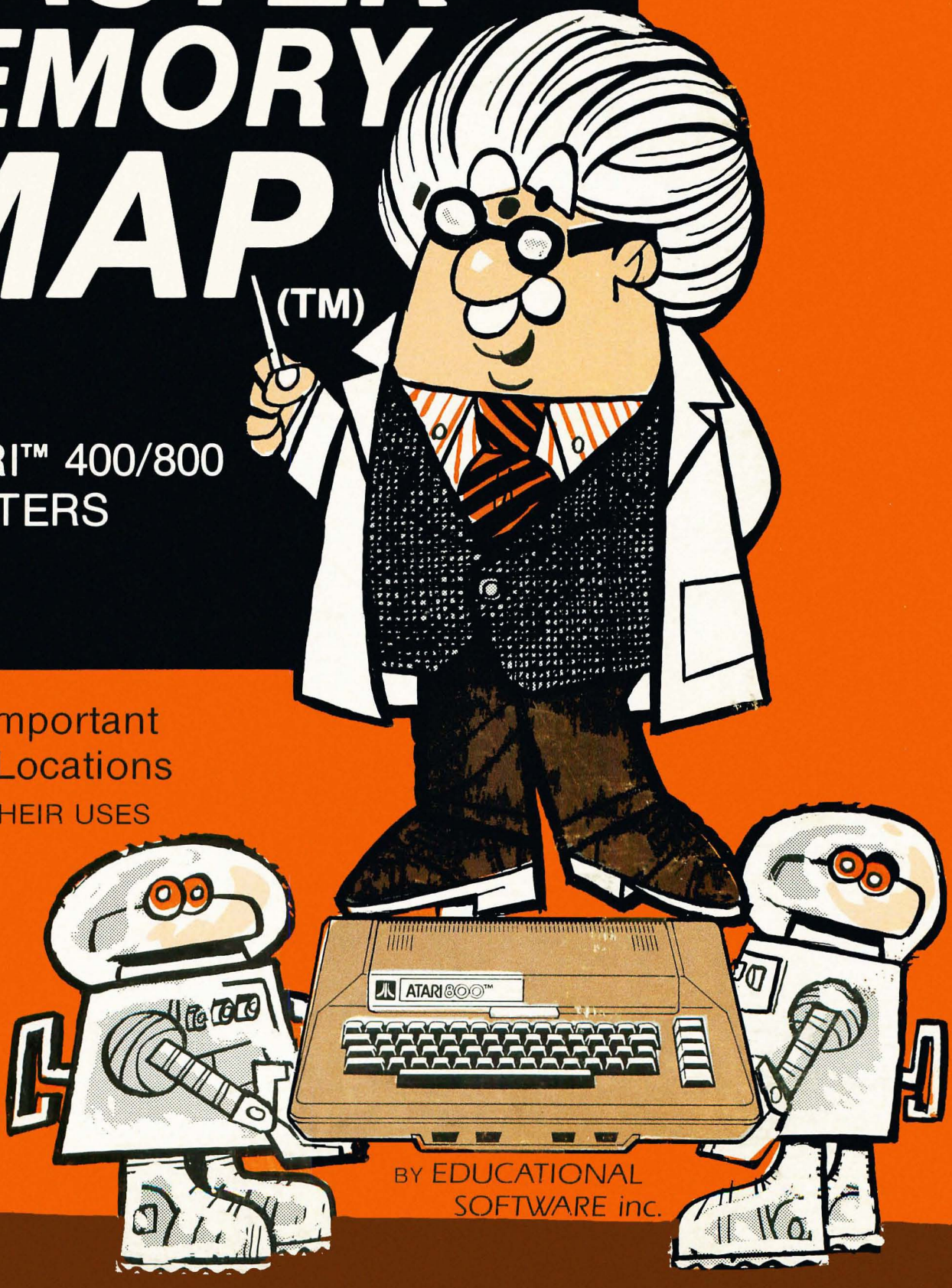
FOR BEGINNERS OR EXPERTS!

MASTER MEMORY MAP (TM)

for ATARI™ 400/800 COMPUTERS

100's of Important Memory Locations

HINTS ON THEIR USES



BY EDUCATIONAL SOFTWARE inc.

MASTER MEMORY MAP™

by

Robin Alan Sherer

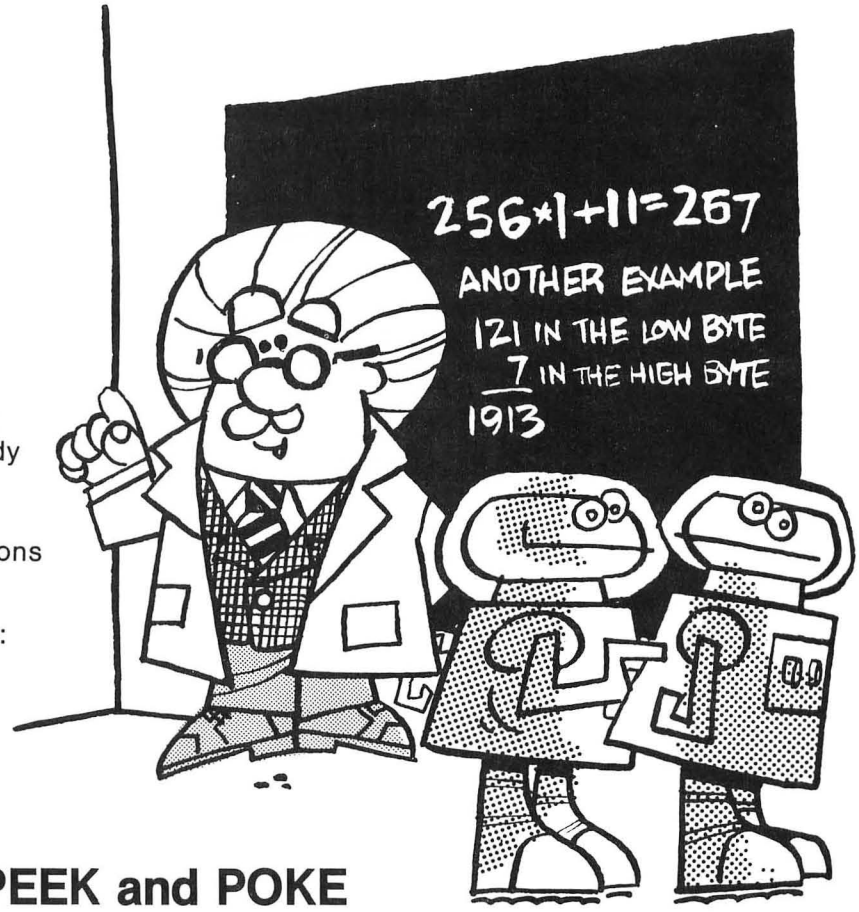
CONTENTS

Notes

Hello students.
I am Professor Von Chip.

I've been asked to teach you all about the wonderful things you can do with your ATARI computer. My lessons are called the TRICKY TUTORIALS™, but before you study them, I thought a nice reference book of all those memory locations would be nice. I have marked the most commonly used locations in **boldface** to make them easier to find.

The first thing we will discuss is for beginners:



How to PEEK and POKE

This part is for those who have yet to learn how to use a memory map. Basically, a memory map is a list of valuable locations within the computer (in this case an ATARI), that you can directly use for various purposes. These locations are simply bytes (memory locations) of memory at a specific place. If you have 16K of memory, then there are 16*1024 memory locations that you have to work with. Although some of these bytes are used for the computer's Operating System, most of them are blank for you to use in your programs. This manual will tell you about the ones that you can do something with.

For example, you can quickly look down this list to find the memory location that contains the value of the tabs. By following the included hints, you can change the "normal" value in that location, so that when you tab, the cursor now goes to the columns that you want. Please note that any of the changes that you make are only temporary and will go away when the computer is turned off.

Now to explain how to make changes from BASIC. Say you look down the list and decide to change location 752 (all numbers are decimal unless marked with a $ symbol, which denotes a HEXadecimal number, or in a column marked "hex"). 752 is called CRSINH by ATARI, and its function is to make the cursor visible or invisible. So, if at a certain point in a program you want to have a display without the cursor, you simply have to look up the correct value to POKE into location 752.

In this example, the memory map says that you use the number 1 for off and 0 for on, so to turn off the cursor we use 1. The basic instruction to put a number into memory is called "POKE" (see your ATARI BASIC Manual). After all this long-winded explanation, you can now see how simple an example of the actual BASIC code is:

```
10 POKE 752,1
```

Always use the decimal numbers with a POKE statement. This means that sometimes you will have to convert between binary, hexadecimal, and decimal. I will offer some hints on how to do this in a moment. Also, any one memory location can only hold a number up to 255. Why? . . . remember that the ATARI uses eight bits per word (memory location), and eight bits in binary counts from 0 to 255 (internally, the machine uses binary). You may want to read a book on computer mathematics. Because of this limitation, sometimes you must POKE numbers into two locations in a row. The computer will know to put these two numbers together to form one large number. For example, look at memory locations 741 & 742 which are called MEMTOP. These locations hold a number that corresponds to the top of your available memory (called RAM . . . Random Access Memory). Since the top of memory can be up to 48K (48*1024), a number well above the limit of 256 for any one memory location, the computer will need two locations to store the value. Yes, I know 256 for the first location plus 256 for the second doesn't seem to add up to a large enough number to hold 48K, but the computer takes each number in the second location and multiplies it by 256. Examples:

```
  11  stored in the low byte
+ 1   stored in the high byte
-----
 267
```

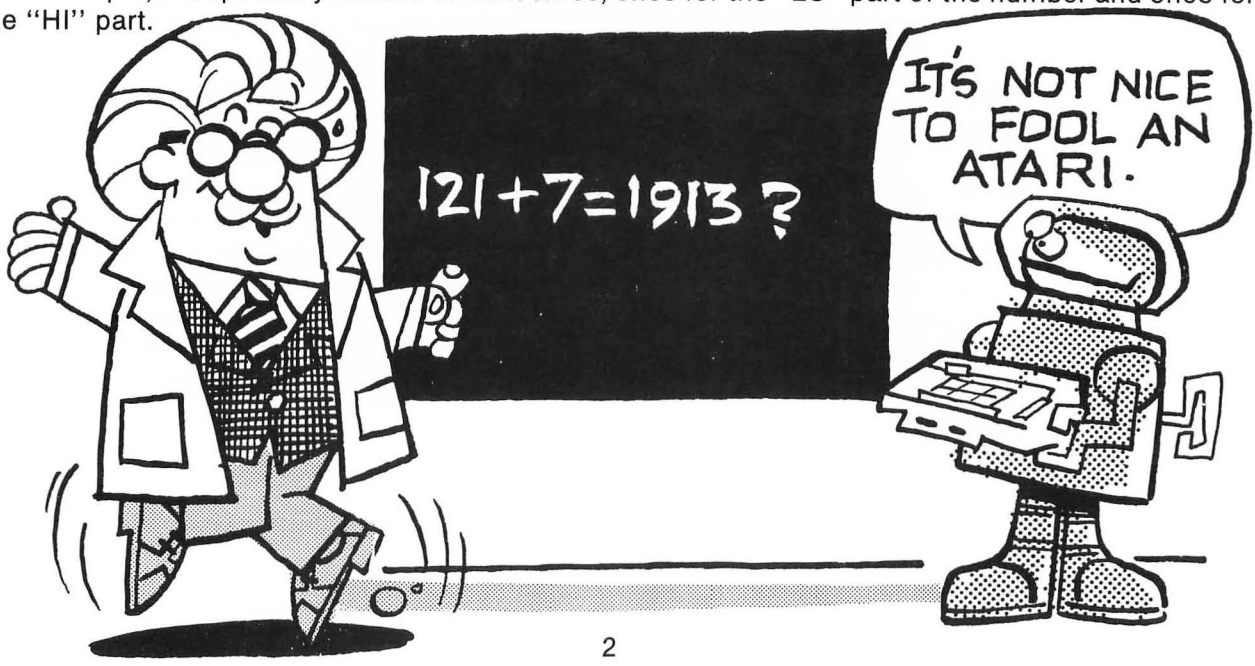
The computer "sees" 256*1 + 11 which equals 267.

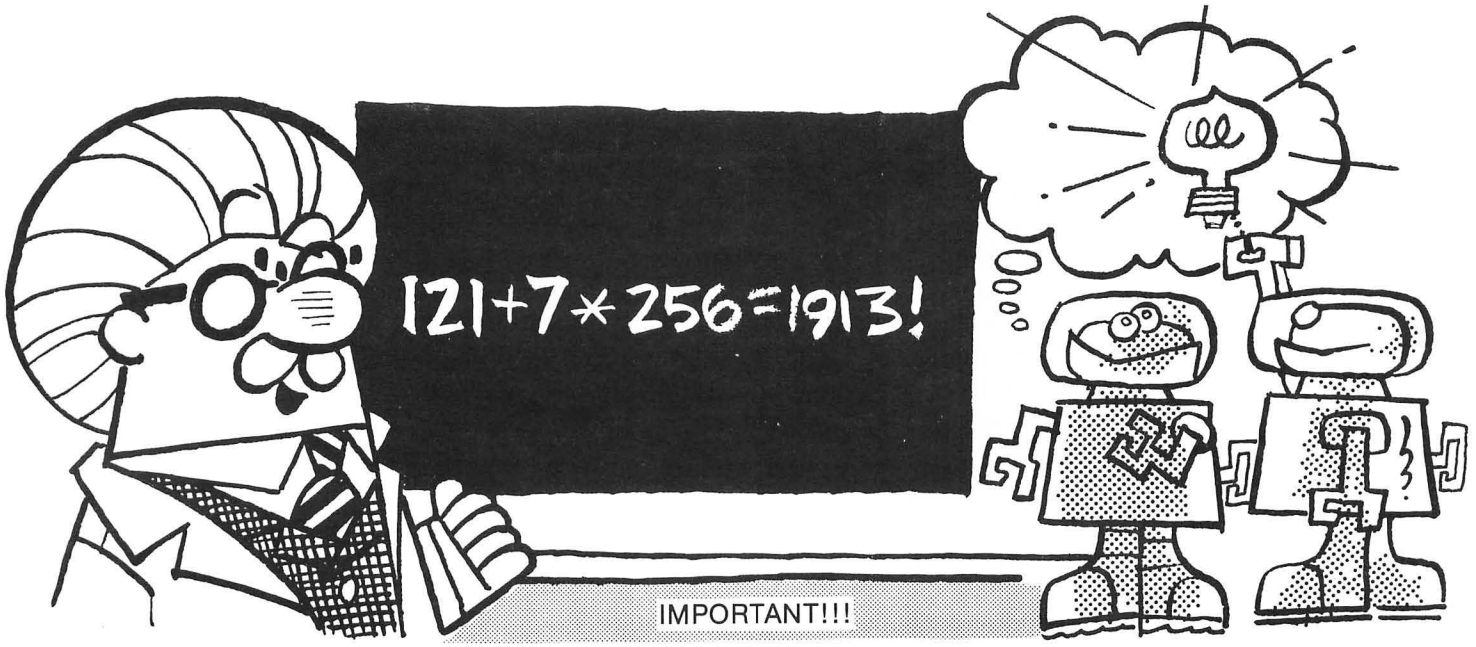
Another example:

```
 121  in the low byte
   7  in the high byte
-----
1913
```

Since 7*256 + 121 = 1913.

Sometimes it is desirable to fool the ATARI into thinking that the top of memory is lower than it actually is, perhaps to keep it from using the last few thousand bytes of memory, thus reserving them for your use (see my Tutorial on Page Flipping). You do the same type of POKE here as in the first example, except that you have to do it twice; once for the "LO" part of the number and once for the "HI" part.





I said the LO part of the number is placed in the first memory location and the HI part is next. Although it seems backwards, this is really not hard to understand. The ATARI (and most other microcomputers) store multiple part numbers this way. Occasionally this rule is broken, so please don't call me up if you find an exception.

Here's what you do. We want to change the value of MEMTOP to be 4K less than it currently is.

1) Find current value . . .

```
10 A = PEEK (741) + PEEK (742) *256
       LO Part              HI Part
```

EX: 12288 Which could be the value of your MEMTOP, and might result from running this statement.

2) Subtract 4K from this value . . . .

```
20 A = A – 4*1024
```

Remember that one K is actually 1024 bytes

3) Break the new value up into LO and HI parts . . . .

```
30 B = INT(A/256): C = A – B*256
```

What this does is make C the LO part of the number and B the HI part

EX: 12288 – 4K gives 8192.

The above line when run will give you 32 for the HIGH part and 0 for the LOW part.

4) POKE these values into memory . . . .

```
40 POKE 741, LO#: POKE 742, HI#   (#'s in decimal!!!)
```

EX: POKE 741,0:POKE 742,32

!!! FINAL WORDS OF WISDOM !!!

1) Feel free to POKE and PEEK all you want, trying out ideas or testing the effects mentioned in the Master Memory Map. The explanations are only the most basic part of how to do the various effects possible on the ATARI computer. I offer a series of TUTORIALS that will take you step by step through Display Lists, Page Flipping, Sound, Player Missile Graphics, and others. These are the techniques that the best programs use, and all of our Tutorials are done in BASIC, although we do sometimes include a machine language subroutine to offer you some advantage like speed. Player Missile Graphics even teaches you how to program a popular Arcade Game!

2) Any errors that occur will often reset values you have changed! You can use the TRAP statement to check for this.

3) Remember that two numbers are required to tell the computer the value for some locations, and these are stored LOW part, HIGH part. This is opposite of what you might think.

4) Many of the following locations are for advanced users only. Don't feel bad if you have no idea what they are for. Some have been left out because they seem of no use to almost any user. The idea is to experiment and learn.

5) This list is designed for general reference. Many locations used by the Operating System, BASIC, etc., are not mentioned. Also, the explanations are intended as a bonus (although we strive to make them accurate).

6) You can usually press Reset if trouble occurs. This will restore the original (default) values of many locations.

7) Many of the locations in the MASTER MEMORY MAP are used to read from only. If your program needs to write a value into one of these locations, you'll have to write instead the value into its "Shadow" location. The shadow locations hold the same value as their corresponding "HARDWARE" registers, except that you must remember that the Operating System will take the value in a Shadow and put it into its Hardware register every 1/60th of a second.

8) Many of these locations work from the BASIC cartridge, but not other languages. Try them to be sure.

Complete confusion?

Okay, just look at 53270. We marked the shadow as 708. To read or write color 0 (which is also known as a playfield color), use the 708 location. What happens if you write a color to 53270? Well, you will get the color you want for 1/60th of a second until the O.S. writes the "old" value from 708 back into 53270. All of the shadows we know of are marked in "( )" next to the hardware locations. NOTE!!! Please consider studying hex-binary-decimal conversion. It's not that hard after a little practice and really allows you to have more fun with your machine.

Here is an alternate method (use the chart on the back page):

8192(dec) = $2000(hex) . . . . $20 = HI part (hex) . . $20 = 32(dec, HI part)
$00 = LO part (hex) . . $00 = 00(dec, LO part)

Go back and re-read the last section at least a few hundred times. There are only four lines in the program that both read the old value of MEMTOP and store a new value. These lines don't have to be part of a program. You could enter them directly.

Locations for beginners are marked in **Boldface**.

Label	Decimal Location	Hexadecimal Location	Description and How to Use
CASINI	2,3	2,3	Cassette boot initialization vector. If cassette booted successfully during powerup then JSR thru here.
RAMLO	4	4	RAM pointer for memory test.
TRAMSZ	6	6	Temporary register for RAM size.
TSTDAT	7	7	RAM test data register.
WARMST	8	8	0 = Powerup initialization. 255 = normal. A warmstart will not wipe out memory, so your variables, programs and data may still be intact. The same as pressing RESET.
BOOT	9	9	Boot flag. The 1 bit is set to indicate that a program was booted from disk and will cause operation at each RESET to jump to the address given in DOSINI to perform disk initialization. The 2 bit is for a cassette booted program and provides for cassette initialization at the address given in CASINI. If both bits are set (value = 3), cassette initialization is performed before disk initialization.
DOSVEC	10,11	A,B	Noncartridge control vector. Disk software start vector.
DOSINI	12,13	C,D	Disk boot initialization vector. Used to store address of initialization of application upon the DOS boot—JSR indirect thru here to initialize application.
APPMHI	14,15	E,F	Contains the highest address of memory you can use for programs, data, etc.
POKMSK	**16**	**10**	**Try 64 here and also POKE 64 into 53774 to turn off the break key. You will want to do this in every program that you don't want anyone to stop while it's running. For example, press the BREAK key on your computer now and I will keep on talking! This location is also know as the POKEY interrupt vector. IRQ service uses and alters POKMSK. These are POKEY interrupts. Shadow for IRQEN [$D20E].** **bit 7 = 1 Break key interrupt enable.** **bit 6 = 1 Other key interrupt enable.** **bit 5 = 1 Serial input data ready interrupt enable.** **bit 4 = 1 Serial output data needed interrupt enable.** **bit 3 = 1 Serial out transmission finished interrupt enable.** **bit 2 = 1 Timer 4 interrupt enable.** **bit 1 = 1 Timer 1 interrupt enable.**
BRKKEY	17	11	0 = Break key pressed, 0 means it is not pressed.

RTCLOCK	18,19,20	12,13,14	Internal Clock . . . every 1/60 sec loc. 20 increments by one 'til 255 is reached then 20 becomes 0 and 19 increments by 1 while loc. 20 cycles through to 255 again and 19 increments, etc., 'til 19 reaches 255, then 18 increments, 19 & 20 are 0 and begin again . . . got that? Try this in a program:

```
100 SECONDS = INT((PEEK(18)*65536 + PEEK(19)*256 + PEEK(20))/60)
```

Said another way, loc. 20 changes with each TV frame, 60 per second. Loc. 19 changes every 4.27 seconds and 18 changes every 65536 TV frames, or 18.2 minutes. Isn't computer talk grand!

BUFADR	21	15	Indirect buffer address register. Used as temporary page 0 pointer to current disk buffer.
ICCOMT	23	17	Command for vector.
DSKFMS	24	18	Disk file manager pointer.
DSKUTL	26	1A	Disk utilities pointer.
PTIMOT	28	1C	Printer timeout every printer status request. Typical timeout for the 825 is 5 seconds. Inititialed to 30 sec.
PBPNT	29	1D	Print buffer pointer, index into printer buffer ranges from 0 to value of PBUFSZ.
PBUFSZ	30	1E	Print buffer size of printer record for current mode. normal = 40 bytes. double width = 20 bytes sideways = 29 bytes status = 4
PTEMP	31	1F	Printer handler uses this temp register to save value of character to output to printer.

Input/Output Control Blocks (IOCB)

The following locations, 32 to 47, are explained as an example of the use of IOCB (Input—Output Control Blocks). See locations 832 to 959 for other IOCB's (which are structured the same). CAUTION: DO NOT USE PAGE ZERO IOCB FOR NORMAL I/O USAGE.

			—Page Zero IOCB—
ICHIDZ	32	20	Handler index number. Set by the O.S. as an index into the device name table for a currently open file. Set to 255 if no file open on this IOCB.
ICDNOZ	33	21	Device drive number. Also set by the O.S. to 1 to 8 for the drive to use.
ICCOMZ	34	22	Command byte. From the user program. Defines how the rest of the IOCB is formated.
ICSTAZ	35	23	Status byte returned by the device. Set by the O.S. May or may not be the status returned by the STATUS command.

ICBALZ/H	36,37	24,25	Buffer address for data transfer or address of filename for commands like OPEN, STATUS, etc.
ICPTLZ/HZ	38,39	26,27	Put one byte vector. Set by O.S., it = the address – 1 of devices put one byte routine. Points to CIO's "IOCB not OPEN" (on CLOSE.)
ICBLLZ/HZ	40,41	28,29	Buffer length byte count for put/get operations. Decreases by one for each byte transferred.
ICAX1Z	42	2A	Auxiliary information used in OPEN to specify kind of file access needed.
ICAX2Z	43	2B	CIO working variables. Some serial port functions use this byte.
ICAX3&4Z	44,45	2C,2D	The place to transfer disk sector numbers from NOTE and POINT. Other uses possible.
ICAX5Z	46	2E	The byte within the above sector. IOCB Number multiplied by 16.
ICAX6Z	47	2F	Character byte for current operation.

Example uses of IOCB's

BASIC COMMAND	Operating System IOCB Parameters
OPEN #1,12,0,"E:"	IOCB = 1 Command = 3 (open) Aux1 = 12 (Input/Output) Aux2 = 0 Buffer Address = ADR ("E:")
GET #1,X	IOCB = 1 Command = 7 (Get Characters) Buffer Length = 0 Character returned in accumulator
PUT #1,X	IOCB = 1 Command = 11 (Put Characters) Buffer Length = 0 Character Length = 0 Character output through accumulator
INPUT #1,A$	IOCB = 1 Command = 5 (Get Record) Buffer Length = Length of A$ (not over 256) Buffer address = Input line Buffer
PRINT #1,A$	IOCB = 1 BASIC uses a special put byte vector in the IOCB to talk directly to the handler
XIO 18,#6,12,0,"S:"	IOCB = 6 Command = 18 (Special, "fill") AUX1 = 12 AUX2 = 0

STATUS	48	30	Internal status storage.
CHKSUM	49	31	Single byte sum with carry to least significant bit.
BUFRLO	50	32	Pointer to data buffer low byte??? Which buffer.
BUFRHI	51	33	Pointer to data buffer high byte.
BFENLO	52	34	Next byte (low byte) past end of data buffer.
BFENHI	53	35	Next byte (high byte) past end of data buffer.
CRETRY	54	36	Number of command frame retries.
DRETRY	55	37	Number of device retries.
BUFRFL	56	38	Buffer full flag.
RECVDN	57	39	Receive done flag.
XMTDON	58	3A	Transmission done flag.
CHKSNT	59	3B	Checksum sent flag.
NOCKSM	60	3C	No checksum follows data flag.
BPTR	61	3D	Cassette record data index into data portion of record being read or written. Values range 0 to current value BLIMI[$28A] when BPTR = BLIM then buffer CASBOFF[$3FD] is empty if reading or full if writing.
FTYPE	62	3E	Interrecord Gap type. Copy of ICAX2Z from OPEN command FTYPE 0 normal gaps. FTYPE $80 continuous gaps.
FEOF	63	3F	Cassette end of file flag used by cassette handler to indicate end of file.
FREQ	64	40	Beep count retain and count number of beeps requested of beep routine by cassette handler during open processing.
SOUNDER	**65**	**41**	**Noisy I/O flag . . . try = 0 to quiet the computer during certain operations like disk or tape inputs (beep . . . beep . . .beep).**
CRITIC	66	42	1 = disables repeat action of the keys and changes the sound of the cntl – 2 buzzer. Indicates critical I/O (e.g., data input), is taking place.
FMSZPO	67	43	Disk file manager zero page.
CKEY	74	4A	Cassette boot request flag on powerup (coldstart). Start key checked, if pressed then CKEY is set.
CASSBT	75	4B	Cassette boot flag.
DSTAT	76	4C	Display status used by display handler.
ATRACT	**77**	**4D**	**Less than 128 is normal operation, 128 gives reduced luminescence and rotates the colors to protect the screen. This could be used to instantly darken the screen for special effects in a game. If your program doesn't use the keyboard within several minutes (9.01 mins), you should POKE this location in the code to prevent the attract mode from happening. This happens, for example, during games that use only joysticks.**

DRKMSK	78	4E	254 = Normal brightness. Dark attract mask = $FE when attract mode inactive.
COLRSH	79	4F	Attract color shifter XOR'd with playfield colors. At stage 2 Vblank color registers are XOR'd with COLRSH and DRKMSK, then sent to hardware color registers when attract inactive COLRSH = 0 and DRKMSK = $F6 reducing luminence 50% and COLRSH = RTCLOCK + 1 affecting color change every 256/60 = 4.1 sec.
TEMP	80	50	Used by display handler in moving data to and from screen.
HOLD1	81	51	Same as TEMP.
LMRGIN	82	52	Left margin, default = 2
RMRGIN	83	53	Right margin, default = 39 Use these two to cause your text or graphics to start or end where YOU want it to!
ROWCRS	84	54	Current graphics cursor row (0 to 191). Try this:

```
10 ?"ONE":POKE 84,3:?"TWO":?"THREE"
```

COLCRS	85,86	55,56	Current graphics cursor column (0 to 319). These can be useful, either when you want to know where the cursor is, or, to place the cursor on the screen (must do a PRINT before change takes place). Try this, too.

```
10 ?"ONE":POKE 85,6:?"TWO":?"THREE"
```

CRMODE	87	57	Used to fool the O.S. into thinking that it is in a graphics mode other than what you originally had. For example, when modifying Display Lists, you often use this POKE to allow normal PLOT and PRINT commands to work. POKE with any number from 0 to 8 (regular graphics modes). Try a POKE 87,7 AFTER you have entered GRAPHICS 8. The top half of the screen will allow GR.7 while the bottom will still be GR.8! See our Display List Tutorial for more information.
SAVMSC	88,89	58,59	Lowest address of screen memory. Data in this address will be plotted at the upper left corner and the next # of bytes will be plotted following. Use this to redirect the computer to display graphics or text other than what is shown.
OLDROW	90	5A	Previous graphics cursor row.
OLDCOL	91,92	5B,5C	Previous graphics cursor column.
OLDCHR	93	5D	Data under graphics window cursor. Value of character under cursor used by OS to restore character when cursor is moved. To print the return character which normally can't be put on the screen: POKE 93,219:?" – ";
OLDADR	94	5E	Retains memory address of current visible text cursor location. Used to conjunction with OLDCHR to restore character value when cursor moves low byte.

NEWROW	96	60	Point (row) to which DRAWTO will go.
NEWCOL	97,98	61,62	Point (column) to which DRAWTO will go.
LOGCOL	99	63	Points at column in logical line. A logical line can contain up to 3 physical lines. This variable is used by display handler.
ADDRESS	100	64	Temporary storage holds contents of SAVMSC[$58]. and SAVMSC + 1[$59].??
MLTTMP	102	66	OPNTMP first byte used in open as temp.??
SAVADR	104	68	???
RAMTOP	**106**	**6A**	**Actual top of RAM memory given in number of pages (A page = 256 bytes of memory). This is often used to move where the computer thinks the top of memory is, thus saving a safe area for programs or data. You just POKE 106,PEEK(106) – # pages you want to save.**
BUFCNT	107	6B	Screen editor current logical line size.
BUFSTR	108	6C	Editor low byte.???
BITMSK	110	6E	???
SHFAMT	111	6F	Pixel justification???
ROWAC	112	70	Accumulator. Controls point plotting and increment and decrement functions.
COLAC	114	72	Controls column point plotting.
LOMEM	128,129	80,81	BASIC LO memory pointer (at the end of the Operating Systems RAM). Also starting here is the token output buffer which BASIC uses to convert your code into tokens, i.e., numbers that represent instructions. It is 256 bytes long. Same space used as stack by BASIC for executing expressions.
VNTP	130,131	82,83	Variable name table beginning address. This table holds the names of all variables that have been entered in your program. Each is stored in ATASCII in the order input with high bit on in last character in each name.
VNTD	132,133	84,85	Dummy end of the variable name table. BASIC uses this pointer to indicate the end of the name table. When there are less than 128 variables this normally points to a dummy zero byte. When 128 variables are present, this points to the last byte of the last variable name.
VVTP	134,135	86,87	Variable value table address. This table contains information on each variable, stored in 8 bytes for each variable.

These tables of your variables' names will fill up with names you are no longer using in your program. To clear them out, and save memory, list them to tape/disk; do a NEW; and ENTER the program back into memory. You can have up to 128 variable names.

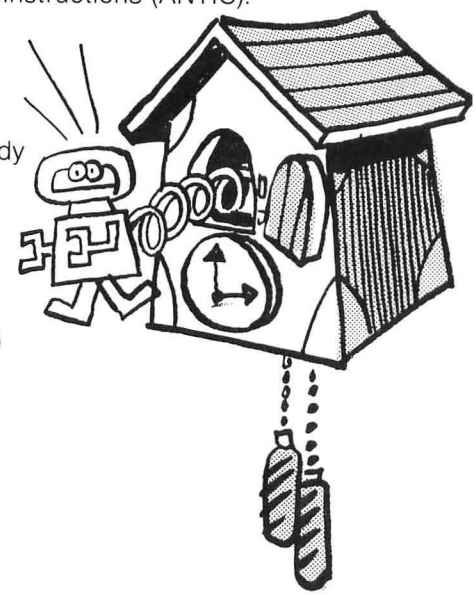
STMTAB	136,137	88,89	Statement table address. The table includes all the lines of code that have been entered by the user and tokenized by BASIC, and also the immediate mode line.

STMCUR	138,139	8A,8B	Current statement pointer for BASIC, used to reference particular tokens within a line of the statement table. When BASIC is waiting for input, this pointer is set to the beginning of the immediate mode line.
STARP	140,141	8C,8D	String/array table address. Also where your BASIC program ends. The table contains all string and array data. String characters only use one byte each. Arrays are stored as 6 byte BCD (binary coded decimal) numbers. This area grows as each dimension is interpreted by BASIC, thus reducing free memory.
RUNSTK	142,143	8E,8F	Run time stack address. This software stack contains GOSUB and FOR-NEXT entries. 4 bytes are used per GOSUB and 16 per FOR-NEXT.
MEMTOP	144,145	90,91	BASIC top of memory pointer. This is the end of the user PROGRAM. You still have room to expand until you reach the bottom of the Display List. FRE(0) is obtained by subtracting this from HIMEM.
?????	173,174	AD,AE	LIST pointer. Contains the high and low byte address of the line LISTed. If LIST alone had been typed, a 32767 would be here.
?????	183,184	B7,B8	Current number stored in Basic Data line pointer. This value is the line number from which data is currently being read. Could be used to list the line number where a read error occurred.
DATAOFF	185	B9	Offset in current DATA line.
STOPLN	**186,187**	**BA,BB**	**Line number where a STOP or TRAP occurred.**
ERRSAVE	**195**	**C3**	**Error number causing the STOP or TRAP to occur. Here is an example that uses these two locations:**

```
10 Trap 100
100 ?"ERROR # ";PEEK(195):LIST (PEEK(186) + 256*PEEK(187))
```

PTABW	**201**	**C9**	**Print tab width (default = 10). Set to other tab widths if wanted.**
FR0	212-217	D4-D8	Floating point register 0. (Value to be returned from USR function.)
FR1	224-229	E0-E5	Floating point register 1.
CIX	242	F2	Character index
INBUFF	243,244	F3,F4	Input text buffer pointer
RADFLG& DEGFLG	251	FB	Radians/degrees flag (Rad = 0, Deg. = 6)

VDSLST	512,513	200,201	Display List Interrupt (DLI) vector; you must POKE 54268 first before doing a DLI. DLI's are used to stop the normal flow of processing, and go do something else for a few microseconds, then return. For example, some games do many DLI's each time a screen is drawn in order to do music. This method is needed since a program might get hung up in a loop waiting for a joystick input and couldn't do music statements. The DLI continues since the screen is always being redrawn. DLI instructions must be in an assembly code that is vectored to by the Display List instructions (ANTIC).
VPRCED	514,515	202,203	Serial Proceed
VINTER	516,517	204,205	Serial Interrupt
VBREAK	518,519	206,207	Break Instruction vector
VKEYBD	520,521	208,209	Keyboard Interrupt vector
VSERIN	522,523	20A,20B	Serial I/O bus receive data ready
VSEROR	524,525	20C,20D	Serial I/O transmit ready
VSEROC	526,527	20E,20F	Serial bus transmit complete
VTIMR1	528,529	210,211	POKEY timer1 vector
VTIMR2	530,531	212,213	POKEY timer2 vector
VTIMR3	532,533	214,215	POKEY timer3 vector
VIMIRQ	534,535	216,217	IRQ Immediate vector (general)



System Timers

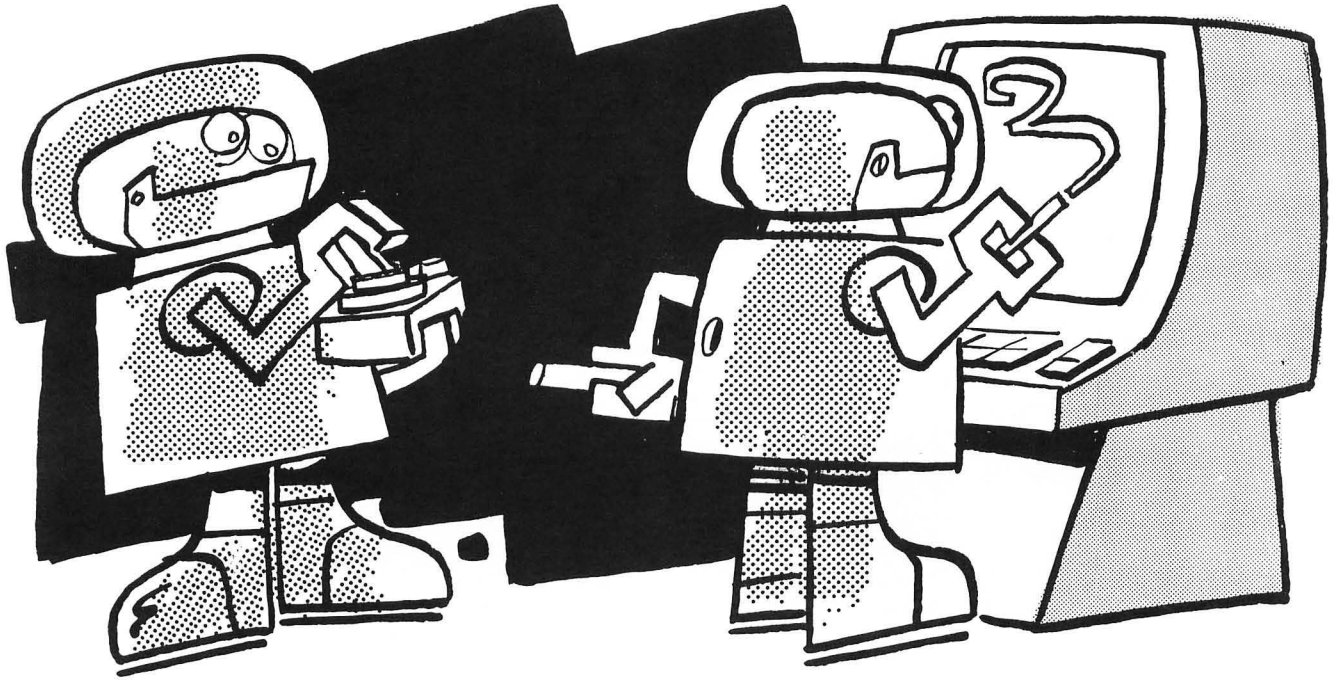
Accessed from assembly, some of these timers count backwards in 1/60th of a second intervals until they reach 0. For example POKE 540 with a 60, and then test for it to hold 0. After 1 second it will have counted BACKWARDS to 0. Use them to time music, a clock, or whatever.

CDTMV1	536,537	218,219	System timer 1 value
CDTMV2	538,539	21A,21B	System timer 2 value
CDTMV3	540,541	21C,21D	System timer 3
CDTMV4	542,543	21E,21F	System timer 4
CDTMV5	544,545	220,221	System timer 5
VVBLKI	546,547	222,223	VBlank Immediate jump address
VVBLKD	548,549	224,225	VBlank Deferred jump address
CDTMA1	550,551	226,227	System timer 1 jump address
CDTMA2	552,553	228,229	System timer 2 jump address
CDTMF3	554	22A	System timer 3 flag
CDTMF4	556	22C	System timer 4 flag
CDTMF5	558	22E	System timer 5 flag

BACK TO EASIER LOCATIONS!!!!

SDMCTL	559	22F	Direct memory access (DMA) enable. Use to turn off the screen display chip, the ANTIC, which will speed up the ATARI by 30% or more. (Remember to turn it back on or you won't see your results.) Original # = On; 0 and other #'s = Off. Save the actual number that is in here before you POKE with a 0. Then, to turn the screen back on, POKE in the SAVED number.

			Be sure to use a TRAP statement in case of errors . . . you can't see them with the display turned off! This is a shadow for 54272, so see that location for the correct values to use here for other uses.
SDLSTL	560,561	230,231	Display list pointer. Gives the starting address of the display list which, when known, you can then modify for many effects. See our Display List Tutorial.
SSKCTL	562	232	Serial Port Control.
LPENH	564	234	Light Pen Horizontal Value.
LPENV	565	235	Light Pen Vertical Value.
COLDST	580	244	1 = coldstart. 0 = normal. This is the same as turning the computer off and on. If you POKE 580,1; then press RESET, the coldstart occurs. Very useful to go run a program through AUTORUN.SYS on your disk. This will keep kids from crashing a disk based program since it will reboot!
GPRIOR	623	26F	SHADOW OF 53275. SEE 53275.

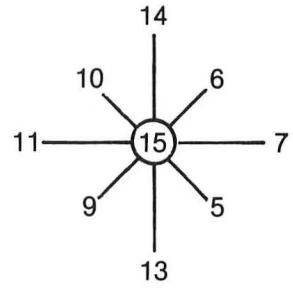


Paddles, Joysticks and Lightpen Controls

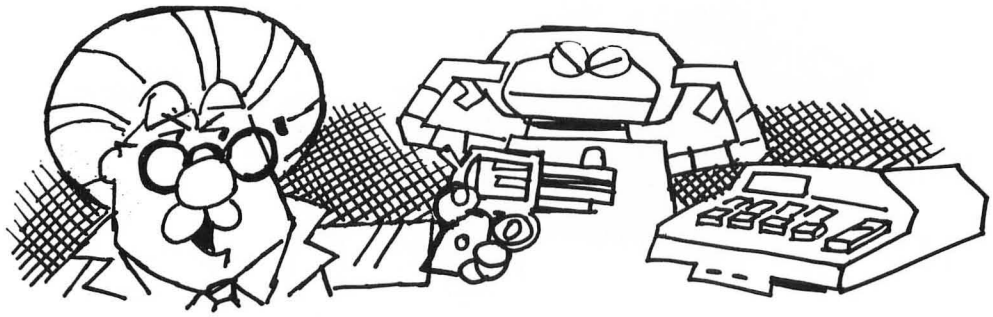
PADDL0	624	270	Value of Paddle 0 (if plugged in) 0 to 28. Use 0 to 39 and 201 to 228 to keep players off screen until needed.
PADDL1	625	271	
PADDL2	626	272	
PADDL3	627	273	
PADDL4	628	274	These are the same, but for other paddles.
PADDL5	629	275	
PADDL6	630	276	
PADDL7	631	277	

The following (632-635) are also used to read a lightpen switch if it's plugged into port 1-4:

STICK0	632	278	Value of joystick returned by: Print PEEK (632-5) if Stick 0-3 is in these positions:
STICK1	633	279	
STICK2	634	27A	
STICK3	635	27B	



PTRIG0	636	27C	Paddle Triggers: 0 = pressed
PTRIG1	637	27D	1 = not pressed
PTRIG2	638	27E	
PTRIG3	639	27F	
PTRIG4	640	280	
PTRIG5	641	281	
PTRIG6	642	282	
PTRIG7	643	283	
STRIG0	644	284	Joystick control trigger: 0 = pressed
STRIG1	645	285	1 = not pressed
STRIG2	646	286	
STRIG3	647	287	



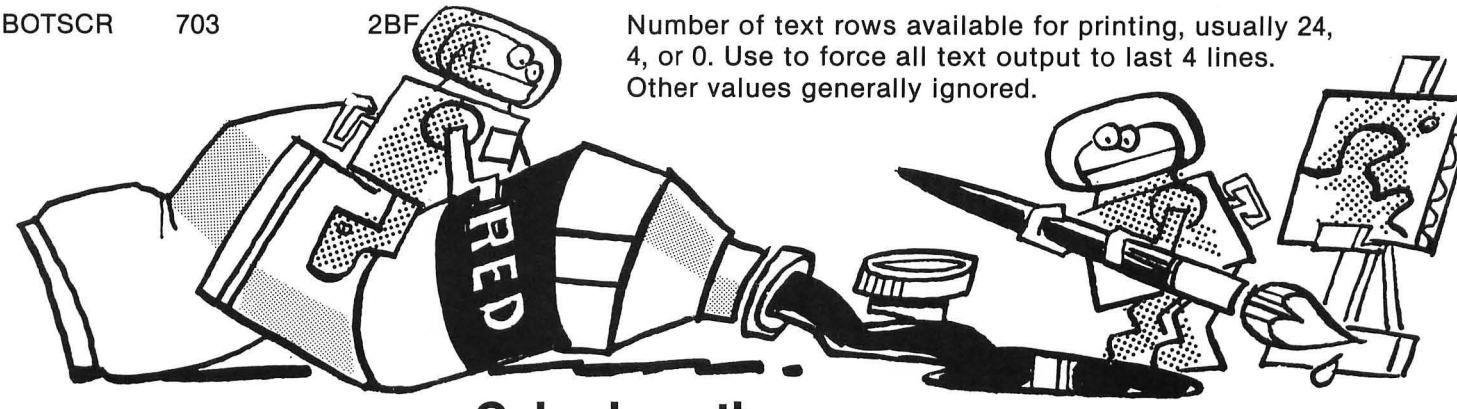
TXTROW	656	290	Text Cursor Row (0 to 3).
TXTCOL	657,658	291,292	Text Cursor Column (0 to 39). Location 658 should always be 0.
TXTMSC	660,661	294,295	Upper left corner of text window.
TABMAP	675-689	2A3-2B1	A 1 in any bit position of any of these 120 bits will act as a stop for the tab key. To clear all tabs, POKE each location with 0's. To set tabs, POKE in the decimal number that corresponds to the binary number that results when the desired bits are set. For example, to set a tab stop in Col. 6 on the screen, the binary number would be 00000100 = 4,SO = = =>POKE 675,4.

INVFLG	694	2B6	128 = Inverse Video (ATARI Key); 0 = Normal. POKE either of these if you want to be sure the next input will be either inverse or normal (prevents errors on input . . . other uses too). Computer will act like input keys are inverse or normal! Use this location with 702 as in this example:

```
100 POKE 694,128:POKE 702,0
```

This will make all input come out as inverse lowercase!

SHFLOK	702	2BE	0 = lower case letters; 64 = shift lock key pressed, i.e., capitals which is the normal mode; and 128 = control lock key pressed, i.e., special graphics characters. 255 = all letters will be ignored. What these values do is redirect the meaning of the keys. Note: takes effect upon the next input.
BOTSCR	703	2BF	Number of text rows available for printing, usually 24, 4, or 0. Use to force all text output to last 4 lines. Other values generally ignored.



Color Locations

To use the following color locations, use this formula:

POKE #,Z where # is the location number and:

Z = COLOR*16 + LUM

Color = 0 to 15 as in your BASIC Manual
Lum = 0 to 14 for luminescence (even #'s).

PCOLR0	704	2C0	Color of Player & Missile 0.
PCOLR1	705	2C1	Color of Player & Missile 1.
PCOLR2	706	2C2	Color of Player & Missile 2.
PCOLR3	707	2C3	Color of Player & Missile 3.
COLOR0	708	2C4	Color Register 0 (setcolor 0) (CAPITALS).
COLOR1	709	2C5	Color Register 1 (setcolor 1) (lowercase).
COLOR2	710	2C6	Color Register 2 (setcolor 2) (Inverse CAPITALS, Text Window, Borders).
COLOR3	711	2C7	Color Register 3 (setcolor 3 and Missile 4) (Inverse lowercase).
COLOR4	712	2C8	Color Register 4 (setcolor 4) (BACKGROUND).

For 708 to 712, "( )" applies to text colors when using GRAPHICS MODES 1 or 2.

RAMSIZ	740	2E4	Top of RAM address.
MEMTOP	741,742	2E5,2E6	Operating System Top of Memory Pointer.
MEMLO	743,744	2E7,2E8	Operating System Bottom of Memory Pointer.
DVSTAT	746	2EA	Device Status.
CRSINH	**752**	**2F0**	**Cursor inhibit, turns on/off cursor. Takes effect upon the next screen output (like a PRINT statement). 0 = on; not = 0 for off. Try this program:**

```
10 POKE 710,0:POKE 752,1:POKE 82,0:FOR I = 1 TO 959:?".";:NEXT I
20 GOTO 20
```

KEYDEL	753	2F1	(R) 0 = No key pressed. 3 = any key pressed.
CHACT	**755**	**2F3**	**Character Mode Register. 4 through 7 = upside down letters as in 0 to 3. 3 = inverse solid blocks. 2 = normal letters. 1 = show inverse video character. 0 = shows inverse video character as normal character. You can get attention in your programs by going between inverse and solid letters. Try this:**

```
10 POKE 755,4:?"ABC":POKE 755,2:GOTO 10
```

CHBAS	**756**	**2F4**	**Character Base Register: Default = 224 for upper case letters; 226 for lower case set of letters.**
ATACHR	763	2FB	Last ATASCII character read or written, or the value of the graphics point. Also the DRAWTO (as an XIO) command uses the value in this location to get the color for the fill command line.
CH	**764**	**2FC**	**Internal code of the last key pressed. POKE with 255 first to clear. Used to read keys without input statements and to wait for input. For example, POKE a 12 here and the return key would not have to be pressed. Note that "last key" means if you press "S" and then "A", only the code for the A is stored here. For example:**

```
100 POKE 764,255
110 IF PEEK(764) = 33 THEN 110
```

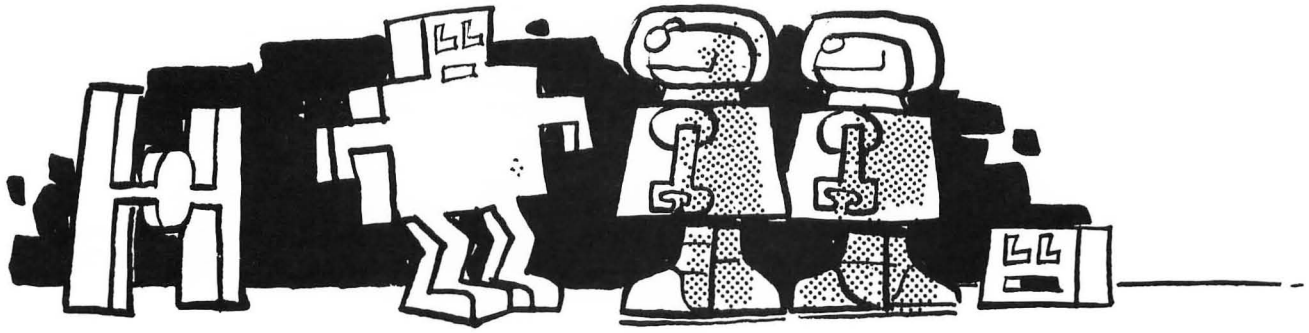
WHAT this will do is to wait for the letter "A" to be pressed.

FILDAT	765	2FD	Color for Graphics Fill (X10)
DSPFLG	766	2FE	Display Flag: 0 = normal; 1 = control character displayed (Ex: To display the symbol for "clear" page, an arrow to the left).
SSFLAG	767	2FF	Start/Stop flag for paging. 0 = normal. Set by CNTL – 1 keys pressed.

Disk I/O

DDEVIC	768	300	Device bus ID.
DUNIT	769	301	Disk Device Number, 1 to 8.
DCOMND	770	302	Disk Operation to be Performed.
DSTATS	771	303	Status Code Upon Return to User.
DBUFLO/H	772,773	304,305	Address of the Source or Destination of Disk Sector Data.
DTIMLO	774	306	Timeout Value for the Handler.
DBYTLO/H	776,777	308,309	Number of bytes transferred to or from disk as result of most recent operation.
DAUX1/2	778,779	30A,30B	Disk Sector Number to Read or Write.
HATABS	794-831	31A-33F	Handler Address Table (3 bytes/handler). A maximum of 12 entries. Contains the single character device name (K,P,E,S,C,D,) and the handler address for each entry. The rest of the 38 bytes are 0's. A handler is a set of instructions that tell the computer "handle" the screen, disk, or whatever.
IOCBO	832-847	340-34F	I/O Control Block 0. See page 0 IOCB above for notes on general useage.
			Examples of use:
ICPTL/H	838,839	346,347	POKE 838,166 and 839,238 and everything that would normally go to the screen will now go to the printer! (Or any other device.) The normal values for these locations to put things back on the screen are POKE 838,163;POKE 839,246.
ICAX1	**842**	**34A**	**13 = Read from screen; 12 = write to screen. These can be used or write input from/to the screen, for example, to read in line numbers and thus delete lines from a program while it's running! This is used in our Tricky Tutorial #1, Display Lists, to input new DATA statements without stopping the program. Neat!**
IOCB1	848-863	350-35F	I/O control block, IOCB 1.
IOCB2	864-879	360-36F	IOCB 2
IOCB3	880-895	370-37F	IOCB 3
IOCB4	896-911	380-38F	IOCB 4
IOCB5	912-927	390-39F	IOCB 5
IOCB6	928-943	3A0-3AF	IOCB 6
IOCB7	944-959	3B0-3BF	IOCB 7
LBUFF	1408	580	Text Buffer
?????	1801	709	The number of disk files that may be open at one time (normally 3).
?????	**1802**	**70A**	**The number of disk drives in your system goes here. Normally set to 2. If you have more or less you should change the number and then resave the modified DOS to your disks. For example, if you have drives 1, 2, and 4 the binary value is 00001011 which = 11 to POKE into this location.**

??????	1913	779	POKE with 80 to turn off write verify when doing disk copying. 87 turns back on. It's much faster, but an occasional error may occur, so use at your own risk. After you POKE this location, you could re-write DOS and thus have the verify off every time you use this copy of DOS.
?????	3118	C2E	0 will cause only the first of two files with the same name on your disk to be deleted if you try to use the DELETE command from DOS to get rid of only ONE. 184 will restore normal DOS usage.
?????	42082	A462	Search list for common name or abbreviation.
?????	43508	A9F4	Test whether BREAK key has been pushed.



Player Missile Graphics Registers

We want to tell you of a feature of the ATARI called "shadowing". Shadowing comes from the fact that many of the following locations can't be written to directly. They are "Hardware Registers" since these locations reflect the state of certain pins on chips within the computer, i.e., the hardware. Anyway, we have marked these locations with a "(R)" if it can be read and a "(W)" if you can write to it. The shadowing comes in because for each location that you can't write to, there is a Shadow at another location that you can write to. The value in the Shadow is placed into the hardware register every time a frame is drawn on the screen . . . 60 times a second! This is why you can't write to it; the Shadow value would replace anything you wrote after 1/60th second. What to do? Well, we put the location to write to in "( )" marks. For example, to turn off the screen you want a 0 in DMACTL, 54272. Just POKE 559,0 instead, since 559 is its shadow.

To really use Player/Missile Graphics, you should either get our tutorial on Player Missile Graphics, or see articles on the subject in magazines. Warning!! Most articles give only the briefest of introductions to using ATARI graphics. Our tutorials give you enough to create your own games, but not of the quality of Star Raiders, which is in assembly language. We are talking about BASIC games and simple business applications.

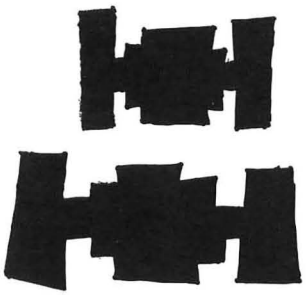
Finally, note that in many of these locations you read one thing from it (like a collision), and write something else into it (like a horizontal position). Collisions mean that two things, say a player and the background, a maze shape, have touched or overlapped. Priority allows one shape to appear in front of the other (only one color shows). Horizontal position registers instantly move players across the screen. The size of PLAYERS:

0 = normal (2 color clocks wide)
1 = double (4 color clocks wide)
2 = normal (2 color clocks wide)
3 = quadruple (8 color clocks wide)

AND FOR COLLISIONS:

VALUE READ	COLLIDED WITH PLAYER OR PLAYFIELD NUMBER
0	0
1	1
2	2
4	3

HPOSP0 M0PF	53248	D000	(W) Horizontal Position of Player 0 (R) Missile 0 to Playfield Collision
MPOSP1 M1PF	53249	D001	(W) Horizontal Position of Player 1 (R) Missile 1 to Playfield Collision
HPOSP2 M2PF	53250	D002	(W) Horizontal Position of Player 2 (R) Missile 2 to Playfield Collision
HPOSP3 M3PF	53251	D003	(W) Horizontal Positon of Player 3 (R) Missile 3 to Playfield Collision
HPOSM0 P0PF	53252	D004	(W) Horizontal Position of Missile 0 (R) Player 0 to Playfield Collisions
HPOSM1 P1PF	53253	D005	(W) Horizontal Position of Missile 1 (R) Player 1 to Playfield Collisions
HPOSM2 P2PF	53254	D006	(W) Horizontal Position of Missile 2 (R) Player 2 to Playfield Collisions
HPOSM3 P3PF	53255	D007	(W) Horizontal Position of Missile 3 (R) Player 3 to Playfield Collisions
M0PL SIZEP0	53256	D008	(R) Missile 0 to Player Collisions (W) Size of Player 0
M1PL SIZEP1	53257	D009	(R) Missile 1 to Player Collisions (W) Size of Player 1
M2PL SIZEP2	53258	D00A	(R) Missile 2 to Player Collisions (W) Size of Player 2
M3PL SIZEP3	53259	D00B	(R) Missile 3 to Player Collisions (W) Size of Player 3
P0PL SIZEM	53260	D00C	(R) Player 0 to Player Collisions (W) Sizes for all missiles



for Missile #	NORMAL	DOUBLE	QUAD
0	0	1	3
1	0	4	12
2	0	16	48
3	0	64	192

For example, to have Missile 0 be double size and Missile 2 be normal and Missile 3 be quadruple size:

POKE 53260,(1 + 0 + 192)

GRAFP0	53261	D00D	(W)	Graphics for Player 0 (shape of player)
P1PL			(R)	Player 1 to Player Collisions
GRAFP1	53262	D00E	(W)	Graphics for Player 1 . . . You place the shape of players in GRAFP# (1, 2, 3, 4 or M) registers only if you are not using DMA (GRACTL)
P2PL			(R)	Player 2 to Player Collisions
GRAFP2	53263	D00F	(W)	Graphics for Player 2
P3PL			(R)	Player 3 to Player Collisions
GRAFP3	53264	D010	(W)	Graphics for Player 3
TRIG0			(R)	Joystick Trig 0 (644)
GRAFPM	53265	D011	(W)	Graphics for all missiles
TRIG1			(R)	Joystick Trig 1 (645)
COLPM0	53266	D012	(704)	Color and LUM of Player/Missile 0
TRIG2				Joystick trigger 2(646)
COLPM1	53267	D013	(705)	Color and LUM of Player/Missile 1
TRIG3				Joystick trigger 3(647)
COLPM2	53268	D014	(706)	Color and LUM of Player/Missile 2
COLPM3	53269	D015	(707)	Color and LUM of Player/Missile 3
COLPF0	53270	D016	(708)	Color and LUM of Playfield 0
COLPF1	53271	D017	(709)	Color and LUM of Playfield 1
COLPH2	53272	D018	(710)	Color and LUM of Playfield 2
COLPH3	53273	D019	(711)	Color and LUM of Playfield 3 and Missile 4
COLBK	53274	D01A	(712)	Color and LUM of background
PRIOR	53275	D01B	(W)	Priority select: to choose which of the objects on the screen will appear to be in front of the others, i.e., a player for example, might pass over a star and this register controls if the star will not be seen (player in front) or if the player will be partly covered up (star in front). Choose desired options and POKE INTO 623, THE SHADOW LOCATION. PF# = playfield #0 to 3. P# = player #0 to 3. BAK = background.

	ADD	
To use GRAPHICS 11	192	
To use GRAPHICS 10	128	GTIA ONLY!
To use GRAPHICS 9	64	

(CHOOSE ONLY ONE OF ABOVE OR NONE AT ALL)

For overlapped areas of players to have a third color	32
To use all 4 missiles as a 5th player	16 (color is COLPF3)

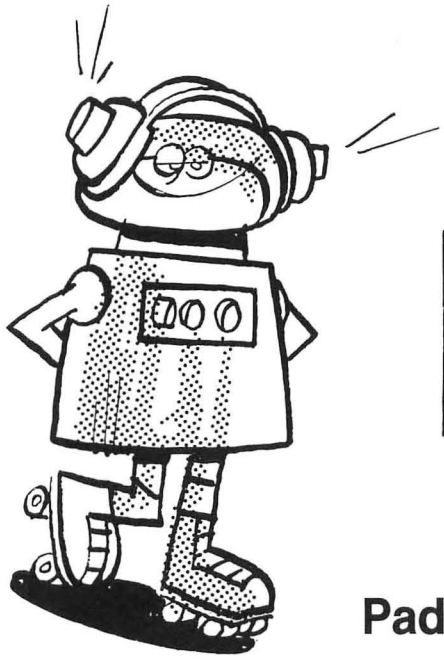
Priorities in this order:

PF0,PF1, PO-P3, PF2-3, background	8	⌐
PF0-3, PO-3, BAK	4	├ choose only one
P0-1, PF0-3, P2-P3, BAK	2	├
P0-P3, PF0-3, BAK	1	⌙

Example: 5th Players = 16 + Last Priority Choice = 1 ---> POKE 623,17

VDELAY	53276	D01C	(W)	Vertical DELAY

GRACTL	53277	D01D	(W) Used with DMACTL (below) to 1) latch triggers (remember if triggers have been pressed), 2) turn on players, 3) turn on missiles. To get the value to POKE here. For Missile DMA, add 1 For Player DMA, add 2 To latch the trigger inputs add 4.
HITCLR	53278	D01E	(W) POKE with any # to clear collision registers.
CONSOL	53279	D01F	(W/R) Use to see if the special consol switches are pressed. Numbers are those that will result if you print PEEK(53279). POKE with 8 first before reading this location to clear. POKE a number (0 to 7) into this location to "click" the speaker. X means that if you press the keys indicated, the number shown is placed in this register.



Switch	1	2	3	4	5	6	7	0
Start		X		X		X		X
Select	X			X	X			X
Option	X	X	X					X

Paddles and Audio Controls

The AUDF# locations correspond to the pitch for the sound 0 to 3 channels. Use the values of the notes (0 to 255) as in your BASIC manual. The AUDC# are the volume controls. These may be POKEd to learn their effect or see our SOUND Tutorial:

Pots are the values read from paddles 0 to 7 plugged into the front ports. The values read range from 0 to about 228. When using players and missiles, a value less than 40 or more than 200 represents an area off screen to either side. The sum of the volumes of the four channels should not exceed 32. If you POKE in sound, rather than using the SOUND command, first initialize with POKEs of 0 to locations 53768 and 3 to 53775.

AUDF1	53760	D200	(W) Audio Channel 1 Frequency
POT0			(R) Pot 0 (Paddle)(624)
AUDC1	53761	D201	(W) Audio Channel 1 Control
POT1			(R) Pot 1 (625)
AUDF2	53762	D202	(W) Audio Channel 2 Frequency
POT2			(R) Pot 2 (626)
AUDC2	53763	D203	(W) Audio Channel 2 Control
POT3			(R) Pot 3 (627)
AUDF3	53764	D204	(W) Audio Channel 3 Frequency
POT4			(R) Read Pot 4 (628)
AUDC3	53765	D205	(W) Audio Channel 3 Control
POT5			(R) Read Pot 5 (629)
AUDF4	53766	D206	(W) Audio Channel 4 Frequency
POT6			(R) Read Pot 6 (630)

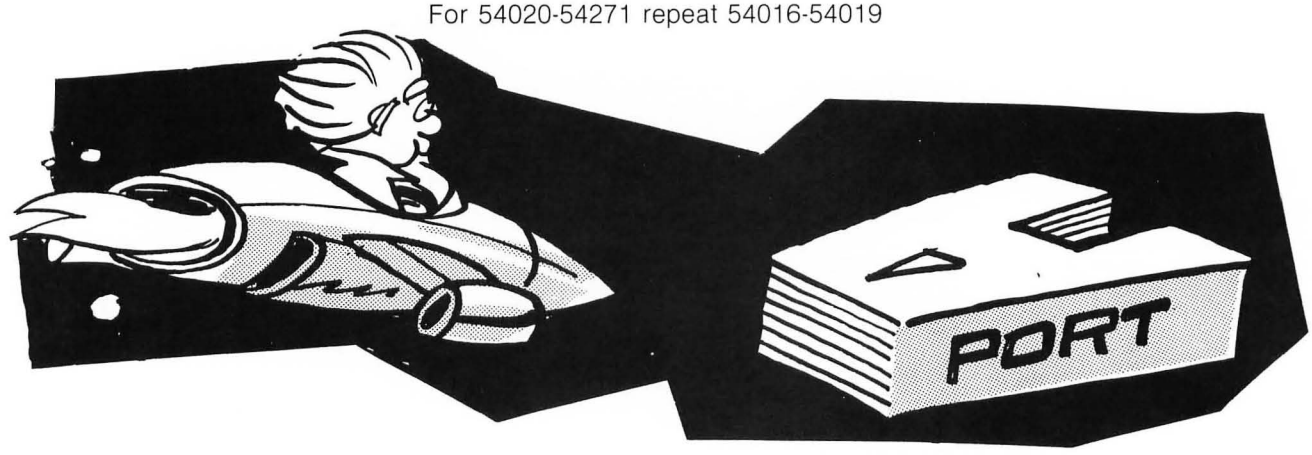
AUDC4	53767	D207	(W) Audio Channel 4 Control
POT7			(R) Read Pot 7 (631)
AUDCTL	53768	D208	(W) Audio Control
ALLPOT			(R) 8 Line Pot Port State

KBCODE	53769	D209	(R) Keyboard Code (764)
STIMER			(W) Start Timer (reset audio-frequency dividers to AUDF values).
RANDOM	53770	D20A	(R) Random Number Generator.
SKREST			(W) Reset Serial Port Status, 53775.
POTGO	53771	D20B	(W) Start POT Scan Sequence.
SEROUT	53773	D20D	(W) Serial Port Output.
SERIN			(R) Serial Port Input.
IRQEN	53774	D20E	(W) Interrupt request enable. If you turn this off, the keys that do interrupts won't work. 0 = Off; PEEK(53774) = On. Also POKE location 16 to disable break key.
SKCTL	53775	D20F	(R) (562) Serial port control. 3 = stops occasional noise after input/output from cassette. This location will hold 251 if most any key is pressed, 255 if no key is pressed, and 247 if shift is pressed.
SKSTAT			(R) Read serial port status.

For 53776-54015 repeat 53760-53775

PORTA	54016	D300	(R/W) Reads or writes data from controller jacks 1 & 2 if bit 2 of PACTL = 1. Writes to direction control if bit 2 of PACTL = 0. (632 for jack 1 and 633 for jack 2.)
PORTB	54017	D301	(R/W) Reads or writes data from controller jacks 3 & 4 as above. (634 for jack 3 and 635 for jack 4.)
PACTL	**54018**	**D400**	**(W) Port A controller . . . 60 turns cassette motor off, 52 turns it on. Use this to do program control of music or voice, or you could interface this to a light or other type of control (home heater, etc.).**
PBCTL	54019	D303	Port B controller

For 54020-54271 repeat 54016-54019



DMACTL	54272	D400	(W) (559) Direct Memory Access control (DMA). Turns of DMA, gives 1 or 2 line resolution to players, turn on player/missiles, choose from the following options and add the total to get the value to POKE into 559:

For	Add	
Wide Playfield	3	Choose only one
Standard Playfield	2	
Narrow Playfield	1	
No Playfield	0	
Enable Missile DMA	4	
Enable Player DMA	8	
2 Line "Thick" Players, or	0	
1 Line "Thick" Players	16	
Enable Instruction Fetch DMA	32	

Better see our Tutorial on Player Missile Graphics to understand this.

CHACTL	54273	D401	(W) (755) Character control. 4 = upside-down letters; 2 = like ATARI key; 1 = blank letters . . . blinking letters can be done by toggling these values.
DLISTL/H	54274/5	D402/3	(560/1) Display List pointer. This address will tell you where the OS has placed the instructions to the computer as to what modes and what data to put on the screen. Yes, we do have a Tutorial on the subject!
HSCROL	54276	D404	(W) Horizontal scroll enable . . . long explanation . . . see Tutorial #2. Basically you POKE with 0 to 16 clock cycles.
VSCROL	54277	D405	(W) Vertical Scroll enable (see Tutorial #2). This one gets POKEd with 0 to 16 scan lines. Both scrolls only take place if the Display List has been modified.
PMBASE	54279/80	D407/8	(W) Player missile base address. You will use this often to locate where to place your Players and Missiles. See our tutorials or recent magazines. A little tricky to use.
CHBASE	54281	D409	(W) (756) Character base address. The location of the start of your character set can be changed from the standard ATARI characters to your own custom set. See a program by IRIDIS or our new tutorial.
WSYNC	54282	D40A	(W) Wait for horizontal sync. Simply accessing this location halts the CPU until horizontal sync. occurs.
VCOUNT	54283	D40B	(R) Vertical line counter. . . In assembly programs this is used to keep track of where the picture is currently being generated (for color or graphics changes).
PENH	54284	D40C (564)	(R) Light pen horizontal position.
PENV	54285	D40D (565)	(R) Light pen vertical position.

NMIEN	54286	D40E	(W) Non-maskable interrupt enable 192 will allow a Display List interrupt.
NMIRES	54287	D40F	(W) Reset NMIST.
NMIST			(R) NMI status.

For 54288-54277 repeat 54272-54287

Floating Point Package (ROM) Entry Points

AFP	55296	D800	ASCII to FP convert.
FASC	55526	D8E6	FP to ASCII convert.
IFP	55722	D9AA	Integer to FP convert.
FPI	55762	D9D2	FP to integer convert.
ZFRO	55876	DA44	Clear FRO.
ZFI	55878	DA46	Clear FP number.
?????	55889	DA51	Load INBUFF (243) with address of Text Buffer (1408).
FSUB	55904	DA60	FP subtract.
FADD	55910	DA66	FP add.
FMUL	56027	DADB	FP multiply.
FDIV	56104	DB28	FP divide.
?????	56239	DBAF	Check character in Text Buffer for a numeric digit.
?????	56251	DBBB	Check string in Buffer for a number with possible +, −, or decimal.
?????	56255	DBA1	Skip spaces in Text Buffer.
PLYEVL	56640	DD40	FP polynominal evaluation.
FLD0R	56713	DD89	Load FP number.
FLD0P	56717	DD8D	Load FP number.
FLD1R	56728	DD98	Load FP number.
FLD1P	56732	DD9C	Load FP number.
FSTOR	56743	DDA7	Store FP number.
FSTOP	56747	DDAB	Store FP number.
FMOVE	56758	DDB6	Move FP number.
EXP	56768	DDCO	FP base e exponentiation.
EXP10	56780	DDCC	FP base 10 exponentiation.
LOG	57037	DECD	FP base e logarithm.
LOG10	57041	DED1	FP base 10 logarithm.

Handlers

Base addresses for handler vectors for the resident handlers:

Screen editor (E) = = = = = = = = = =	E400
Display handler (S) = = = = = = = = =	E410
Keyboard handler (K) = = = = = = = =	E420
Printer handler (P) = = = = = = = =	E430
Cassette handler (C) = = = = = = =	E440

O.S. Jump Instruction Addresses

DISKIV	58448	E450	Disk handler initialization.
DSKINV	58451	E453	Disk handler entry.
CIOV	58454	E456	CIO utility entry (often used).
SIOV	58457	E459	SIO utility entry.
SETVBV	58460	E45C	Set system timers routine.
SYSVBV	58463	E45F	Stage 1 VBLANK entry.
XITVBV	58466	E462	Exit VBLANK entry.
SIOINV	58469	E465	SIO utility entry.
SENDEV	58472	E468	Send enable routine.
INTINV	58475	E46B	Interrupt handler initialization.
CIOINV	58478	E46E	CIO utility initialization.
BLKBDV	58481	E471	Blackboard mode entry.
WARMSV	58484	E474	Warmstart entry (reset button) POKE 580, previous USR to this: X = USR(58484).
COLDSV	58487	E477	Coldstart entry (powerup).
RBLOKV	58490	E47A	Cassette read block vector.
CSOPIV	58493	E47D	Cassette open for input vector.

Miscellaneous Notes

Many of these locations are beyond the needs of most of us, but if you have looked at the list and think you made a mistake in buying it, don't . . . if you will just try and use some of the values and ask other programmers questions, you will soon find you can do some very nice things on your ATARI.

Please write to us with any corrections you may find or to place an order.

EDUCATIONAL SOFTWARE, INC.
5425 Jigger Dr.
Soquel, CA 95073
(408) 476-4901

Want great sound from your programs? Just plug your ATARI into a stereo. Here's how. Look at the side of your ATARI 800 (sorry 400 owners) and find the socket marked "monitor". It takes a standard 5 pin DIN plug available at many stereo, TV, and electronics shops like RADIO SHACK. Besides the plug, buy a 2-wire cable with an RCA plug at one end. It doesn't matter what is on the other end as you must cut this off and attach it to the DIN plug. You want the inner wire (signal) of your cable to attach to the pin that will plug into the monitor plug's #1 hole, which is all the way to the left as you look at it. The other wire, the ground or shield, goes to the #3 hole, which is in the center of the monitor plug's five holes.

Now just plug in your cable to both the ATARI and the "AUX" input on most stereos. If you don't have a stereo-mono switch, you can splice two RCA plugs to the same end to get the sound signal to both left/right inputs of your stereo. You will now hear music and game sounds like never before!

For those of you with disk drives made before August 1981, here is a useful modification. The early drives had, among other problems, trouble reading disks that were recorded on machines that ran at slightly different speeds than their own speed. Newer drives have what's called a Date Separator, which seems to get rid of this problem. ATARI will upgrade on your older drives if needed, but the cost will depend on your debating abilities. A company called Percom also sells a separator for ATARI drives.

BASIC Hints

First, here are a few tips on how to speed up your BASIC programs and also save memory:

Put both subroutines and FOR-NEXT loops that are often used at the front of a program. Whenever BASIC has to look for a line number, like at the start of EVERY loop in a FOR-NEXT look, it starts at the beginning of the program and works its way down. This takes up a LOT of time.

If a constant is used often, give it a name as in "CONST = 12.3" when first needed, then just refer to it by name everywhere else. This saves memory. Some programs you will buy use constants like Z1 = 1, Z2 = 2, Z10 = 10, etc., this can be confusing to read.

Put many statements (10 ?"HI":POKE 84,6:?"BYE") on the same line using colons to separate them. This also saves memory space and time.

If you are going to have the computer do something that takes longer than a few seconds, print out a message to "Please Wait" then POKE off the screen display (POKE 559,0). The processor will run up to 30% faster. Then POKE back the original number (usually 34) into 559 to turn the screen on again.

If you can't turn the screen off, using a graphics or text mode that requires less memory will save the processor much time, thus allowing it to run faster. The less memory it has to put on the screen, the faster it goes. See BASIC Manual for memory requirements.

When you write a program, you may not use every variable that you started with. Even though you take them out, if you ever ran the program with them, they are still taking up room in the variable name and value tables. They are removed by LISTing the program out to tape or disk, type NEW, then ENTERing it back in again.

If you are outputting the same text message over and over, store it in a string and output the string instead of having the message several times in your program.

With frequently used GOSUB's and GOTO's, have them refer to a variable that has been defined. For example:

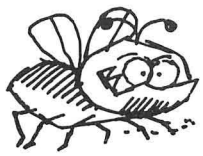
```
10 PLACE = 100
20 FOR I = 1 TO 1000
30 GOSUB PLACE
40 NEXT I
50 END
100 PRINT "HELLO 100 TIMES"
101 RETURN
```

Replace SETCOLOR commands with POKE commands.

For advanced users, it is possible to have the program actually delete lines of code after it is running, so you could delete many lines before dimensioning variables, etc. This trick uses location 842 (look in our Tutorial #1).

Finally, you can chain programs to run each other.

These are only some of the many unique things you can do to save memory space and time using the ATARI's O.S. Most BASIC manuals give many other things you can do to write good programs.



Bugs in ATARI BASIC

Yes, both the BASIC cartridge and the Operating System (OS) have problems. Here are some of them:

1) An input statement without an associated variable doesn't cause an error message, but can cause the computer to lock up if run.
2) Sometimes, if you do a lot of editing, you will find the machine suddenly either has lost some of your program, or no longer responds to any input (this is called "going to sleep").
3) String assignments that involve the movement of multiples of 256 bytes do not move the first 256 bytes. This refers to the internal length, not the number of characters. Since most strings don't come out exactly this number, you will seldom have this problem. If a problem occurs, just add one character.
4) PRINT A = NOT B puts the computer to sleep.
5) You must use the LPRINT command from direct mode BEFORE doing a SAVE"C" or CSAVE. Leave any printers you have OFF. This is because the cassette handler doesn't always set up the hardware properly for output. You will get an error message which you may ignore.
6) Don't type in a program line longer than 3 screen lines unless you want the excess beyond 3 lines to be taken as the next line in your program. Other strange things may happen as well.
7) Many exponents don't evaluate exactly. For example, 5/3 = 124.999998, not 125. The lack of accuracy will mostly effect comparisons where an exact number is expected. Your program can look for a small range of numbers instead.
8) A printed CNTL R or CNTL U is treated as a semicolon.
9) Watch out for the use of the letters "NOT" at the beginning of variable names.
10) LOCATE and GET do not reinitialize their buffer pointer. This can cause your program to change when next run (for example, some line numbers may change). To fix, reinitialize the pointers (by using a STR$ call like:) A = STR$(0); or you can print a numeric value like: Print A.
11) An input of more than the standard 128 bytes will write them into the so called safe region in page six of memory ($0600 to $067F). Since many programs store their assembly routines in page six, you must be careful to relocate these routines if longer input is a must.

Bugs in the O.S.

Yes, but please realize how good the operating system is before you feel bad about a few bugs. Here are the ones that ATARI has fixed in revision "B", which should be in machines shipped from ATARI after about DECEMBER 1981 (this date is not exact!).

1) During disk input and output, the disk drives would occasionally "time out" for several seconds, then start up again (this one caught me MANY times). The problem is fixed with no impact on your old software.
2) Under certain I/O conditions, the TV display would go away. This is also fixed.
3) Sometimes you would get an error message "device timeout error", "ERROR 138", which was false, but you had no way of knowing this. Problem fixed.
4) POKEY timer #4 IRQ vector is now working. This would be for advanced users.
5) Sending the SIO utility a buffer address ending in $FF caused SIO to loop forever.
6) A vector for the BREAK key has been added. This means programs may now use the BREAK key to do something other than stop a BASIC program from running. The vector address is at $236,$237 and bytes at $28B-$28D are used by the IRQ interrupt handler as temporary storage registers.

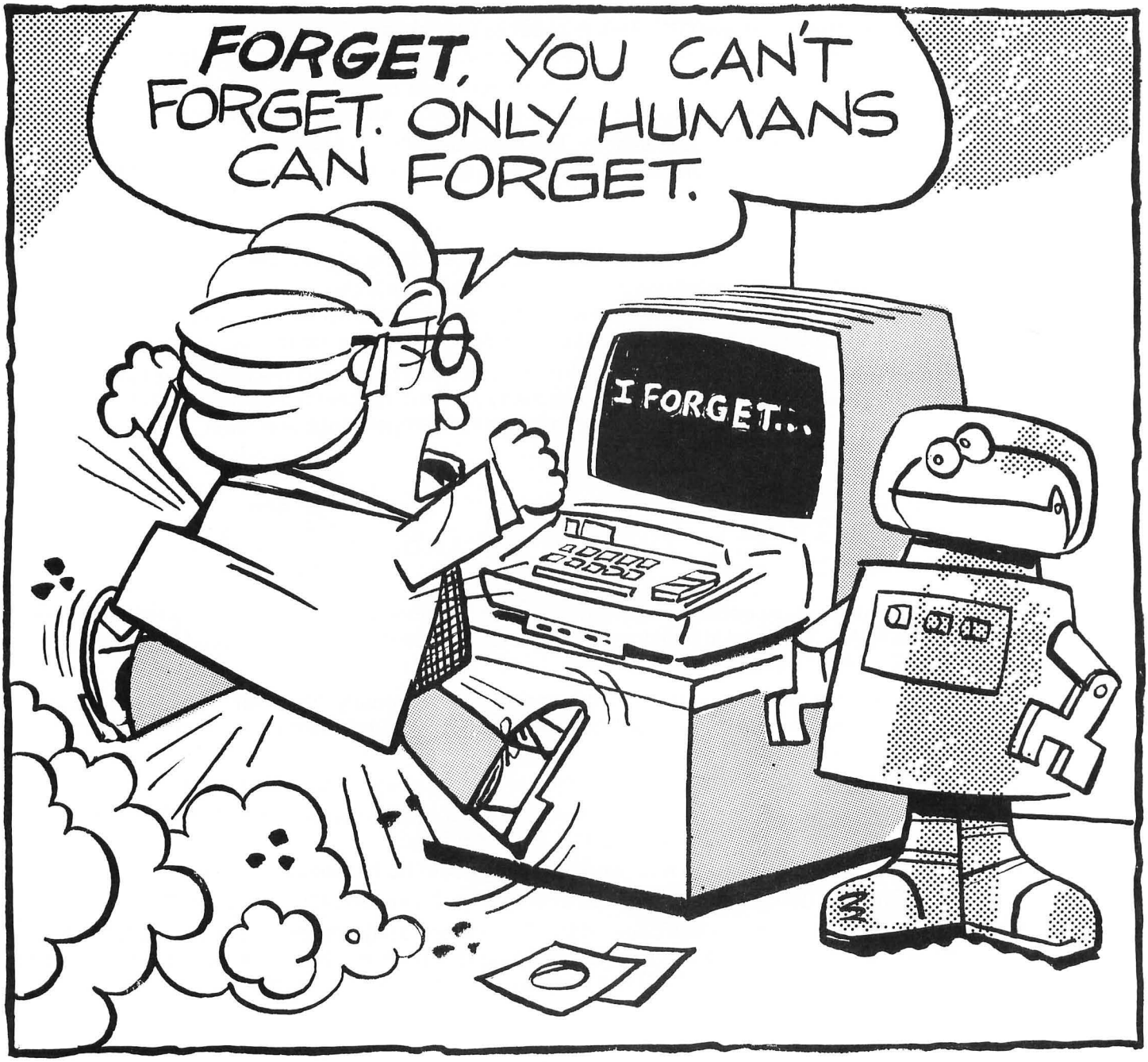
Notes on the O.S.

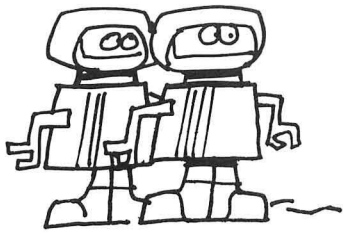
The IRQ interrupt handler has been completely recoded, although it is functionally identical. Illegal entry points have been eliminated.

The disk drive time out problem was solved by changes in the SETVBL routine (sets vertical blank vectors and timers).

The two routines modified to fix the incorrect "ERROR 138" messages were SETVBX and SETVBL.

ATARI says that assembly programmers who make use of the above information should be sure to enter the OS only through $E450 to $E480 as shown above in the MEMORY MAP. I personally have programs from very well known companies that no longer run because they entered the OS routines where they shouldn't have. If you find some of your programs don't work, the problem may be in software designed for the old OS. Check with the software vendor. ATARI has done a fine job of correcting the few bugs that were in the OS.





GTIA Chip

Well, that's about it for the MEMORY MAP. This started out as a normal memory map with just locations and what they were for. Then, I started adding new locations from various sources including readers of the MASTER MEMORY MAP (for which they were rewarded with free software!). Next came the hints which would make the ATARI more usable to the new owner. Finally, for this revision, many more advanced locations were added along with a lot of general notes. You may have seen many of the locations in this document before, but think how long it would take to write all this down. Please help me keep the price low by NOT COPYING THIS FOR YOUR FRIENDS. Besides being illegal, it's a rotten thing to do to a fellow ATARI enthusiast. On the other hand, I am legally obligated to say the following: "ATARI is a registered trademark of Warner Communications, Inc.". There, now that that is done with here's your bonus #47:

ATARI now is shipping computers with a different chip called GTIA (George's Television Interface Adapter). The old chip is called CTIA. It controls the graphics modes available to you. GTIA adds modes 9, 10 and 11. Older computers will be able to get an upgrade. Call 800-538-8737. Cost is about $62.50 installed.

All of your programs will still run with the GTIA. The old CTIA used only the four color registers that correspond to the setcolor command. The GTIA uses all nine color registers within the ATARI, the additional ones being those used for the four players and the one for the missiles when used as a fifth player. This gives you NINE COLORS or 16 hues with one luminance, or 16 luminance levels of one hue. ALL from BASIC!

In addition to the above you can still add Players and Missiles. Your regular BASIC Cartridge supports modes 9 to 11 even though your manual doesn't say so. Here is a brief description of each.

GRAPHICS 9

—This mode produces up to 16 different brightness levels of the same hue. This allows 3-D effects where you can shade an object (it looks great).

GRAPHICS 10

—All nine colors are allowed as mentioned above each with different brightness levels. You address them as COLOR 0 to 8. Actually the chip will allow you to call for up to COLOR 15, but since there are only the nine actual color registers in the computer's hardware, such a call will get you one of the first nine colors.

GRAPHICS 11

—The opposite of mode nine in that you can have 16 different hues with the same brightness. You set the brightness with:

```
SETCOLOR 4,0,luminance #.
```

The hue is called similar to the following:

```
FOR I=0 to 15:COLOR I
PLOT 4,I+10:NEXT I
```

Okay, so that is all the good stuff about the GTIA. What about the proverbial bad news. Well, nothing about the new chip is really bad. Software people won't write programs for it until there are a lot of them in production, and then these new "colorful" programs won't run correctly on older machines. Also, the new modes all use the same memory as graphics 8 (7900 bytes), yet they have a resolution of only 80 horizontally by 192 vertically. This resolution will be fine, however, for solid drawing with lots of colors, i.e., GAMES!!

Hex Conversion Chart

Column #	4th	3rd	2nd	1st	# HEX
	4096	256	16	1	1
	8192	512	32	2	2
	12288	768	48	3	3
	16384	1024	64	4	4
	20480	1280	80	5	5
	24576	1536	96	6	6
	28672	1792	112	7	7
	32768	2048	128	8	8
	36864	2304	144	9	9
	40960	2560	160	10	A
	45056	2816	176	11	B
	49152	3072	192	12	C
	53248	3328	208	13	D
	57344	3584	224	14	E
	61440	3840	240	15	F

Examples:

Hex to Decimal

	4th	3rd	2nd	1st	
$ 4A3F =	4	A	3	F	
					15
					48
					2560
					16384
					TOTAL = 19007 (dec)

Decimal to Hex

9304		
– 8192	(largest # less than 9304 on chart, 4th column) →	2
1112		
– 1024	3rd column →	4
88		
80	2nd column →	5
8	1st column →	8 = $2458

FOR BEGINNERS OR EXPERTS THIS BOOK IS FOR YOU!

BKA066
MASTER MEMORY MAP (THIN)
BROWN BOOK

$4.95

The MASTER MEMORY MAP™ is rapidly becoming the ndard reference book for owners of ATARI computers. After you ve bought this book, you will understand why. WHAT? You want know how. OK, I'll tell you. The MMM, as I lovingly call it, was itten by the best qualified person in the whole world, ME! I am Professor Von Chip, your instructor for learning to use all the power built into your computer.

The 100's of locations I talk about inside offer you almost everything you will want to know about your computer,

EVEN IF YOU DON'T PROGRAM!

Don't forget to ask for my other lessons teaching you about all of the great GRAPHICS and SOUND tricks that the ATARI computers can do: the TRICKY TUTORIALS™ for 16K:

#1 — DISPLAY LISTS (many Graphics Modes at the same time!)
#2 — SCROLLING (move your Graphics and text around smoothly)
#3 — PAGE FLIPPING (a professional looking way to redraw many screens of text or graphics)
#4 — BASICS OF ANIMATION (a beginner's lesson in moving shapes around the screen)
#5 — PLAYER MISSILE GRAPHICS (Learn to write a PACMAN™ type game of your own!)
#6 — SOUND AND MUSIC (from single notes & chords to songs & special effects, I explain all)
#7 — DISK UTILITIES (Utility programs to help you use your disk drives (32K)
#8 & 9 Coming soon to a blackboard near you!