CIO Tutorial#

Reprinted from A.C.E.C. BBS (614)-471-8559

Accessing & Using the CIOV#

The Central Input/Output Vector (CIOV) is how Atari BASIC accesses all of the peripherals (the disk drive, modem, screen, etc.). Performing various I/O functions from BASIC is easy since BASIC contains special commands to allow the user to access the CIOV easily. From machine language, however, it is quite a different story. This Programming Guide will explain how to use and access the CIOV in your machine language programs. It assumes a working knowledge of 6502 machine language.

All I/O functions are executed by setting several locations in page 3 and then jumping the the CIOV. The address of the CIOV is $E456 (58454 decimal).

The areas in Page 3 that contains the locations that must be set prior to accessing the CIOV are called Input/Output Control Blocks (IOCB). Each I/O Channel has one corresponding IOCB. Each IOCB is 16 bytes long. The addresses of each IOCB are as follows:

$340-$34F:	IOCB For Channel 0	
$350-$35F:	IOCB For Channel 1	
$360-$36F:	IOCB For Channel 2	
$370-$37F:	IOCB For Channel 3	
$380-$38F:	IOCB For Channel 4	
$390-$39F:	IOCB For Channel 5	
$3A0-$3AF:	IOCB For Channel 6	
$3B0-$3BF:	IOCB For Channel 7	

You only access locations which are in the IOCB for the channel you are accessing. In other words, if you are doing I/O with channel #3, the only locations you will deal with are $370-$37F. The following are "Location Names" for each byte of the IOCB. After the name is the offset of the byte within each IOCB and a short description of the purpose of the location. ICCOM: (+2) Command to be executed. You must set this before accessing the CIOV.

ICBAL/ICBAH:	(+4, +5)	This is a two byte number that contains the buffer address for the data being involved in the operation. This is not always required but usually is.	
ICBLL/ICBLH	(+8, +9)	This is a two byte number that contains the number of bytes to be written or read.	
ICAX1, ICAX2, ICAX3, ICAX4, ICAX5, ICAX6	(+10, +11, +12, +13, +14, +15)	The last three bytes in the IOCB contain misc. data. The contents of these differ depending on what handler you are accessing.	

The first thing you must do is set ICCOM to the command you want to execute. ICCOM should be set to the following values depending on the desired operation:

3:	Open Channel Command	
5:	Input One Line (same as BASIC's INPUT Command)	
7:	Load Binary Buffer (similar to using multiple GETs in BASIC)	
9:	Write One Line (same as BASIC's PRINT Command)	
11:	Write Binary Buffer (similar to using multiple PUTs in BASIC)	
12:	Close Channel Command	
13:	Status Command (same as the STATUS function in BASIC)	

The second thing you must do is set ICBAL and ICBAH. You should set these to the low/high byte of the address where the data to be transfered is in memory. For example, if you wanted to dump some of memory starting at address $3035, you would set ICCOM to 11 (for WRITE BINARY BUFFER), set ICBAL to $35 (the low byte of the source address), and ICBAH to $30 (the high byte of the source address). If you were reading data into memory from a device you would set ICBAL and ICBAH to the address where you want the data to be put in memory.

The third and, usually, final thing you must do is set ICBLL and ICBLH. These contain the number of bytes that are going to be involved in the operation. This location must be set when using the following commands: 3, 5, 7, 9, and 11. It need not be set for the commands 12 and 13. Setting this location is VERY important. Expanding on the above example, lets say you wanted to dump locations $3035 through $3236 (a total of $201 bytes). You would set up ICCOM, ICBAL, and ICBAH as explained above. Then you would set ICBLL to $01 (the low byte of the number of bytes you wish to write) and set ICBLH to $02 (the high byte of the number of bytes you wish to write). When reading, the number stored here is the maximum number of bytes that are to be read--any bytes that are in excess of this value will be ignored and CIOV will return an error 137 (record truncated). This location must be set for the following commands: 5, 7, 9, and 11. When doing INPUTs (command 5) the CIOV will not accept lines that are longer than the size specified here. That means if you set ICBLL and ICBLH to 20 and get input from the user, the CIOV will not accept any more than 20 characters. The user will be able to continue entering input but only the first 20 bytes will be saved--the rest discarded. This is useful since you can set aside a fixed number of bytes in an input buffer and be sure that the user will not be able to overflow the buffer. When doing PRINTs (command 9) be sure to set this to the length of the longest line you expect to print. If you set these locations to 20 and try to print 30 characters only the first 20 will actually be printed.

The fourth thing is EXTREMELY important (don't worry, we've made it through the tough stuff). You MUST set the X register to the channel number you are doing to access MULTIPLIED BY 16. This means if you are accessing channel 1, you should LDX #16 (LDX #$10). If you are accessing channel 3, you should LDX #48 (LDX #$30). This is very important since this is the way the CIOV knows which IOCB to work with.

Finally, simply JSR $E456 to call the CIOV. The CIOV will do the rest. When it returns, the Y register will contain the error code. A code of 1 means the operation was successful. Any value greater than 127 indicates an error occured. The error codes are the same as those in BASIC.

One quick, and important note: You must set ICAX1 when issuing the OPEN command. ICAX1 should be set to the value for the type of operation you are going to do. That is, it should be 4 for

READ access, 6 for directory access, 8 for write access, etc. This works just like the corresponding codes in the BASIC OPEN command.

That's basically it! For general purpose I/O such as accessing disk files (or creating them), printing to the screen, and getting input from the user, this is all you need. Some device-specific commands may require that you set some of the Auxillary bytes (such as NOTE, POINT, etc.) but that is not in the scope of this particular Programming Guide. Look for another Programming Guide in the future that will explain some of the device-specific commands such as NOTE, POINT, RENAME, etc.