 OPEN FOR OUTPUT
0163 PUTREC = $09 9 WRITE RECORD
0164 APPEND = $09 9 OPEN TO APPEND TO END OF DISK FILE
0165 MXDMOD = $10 16 OPEN TO SPLIT SCREEN (MIXED MODE)
0166 PUTCHR = $0B 11 PUT-BYTE
0167 CLOSE = $0C 12
0168 OUPDAT = $0C 12 OPEN FOR INPUT AND OUTPUT AT THE SAME TIME
0169 STATUS = $0D 13
0170 SPECIL = $0E 14 BEGINNING OF SPECIAL COMMANDS
0171 DRAWLN = $11 17 SCREEN DRAW
0172 FILLIN = $12 18 SCREEN FILL
0173 RENAME = $20 32
0174 INSLR = $20 32 OPEN TO SCREEN BUT DON'T ERASE
0175 DELETE = $21 33
0176 DFRMAT = $21 33 FORMAT DISK (RESIDENT DISK HANDLER (RDH))
0177 LOCK = $23 35
0178 UNLOCK = $24 36
0179 POINT = $25 37
```

```

0180 NOTE = $26 38
0181 PTSECT = $50 80 RDH PUT SECTOR
0182 GTSECT = $52 82 RDH GET SECTOR
0183 DSTAT = $53 83 RDH GET STATUS
0184 PSECTV = $57 87 RDH PUT SECTOR AND VERIFY
0185 NOIRG = $80 128 NO GAP CASSETTE MODE
0186 CR = $9B 155 CARRIAGE RETURN (EOL)
0187 ;
0188 IOCBSZ = $10 16 IOCB SIZE
0189 MAXIOC = $80 128 MAX IOCB BLOCK SIZE
0190 IOCBF = $FF 255 IOCB FREE
0191 ;
0192 LEDGE = $02 2 DEFAULT LEFT MARGIN
0193 REDGE = $27 39 DEFAULT RIGHT MARGIN
0194 ;
0195 ; OS VARIABLES
0196 ;
0197 ; PAGE 0
0198 ;
0199 LINZBS = $00 0 (800) FOR ORIGINAL DEBUGGER
0200 ; $00 0 (XL) RESERVED
0201 NGFLAG = $01 1 (XL) FOR POWER-UP SELF TEST
0202 CASINI = $02 2
0203 RAMLO = $04 4 POINTER FOR SELF TEST
0204 TRAMSZ = $06 6 TEMPORARY RAM SIZE
0205 TSTDAT = $07 7 TEST DATA
0206 WARMST = $08 8
0207 BOOT? = $09 9 SUCCESSFUL BOOT FLAG
0208 DOSVEC = $0A 10 PROGRAM RUN VECTOR
0209 DOSINI = $0C 12 PROGRAM INITIALIZATION
0210 APPMHI = $0E 14 DISPLAY LOW LIMIT
0211 POKMSK = $10 16 IRQ ENABLE FLAGS
0212 BRKKEY = $11 17 FLAG
0213 RTCLOK = $12 18 3 BYTES, MSB FIRST
0214 BUFADR = $15 21 INDIRECT BUFFER ADDRESS
0215 ICCOMT = $17 23 COMMAND FOR VECTOR
0216 DSKFMS = $18 24 DISK FILE MANAGER POINTER
0217 DSKUTL = $1A 26 DISK UTILITY POINTER (DUP.SYS)
0218 PTIMOT = $1C 28 (800) PRINTER TIME OUT REGISTER
0219 ABUFPT = $1C 28 (XL) RESERVED
0220 PBPNT = $1D 29 (800) PRINTER BUFFER POINTER
0221 ; $1D 29 (XL) RESERVED
0222 PBUFSZ = $1E 30 (800) PRINTER BUFFER SIZE
0223 ; $1E 30 (XL) RESERVED

```

B. OPERATING SYSTEM EQUATES

```
0224 PTEMP = $1F 31 (800) TEMPORARY REGISTER
0225 ; $1F 31 (XL) RESERVED
0226 ZIOCB = $20 32 ZERO PAGE IOCB
0227 ICHIDZ = $20 32 HANDLER INDEX NUMBER (ID)
0228 ICDNOZ = $21 33 DEVICE NUMBER
0229 ICCOMZ = $22 34 COMMAND
0230 ICSTAZ = $23 35 STATUS
0231 ICBALZ = $24 36 BUFFER POINTER LOW BYTE
0232 ICBAHZ = $25 37 BUFFER POINTER HIGH BYTE
0233 ICPTLZ = $26 38 PUT ROUTINE POINTER LOW
0234 ICPTHZ = $27 39 PUT ROUTINE POINTER HIGH
0235 ICBL LZ = $28 40 BUFFER LENGTH LOW
0236 ICBLHZ = $29 41
0237 ICAX1Z = $2A 42 AUXILIARY INFORMATION BYTE 1
0238 ICAX2Z = $2B 43
0239 ICSPRZ = $2C 44 TWO SPARE BYTES (CIO USE)
0240 ICIDNO = $2E 46 IOCB NUMBER X 16
0241 CIOCHR = $2F 47 CHARACTER BYTE FOR CURRENT OPERATION
0242 ;
0243 STATUS = $30 48 STATUS STORAGE
0244 CHKSUM = $31 49 SUM WITH CARRY ADDED BACK
0245 BUFRLO = $32 50 DATA BUFFER LOW BYTE
0246 BUFRHI = $33 51
0247 BFENLO = $34 52 ADDRESS OF LAST BUFFER BYTE +1 (LOW)
0248 BFENHI = $35 53
0249 CRETRY = $36 54 (800) NUMBER OF COMMAND FRAME RETRIES
0250 LTEMP = $36 54 (XL) LOADER TEMPORARY STORAGE, 2 BYTES
0251 DRETRY = $37 55 (800) DEVICE RETRIES
0252 BUFRFL = $38 56 BUFFER FULL FLAG
0253 RECVDN = $39 57 RECEIVE DONE FLAG
0254 XMTDON = $3A 58 TRANSMISSION DONE FLAG
0255 CHKSNT = $3B 59 CHECKSUM-SENT FLAG
0256 NOCKSM = $3C 60 CHECKSUM-DOES-NOT-FOLLOW-DATA FLAG
0257 BPTR = $3D 61
0258 FTYPE = $3E 62
0259 FEOF = $3F 63
0260 FREQ = $40 64
0261 ;
0262 SOUND R = $41 65 0=QUIET I/O
0263 CRITIC = $42 66 CRITICAL FUNCTION FLAG, NO DEFERRED VBI
0264 FMSZPG = $43 67 DOS ZERO PAGE, 7 BYTES
0265 CKEY = $4A 74 (800) START KEY FLAG
0266 ZCHAIN = $4A 74 (XL) HANDLER LOADER TEMP, 2 BYTES
0267 CASSBT = $4B 75 (800) CASSETTE BOOT FLAG
```


0268 DSTAT = \$4C 76 DISPLAY STATUS
 0269 ;
 0270 ATRACT = \$4D 77
 0271 DRKMSK = \$4E 78 ATTRACT MASK
 0272 COLRSH = \$4F 79 ATTRACT COLOR SHIFTER (EORed WITH
 GRAPHICS)
 0273 ;
 0274 TMPCHR = \$50 80
 0275 HOLD1 = \$51 81
 0276 LMARGN = \$52 82 SCREEN LEFT MARGIN REGISTER
 0277 RMARGN = \$53 83 SCREEN RIGHT MARGIN
 0278 ROWCRS = \$54 84 CURSOR ROW
 0279 COLCRS = \$55 85 CURSOR COLUMN, 2 BYTES
 0280 DINDEX = \$57 87 DISPLAY MODE
 0281 SAVMSC = \$58 88 SCREEN ADDRESS
 0282 OLDROW = \$5A 90 CURSOR BEFORE DRAW OR FILL
 0283 OLDCOL = \$5B 91
 0284 OLDCHR = \$5D 93 DATA UNDER CURSOR
 0285 OLDADR = \$5E 94 CURSOR ADDRESS
 0286 NEWROW = \$60 96 (800) DRAWTO DESTINATION
 0287 FKDEF = \$60 96 (XL) FUNCTION KEY DEFINATION POINTER
 0288 NEWCOL = \$61 97 (800) DRAWTO DESTINATION, 2 BYTES
 0289 PALNTS = \$62 98 (XL) EUROPE/NORTH AMERICA TV FLAG
 0290 LOGCOL = \$63 99 LOGICAL LINE COLUMN POINTER
 0291 MLTTMP = \$66 102
 0292 OPNTMP = \$66 102 TEMPORARY STORAGE FOR CHANNEL OPEN
 0293 SAVADR = \$68 104
 0294 RAMTOP = \$6A 106 START OF ROM (END OF RAM + 1), HIGH BYTE
 ONLY
 0295 BUFCNT = \$6B 107 BUFFER COUNT
 0296 BUFSTR = \$6C 108 POINTER USED BY EDITOR
 0297 BITMSK = \$6E 110 POINTER USED BY EDITOR
 0298 SHFAMT = \$6F 111
 0299 ROWAC = \$70 112
 0300 COLAC = \$72 114
 0301 ENDPT = \$74 116
 0302 DELTAR = \$76 118
 0303 DELTAC = \$77 119
 0304 ROWINC = \$79 121 (800)
 0305 KEYDEF = \$79 121 (XL) KEY DEFINATION POINTER, 2 BYTES
 0306 COLINC = \$7A 122 (800)
 0307 SWPFLG = \$7B 123 NON 0 IF TEXT AND REGULAR RAM IS SWAPPED
 0308 HOLDCH = \$7C 124 CH MOVED HERE BEFORE CTRL AND SHIFT
 0309 INSDAT = \$7D 125

B. OPERATING SYSTEM EQUATES

```
0310 COUNTR = $7E 126
0311 ;
0312 ZROFRE = $80 128 FREE ZERO PAGE, 84 BYTES
0313 FPZRO = $D4 212 FLOATING POINT RAM, 43 BYTES
0314 FR0 = $D4 212 FP REGISTER 0
0315 FRE = $DA 218
0316 FR1 = $E0 224 FP REGISTER 1
0317 FR2 = $E6 230 FP REGISTER 2
0318 FRX = $EC 236 SPARE
0319 EEXP = $ED 237 VALUE OF E
0320 NSIGN = $ED 237 SIGN OF FP NUMBER
0321 ESIGN = $EF 239 SIGN OF FP EXPONENT
0322 FCHFLG = $F0 240 FIRST CHARACTER FLAG
0323 DIGRT = $F1 241 NUMBER OF DIGITS RIGHT OF DECIMAL POINT
0324 CIX = $F2 242 INPUT INDEX
0325 INBUFF = $F3 243 POINTER TO ASCII FP NUMBER
0326 ZTEMP1 = $F5 245
0327 ZTEMP4 = $F7 247
0328 ZTEMP3 = $F9 249
0329 DEGFLG = $FB 251
0330 RADFLG = $FB 251 0=RADIANS, 6=DEGREES
0331 FLPTR = $FC 252 POINTER TO BCD FP NUMBER
0332 FPTR2 = $FE 254
0333 ;
0334 ;
0335 ; PAGE 1
0336 ;
0337 ; 6502 STACK
0338 ;
0339 ;
0340 ;
0341 ;
0342 ; PAGE 2
0343 ;
0344 ;
0345 INTABS = $0200 512 INTERRUPT RAM
0346 VDSLST = $0200 512 NMI VECTOR
0347 VPRCED = $0202 514 PROCEED LINE IRQ VECTOR
0348 VINTER = $0204 516 INTERRUPT LINE IRQ VECTOR
0349 VBREAK = $0206 518
0350 VKEYBD = $0208 520
0351 VSERIN = $020A 522 SERIAL INPUT READY IRQ
0352 VSEROR = $020C 524 SERIAL OUTPUT READY IRQ
0353 VSEROC = $020E 526 SERIAL OUTPUT COMPLETE IRQ
```

0354 VTIMR1 = \$0210 528 TIMER 1 IRQ
 0355 VTIMR2 = \$0212 530 TIMER 2 IRQ
 0356 VTIMR4 = \$0214 532 TIMER 4 IRQ
 0357 VIMIRQ = \$0216 534 IRQ VECTOR
 0358 CDMV1 = \$0218 536 DOWN TIMER 1
 0359 CDMV2 = \$021A 538 DOWN TIMER 2
 0360 CDMV3 = \$021C 540 DOWN TIMER 3
 0361 CDMV4 = \$021E 542 DOWN TIMER 4
 0362 CDMV5 = \$0220 544 DOWN TIMER 5
 0363 VVBLKI = \$0222 546
 0364 VVBLKD = \$0224 548
 0365 CDTMA1 = \$0226 550 DOWN TIMER 1 JSR ADDRESS
 0366 CDTMA2 = \$0228 552 DOWN TIMER 2 JSR ADDRESS
 0367 CDTMF3 = \$022A 554 DOWN TIMER 3 FLAG
 0368 SRTIMR = \$022B 555 REPEAT TIMER
 0369 CDTMF4 = \$022C 556 DOWN TIMER 4 FLAG
 0370 INTEMP = \$022D 557 IAN'S TEMP
 0371 CDTMF5 = \$022E 558 DOWN TIMER FLAG 5
 0372 SDMCTL = \$022F 559 DMACTL SHADOW
 0373 SDLSTL = \$0230 560 DISPLAY LIST POINTER
 0374 SSKCTL = \$0232 562 SKCTL SHADOW
 0375 ; \$0233 563 (800) UNLISTED
 0376 LCOUNT = \$0233 563 (XL) LOADER TEMP
 0377 LPENH = \$0234 564 LIGHT PEN HORIZONTAL
 0378 LPENV = \$0235 565 LIGHT PEN VERTICAL
 0379 ; \$0236 566 2 SPARE BYTES
 0380 ; \$0238 568 (800) SPARE, 2 BYTES
 0381 RELADR = \$0238 568 (XL) LOADER
 0382 CDEVIC = \$023A 570 DEVICE COMMAND FRAME BUFFER
 0383 CAUX1 = \$023C 572 DEVICE COMMAND AUX 1
 0384 CAUX2 = \$023D 573 DEVICE COMMAND AUX 2
 0385 TEMP = \$023E 574 TEMPORARY STORAGE
 0386 ERRFLG = \$023F 575 DEVICE ERROR FLAG (EXCEPT TIMEOUT)
 0387 DFLAGS = \$0240 576 FLAGS FROM DISK SECTOR 1
 0388 DBSECT = \$0241 577 NUMBER OF BOOT DISK SECTORS
 0389 BOOTAD = \$0242 578 BOOT LOAD ADDRESS POINTER
 0390 COLDST = \$0244 580 COLD START FLAG, 1 = COLD START IN
 PROGRESS
 0391 ; \$0245 581 (800) SPARE
 0392 RECLEN = \$0245 581 (XL) LOADER
 0393 DSKTIM = \$0246 582 (800) DISK TIME OUT REGISTER
 0394 ; \$0246 582 (XL) RESERVED, 39 BYTES
 0395 LINBUF = \$0247 583 (800) CHARACTER LINE BUFFER, 40 BYTES
 0396 CHSALT = \$026B 619 (XL) CHARACTER SET POINTER

B. OPERATING SYSTEM EQUATES

```
0397 VSFLAG = $026C 620 (XL) FINE SCROLL TEMPORARY
0398 KEYDIS = $026D 621 (XL) KEYBOARD DISABLE
0399 FINE = $026E 622 (XL) FINE SCROLL FLAG
0400 GPRIOR = $026F 623 P/M PRIORITY AND GTIA MODES
0401 GTIA = $026F 623
0402 PADDL0 = $0270 624 (XL) 3 MORE PADDLES, (800) 6 MORE PADDLES
0403 STICK0 = $0278 632 (XL) 1 MORE STICK, (800) 3 MORE STICKS
0404 PTRIG0 = $027C 636 (XL) 3 MORE PADDLE TRIGGERS, (800) 6 MORE
0405 STRIG0 = $0284 644 (XL) 1 MORE STICK TRIGGER, (800) 3 MORE
0406 CSTAT = $0288 648 (800)
0407 WMODE = $0289 649
0408 BLIM = $028A 650
0409 ; $028B 651 5 SPARE BYTES
0410 NEWADR = $028E 654 (XL) LOADER RAM
0411 TXTROW = $0290 656
0412 TXTCOL = $0291 657
0413 TINDEX = $0293 659 TEXT INDEX
0414 TXTMSC = $0294 660
0415 TXTOLD = $0296 662 OLD ROW AND OLD COL FOR TEXT, 2 BYTES
0416 ; $0298 664 4 SPARE BYTES
0417 TMPX1 = $029C 668 (800)
0418 CRETRY = $029C 668 (XL) NUMBER OF COMMAND FRAME RETRIES
0419 SUBTMP = $029E 670
0420 HOLD2 = $029F 671
0421 DMASK = $02A0 672
0422 TEMPLBT = $02A1 673
0423 ESCFLG = $02A2 674
0424 TABMAP = $02A3 675 15 BYTE BIT MAP FOR TAB SETTINGS
0425 LOGMAP = $02B2 690 4 BYTE LOGICAL LINE START BIT MAP
0426 INVFLG = $02B6 694
0427 FILFLG = $02B7 695 FILL DIRING DRAW FLAG
0428 TMPROW = $02B8 696
0429 TMPCOL = $02B9 697
0430 SCRFLG = $02BB 699 SCROLL FLAG
0431 HOLD4 = $02BC 700
0432 HOLD5 = $02BD 701 (800)
0433 DRETRY = $02BD 701 (XL) NUMBER OF DEVICE RETRIES
0434 SHFLOC = $02BE 702
0435 BOTSCR = $02BF 703 24 NORM, 4 SPLIT
0436 PCOLR0 = $02C0 704 3 MORE PLAYER COLOR REGISTERS
0437 COLOR0 = $02C4 708 4 MORE GRAPHICS COLOR REGISTERS
0438 ; $02C9 713 (800) 23 SPARE BYTES
0439 RUNADR = $02C9 713 (XL) LOADER VECTOR
0440 HIUSED = $02CB 715 (XL) LOADER VECTOR
```

0441 ZHIUSE = \$02CD 717 (XL) LOADER VECTOR
0442 GBYTEA = \$02CF 719 (XL) LOADER VECTOR
0443 LOADAD = \$02D1 721 (XL) LOADER VECTOR
0444 ZLOADA = \$02D3 723 (XL) LOADER VECTOR
0445 DSCTLN = \$02D5 725 (XL) DISK SECTOR SIZ
0446 ACMISR = \$02D7 727 (XL) RESERVED
0447 KRPDER = \$02D9 729 (XL) KEY AUTO REPEAT DELAY
0448 KEYREP = \$02DA 730 (XL) KEY AUTO REPEAT RATE
0449 NOCLIK = \$02DB 731 (XL) KEY CLICK DISABLE
0450 HELPPFG = \$02DC 732 (XL) HELP KEY FLAG
0451 DMASAV = \$02DD 733 (XL) SDMCTL (DMA) SAVE
0452 PBPNT = \$02DE 734 (XL) PRINTER BUFFER POINTER
0453 PBUFSZ = \$02DF 735 (XL) PRINTER BUFFER SIZE
0454 GLBABS = \$02E0 736 GLOBAL VARIABLES, 4 SPARE BYTES
0455 RAMSIZ = \$02E4 740 PERMANENT START OF ROM POINTER
0456 MEMTOP = \$02E5 741 END OF FREE RAM
0457 MEMLO = \$02E7 743
0458 ; \$02E9 745 (800) SPARE
0459 HNDLOD = \$02E9 745 (XL) HANDLER LOADER FLAG
0460 DVSTAT = \$02EA 746 DEVICE STATUS BUFFER, 4 BYTES
0461 CBAUDL = \$02EE 750 CASSETTE BAUD RATE, 2 BYTES
0462 CRSINH = \$02F0 752 1 = INHIBIT CURSOR
0463 KEYDEL = \$02F1 753 KEY DELAY AND RATE
0464 CH1 = \$02F2 754
0465 CHACT = \$02F3 755
0466 CHBAS = \$02F4 756 CHARACTER SET POINTER
0467 NEWROW = \$02F5 757 (XL) DRAW DESTINATION
0468 NEWCOL = \$02F6 758 (XL) DRAW DESTINATION
0469 ROWINC = \$02F8 760 (XL)
0470 COLINC = \$02F9 761 (XL)
0471 CHAR = \$02FA 762
0472 ATACHR = \$02FB 763 ATASCII CHARACTER FOR CIO
0473 CH = \$02FC 764
0474 FILDAT = \$02FC 764 COLOR FOR SCREEN FILL
0475 DSPFLG = \$02FE 766 DISPLAY CONTROL CHARACTERS FLAG
0476 SSFLAG = \$02FF 767 DISPLAY START/STOP FLAFLG
0477 ;
0478 ;
0479 ; PAGE 3
0480 ;
0481 ;
0482 ; RESIDENT DISK HANDLER/SIO INTERFACE
0483 ;
0484 DCB = \$0300 768 DEVICE CONTROL BLOCK

B. OPERATING SYSTEM EQUATES

```
0485 DDEVIC = $0300 768
0486 DUNIT = $0301 769
0487 DCOMND = $0302 770
0488 DSTATS = $0303 771
0489 DBUFLO = $0304 772
0490 DBUFHI = $0305 773
0491 DTIMLO = $0306 774
0492 DBYTLO = $0308 776
0493 DBYTHI = $0309 777
0494 DAUX1 = $030A 778
0495 DAUX2 = $030B 779
0496 TIMER1 = $030C 780 INITIAL TIMER VALUE
0497 ADDCOR = $030E 782 (800) ADDITION CORRECTION
0498 JMPERS = $030E 782 (XL) OPTION JUMPERS
0499 CASFLG = $030F 783 CASSETTE MODE WHEN SET
0500 TIMER2 = $0310 784 FINAL VALUE, TIMERS 1 & 2 DETERMINE BAUD
RATE
0501 TEMP1 = $0312 786
0502 TEMP2 = $0313 787 (XL)
0503 TEMP2 = $0314 788 (800)
0504 PTIMOT = $0314 788 (XL) PRINTER TIME OUT
0505 TEMP3 = $0315 789
0506 SAVIO = $0316 790 SAVE SERIAL IN DATA PORT
0507 TIMFLG = $0317 791 TIME OUT FLAG FOR BAUD RATE CORRECTION
0508 STACKP = $0318 792 SIO STACK POINTER SAVE
0509 TSTAT = $0319 793 TEMPORARY STATUS HOLDER
0510 HATABS = $031A 794 HANDLER ADDRESS TABLE, 38 BYTES
0511 MAXDEV = $0321 801 MAXIMUM HANDLER ADDRESS INDEX
0512 PUPBT1 = $033D 829 (XL) POWER-UP/RESET
0513 PUPBT2 = $033E 830 (XL) POWER-UP/RESET
0514 PUPBT3 = $033F 831 (XL) POWER-UP/RESET
0515 ;
0516 ;IOCB' s
0517 ;
0518 IOCB = $0340 832
0519 ICHID = $0340 832
0520 ICDNO = $0341 833
0521 ICCOM = $0342 834
0522 ICSTA = $0343 835
0523 ICBAL = $0344 836
0524 ICBAH = $0345 837
0525 ICPTL = $0346 838
0526 ICPH = $0347 839
0527 ICBL = $0348 840
```

0528 ICBLH = \$0349 841
0529 ICAX1 = \$034A 842
0530 ICAX2 = \$034B 843
0531 ICAX3 = \$034C 844
0532 ICAX4 = \$034D 845
0533 ICAX5 = \$034E 846
0534 ICAX6 = \$034F 847
0535 ; OTHER IOCB's, 112 BYTES
0536 PRNBUF = \$03C0 960 PRINTER BUFFER, 40 BYTES
0537 ; \$03E8 1000 (800) 21 SPARE BYTES
0538 SUPERF = \$03E8 1000 (XL) SCREEN EDITOR
0539 CKEY = \$03E9 1001 (XL) START KEY FLAG
0540 CASSBT = \$03EA 1002 (XL) CASSETTE BOOT FLAG
0541 CARTCK = \$03EB 1003 (XL) CARTRIDGE CHECKSUM
0542 ACMVAR = \$03ED 1005 (XL) RESERVED, 6 BYTES
0543 MINTLK = \$03F9 1017 (XL) RESERVED
0544 GINTLK = \$03FA 1018 (XL) CARTRIDGE INTERLOCK
0545 CHLINK = \$03FB 1019 (XL) HANDLER CHAIN, 2 BYTES
0546 CASBUF = \$03FD 1021 CASSETTE BUFFER, 131 BYTES TO \$047F
0547 ;
0548 ;
0549 ; PAGE 4
0550 ;
0551 ;
0552 USAREA = \$0480 1152 128 SPARE BYTES
0553 ;
0554 ; SEE APPENDIX C FOR PAGES 4 AND 5 USAGE
0555 ;
0556 ;
0557 ;
0558 ;
0559 ; PAGE 5
0560 ;
0561 PAGE5 = \$0500 1280 127 FREE BYTES
0562 ; \$057E 1406 129 FREE BYTES IF FLOATING POINT ROUTINES
NOT USED
0563 ;
0564 ;FLOATING POINT NON-ZERO PAGE RAM, NEEDED ONLY IF FP IS USED
0565 ;
0566 LBPR1 = \$057E 1406 LBUFF PREFIX 1
0567 LBPR2 = \$05FE 1534 LBUFF PREFIX 2
0568 LBUFF = \$0580 1408 LINE BUFFER
0569 PLYARG = \$05E0 1504 POLYNOMIAL ARGUMENTS
0570 FPSCR = \$05E6 1510 PLYARG+FPREC

B. OPERATING SYSTEM EQUATES

```
0571 FPSCR1 = $05EC 1516 FPSCR+FPREC
0572 FSCR = $05E6 1510 =FPSCR
0573 FSCR1 = $05EC 1516 =FPSCR1
0574 LBFEND = $05FF 1535 END OF LBUFF
0575 ;
0576 ;
0577 ; PAGE 6
0578 ;
0579 ;
0580 PAGE6 = $0600 1536 256 FREE BYTES
0581 ;
0582 ;
0583 ; PAGE 7
0584 ;
0585 ;
0586 BOOTRG = $0700 1792 PROGRAM AREA
0587 ;
0588 ;
0589 ; UPPER ADDRESSES
0590 ;
0591 ;
0592 RITCAR = $8000 32768 RAM IF NO CARTRIDGE
0593 LFTCAR = $A000 40960 RAM IF NO CARTRIDGE
0594 C0PAGE = $C000 49152 (800) EMPTY, 4K BYTES
0595 C0PAGE = $C000 49152 (XL) 2K FREE RAM IF NO CARTRIDGE
0596 ; $C800 51200 (XL) START OF OS ROM
0597 CHORG2 = $CC00 52224 (XL) INTERNATIONAL CHARACTER SET
0598 ;
0599 ;
0600 ; HARDWARE REGISTERS
0601 ;
0602 ;
0603 ; SEE REGISTER LIST FOR MORE INFORMATION
0604 ;
0605 ;
0606 HPOSP0 = $D000 53248
0607 MOPF = $D000 53248
0608 SIZEP0 = $D008 53256
0609 MOPL = $D008 53256
0610 SIZEM = $D00C 53260
0611 GRAFP0 = $D00D 53261
0612 GRAFM = $D011 53265
0613 COLPM0 = $D012 53266
0614 COLPF0 = $D016 53270
```


0615 PRIOR = \$D01B 53275
0616 GTIAR = \$D01B 53275
0617 VDELAY = \$D01C 53276
0618 GRCTL = \$D01D 53277
0619 HITCLR = \$D01E 53278
0620 CONSOL = \$D01F 53279
0621 AUDF1 = \$D200 53760
0622 AUDC1 = \$D201 53761
0623 AUDCTL = \$D208 53768
0624 RANDOM = \$D20A 53770
0625 IRQEN = \$D20E 53774
0626 SKCTL = \$D20F 53775
0627 PORTA = \$D300 54016
0628 PORTB = \$D301 54017
0629 PACTL = \$D302 54018
0630 PBCTL = \$D303 54019
0631 DMACTL = \$D400 54272
0632 DLISTL = \$D402 54274
0633 HSCROL = \$D404 54276
0634 VSCROL = \$D405 54277
0635 CHBASE = \$D409 54281
0636 WSYNC = \$D40A 54282
0637 VCOUNT = \$D40B 54283
0638 NMIEN = \$D40E 54286
0639 ;
0640 ; FLOATING POINT MATH ROUTINES
0641 ;
0642 AFP = \$D800 55296
0643 FASC = \$D8E6 55526
0644 IFP = \$D9AA 55722
0645 FPI = \$D9D2 55762
0646 ZFR0 = \$DA44 55876
0647 ZF1 = \$DA46 55878
0648 FSUB = \$DA60 55904
0649 FADD = \$DA66 55910
0650 FMUL = \$DADB 56027
0651 FDIV = \$DB28 56104
0652 PLYEVL = \$DD40 56640
0653 FLDOR = \$DD89 56713
0654 FLDOP = \$DD8D 56717
0655 FLD1R = \$DD98 56728
0656 FLD1P = \$DD9C 56732
0657 FSTOR = \$DDA7 56743
0658 FSTOP = \$DDAB 56747

B. OPERATING SYSTEM EQUATES

```
0659 FMOVE = $DDB6 56758
0660 EXP = $DDC0 56768
0661 EXP10 = $DDCC 56780
0662 LOG = $DECD 57037
0663 LOG10 = $DED1 57041
0664 ;
0665 ;
0666 ; OPERATING SYSTEM
0667 ;
0668 ;
0669 ; MODULE ORIGIN TABLE
0670 ;
0671 CHORG = $E000 57344 CHARACTER SET, 1K
0672 VECTBL = $E400 58368 VECTOR TABLE
0673 VCTABL = $E480 58496 RAM VECTOR INITIAL VALUE TABLE
0674 CIOORG = $E4A6 58534 CIO HANDLER
0675 INTORG = $E6D5 59093 INTERRUPT HANDLER
0676 SIOORG = $E944 59716 SIO DRIVER
0677 DSKORT = $EDEA 60906 DISK HANDLER
0678 PRNORG = $EE78 61048 PRINTER HANDLER
0679 CASORG = $EE78 61048 CASSETTE HANDLER
0680 MONORG = $F0E3 61667 MONITOR/POWER UP MODULE
0681 KBDORG = $F3E4 62436 KEYBOARD/DISPLAY HANDLER
0682 ;
0683 ;
0684 ; VECTOR TABLE, CONTAINS ADDRESSES OF CIO ROUTINES IN THE
0685 ; FOLLOWING ORDER. THE ADDRESSES IN THE TABLE ARE TRUE ADDRESSES
0686 ;
0687 ; ADDRESS + 0 OPEN
0688 ; + 2 CLOSE
0689 ; + 4 GET
0690 ; + 6 PUT
0691 ; + 8 STATUS
0692 ; + A SPECIAL
0693 ; + C JMP TO INITIALIZATION
0694 ; + F NOT USED
0695 ;
0696 ;
0697 EDITRV = $E400 58368 EDITOR
0698 SCRENV = $E410 58384 SCREEN
0699 KEYBDV = $E420 58400 KEYBOARD
0700 PRINTV = $E430 58416 PRINTER
0701 CASETV = $E440 58432 CASSETTE
0702 ;
```

```
0703 ; ROM VECTORS
0704 ;
0705 DSKINV = $E453 58451
0706 CIOV = $E456 58454
0707 SIOV = $E459 58457
0708 SYSVBV = $E45F 58463
0709 VBIVAL = $E460 58464 ADR AT VVBLKI
0710 XITVBV = $E462 58466 EXIT VBI
0711 VBIXVL = $E463 58467 ADR AT VVBLKD
0712 BLKBDV = $E471 58481 MEMO PAD MODE
0713 WARMSV = $E474 58484
0714 COLDSV = $E477 58487
```

B. OPERATING SYSTEM EQUATES

C. MEMORY USE

Page 0

\$00-\$7F Operating system zero-page. The entire first half of page zero is reserved for the operating system.

\$80-\$FF Free zero-page. The top half of page zero is free if BASIC is disabled. BASIC uses all but \$CB-\$D1. The floating point math routines use \$D4-\$FF. If the floating point arithmetic package is not used this memory is free.

Page 1

\$100-1FF This is the 6502 stack. The stack pointer initialized to \$1FF and moves downward as the stack is filled.

Pages 2-5

\$200-\$47F This area is used for operating system database variables. Parts which are not used in some particular programs, such as the cassette buffer or printer buffer, may then be used for other purposes. See the O.S. equate listing for these locations.

\$480-\$57D (\$480-\$6FF if no floating point) This is called the user work space. It is free to be used by programs. If the floating point arithmetic package is not used the user work space extends to \$6FF. This area is used by BASIC.

\$57E-\$5FF This area is used by the floating point arithmetic package. It is free if the package is not used.

Page 6

\$600-6FF Atari has solemnly sworn never to put anything in this page of memory.

Page 7-the screen region

\$700 This is called the boot region. Most machine language programs which don't use DOS load at this address. DOS extends from \$700-\$1CFB.

MEMLO

The address pointed to by the O.S. database variable MEMLO [\$02E7,2 (743)] is the first byte of free memory. This pointer is usually changed by any program's initialization routine. For example, upon power-up, MEMLO points to \$700. When DOS loads in, DOS changes MEMLO to point to \$2A80. If an AUTORUN.SYS program then loads in just above DOS, such as DISKIO, it will usually change MEMLO to point above itself. One important reason for this is to protect the program from BASIC. BASIC uses memory starting at MEMLO.

C. MEMORY USE

MEMTOP

MEMTOP [\$2E5,2 (741)] is set by the O.S. whenever a graphics mode is entered. The graphics region is at the very top of ram and extends downward. The address MEMTOP points to depends on how much memory the screen region uses.

APPMHI

APPMHI [\$0E,2 (14)] should be set by any program to point to the highest address required by the program. If the O.S. cannot set up a screen without going below APPMHI it will return a not-enough-memory-for-screen-mode error.

The cartridge slots

\$8000 (32768) This is the beginning of the 8K bytes used by the right cartridge slot of the 800. This is also where 16K cartridges begin. If there is no cartridge here it is ram.

\$A000 (40960) This is the beginning of the left cartridge of the 800 or the only cartridge slot on all other models. This is where the BASIC ROM resides in the XL/XE models. This area is RAM is there is no cartridge or BASIC is disabled on XL/XE models.

above the cartridges

\$C000-\$CFFF (49152-53247) This area is empty on the 800. Sometimes special ROM chips, such as Omnimon are wired in here. On the XL/XE models \$C000-C7FF is free ram if there are no cartridges. On XL/XE models, the O.S. ROM starts at \$C800

\$D000-\$D7FF (53248-57373) This area is taken up by the hardware chips. The chips actually take only a fraction of this space. If these addresses are further decoded there is space for many, many more hardware chips. For example, The PIA chip uses 256 bytes of memory but needs only 4 bytes. There is room for 64 PIA chips in this reserved memory.

\$E000-E3FF (57344-58367) This is the location of the ATASCII character set.

\$E400-FFF7 (58368-65527) This is the operating system ROM

\$FFF8-\$FFFF (65528-65535) These last 8 bytes contain the addresses of the interrupt vectors. Upon power up the 6502 gets a reset pulse and looks up the reset routine here.

List of Tables

1.1. IOCB Memory Addresses and usage	14
1.2. CIO command codes	15
1.3. CIO direction codes	16
1.4. CIO device names	18
1.5. Useful database variables and OS equates	23
2.1. binary load file header	27
2.2. Run tailer	28
2.3. Init tailer	28
2.4. Disk specific ICAX1 options	30
2.5. XIO commands specific to the disk drive	31
2.6. Status request codes	32
4.1. Cassette handler control byte	41
4.2. Cassette motor control	42
5.1. The [HELP] key register	45
5.3. Useful database variables and OS equates	45
7.1. Screen editor control codes	50
8.1. DISPLAY CODE / ATASCII CODE CONVERSION	53
8.2. CIO Color	54
8.3. SETCOLOR/register relationships	60
8.4. colors represented by hue numbers	61
9.1. Device control block (for resident disk handler)	63
10.1. Interrupt vectors	71

List of Tables

List of Figures

0.1. Register format	11
1.1. Channel-open parameters:	16
1.2. ICAX1 format for opening a channel	16
1.3. IOCB input parameters	18
1.4. IOCB get-byte parameters	20
1.5. Handler vector table	22
2.1. Boot Error	25
2.2. Directory flag byte	26
2.3. File sector structure	27
2.4. ICAX1 for disk open	30
4.1. Cassette record format	41
5.1. Key code format	43
5.2. The console switch register	44
8.1. Text Mode 0 Display Buffer Format	51
8.2. Letter A as represented in the C-set	52
8.3. display buffer byte for mode 3	55
8.4. ICAX1 for screen open	57
8.5. Color register data format	60
9.1. DVSTAT Status format	65
10.1. 6502 interrupts	67
11.1. Floating point number representation	75
11.2. RAM used by floating point package	80