1979

$9.95

THE APPLE II MONITOR PEELED

C A L L S   P E E K S   P O K E S

B Y   T O P I C

ALL EXPLAINED IN ENGLISH

LOCATIONS IN HEX AND DECIMAL

by
William E. Dougherty

FOREWARD

Second printing, May 1979

While writing programs for my APPLE II, I have many times had to stop constructive work to delve into the Monitor to determine (or redetermine) how to make use of a particular function or feature. Being totally unsuccessful in finding a single PEEK, POKE, CALL reference publication in the marketplace, and having delved deeply into a dozen computers in the last twenty years, I decided to put together for myself, and maybe for others, a description of the ROM contents of the APPLE II in an organization by subject instead of organization by machine address. With a lot of encouragment from my associates who have APPLEs, I decided to go beyond the organized notes I needed for my purposes and actually finish it for publication, describing functions and features for readers to whom it would be new information instead of just making up charts of addresses with a few cryptic comments to refresh my memory.

Although the listed CALL points in the Monitor are most useful to the machine language programmer, very many are also useful to BASIC or APPLESOFT programmers as well. For example, keyboard input of single strings which happen to contain commas and cassette tape input and output can be accomplished by the methods described within. While programming in Integer BASIC you may have decided that arrays of two dimensions would be a way to keep dollars and cents separate to allow quantities larger than $327.67 to be manipulated, and then decided otherwise when faced with a multiply. This publication is the first in which I have seen documentation for the use of the 16 bit/32 bit multiply and divide routines in the (non-auto-boot) Monitor.

What is covered in this publication is the APPLE II Monitor, ROM address range F800-FFFF. I have not yet covered the utilities (Floating point arithmetic, Sweet16) or the compilers or DOS.

This is the manual I have been looking for. I hope you feel the same way.

William E. Dougherty

14349 San Jose St.
Mission Hills, Calif. 91345

213.896-6553

TABLE OF CONTENTS

PAGE ZERO USAGE BY THE MONITOR

The Monitor makes use of page zero locations 32 ($20) thru 79 ($4F) for general functions. In addition, locations 80 ($50) thru 85 ($55) are used for the 16 bit Multiply and Divide routines which are not used by the Monitor itself.

Following are definitions of the page zero locations used by the Monitor with descriptions of how and when they are used.

32	$20	WNDLFT	Left column of the Scroll Window: Range is 0 to 39 ($27). This field is used only in VTABZ. The contents, when changed by user program, become effective on the next scroll operation, clear to end of page operation, or carriage return output. CH contains cursor horizontal position relative to (WNDLFT). After changing the contents of WNDLFT, either CALL VTAB or output a carriage return to make it take effect.
33	$21	WNDWDTH	Width of the Scroll Window: Range is 1 to 40-(WNDLFT). Whenever a character is written through COUT to the screen, CH is incremented after use in storing the character in the screen area. At that time the contents of CH is compared with the contents of WNDWDTH to determine whether the cursor has exceeded the right margin of the Scroll Window.
34	$22	WNDTOP	Top line of the Scroll Window: Range is 0 to 22 ($16) for full text screen. Range is 20 to 22 ($14 to $16) for mixed graphics/text. This field is only used during a scroll operation to indicate where the operation should start.
35	$23	WNDBTM	Bottom line of Scroll Window: Range is (WNDTOP)+1 to 24 ($18). Contents of WNDBTM are tested only on output of a carriage return ($8D) or line feed ($8A). It is used by Clear to End of Page and by Scroll routines.
36	$24	CH	Displacement from WNDLFT where next character to the screen will be placed: Range is 0 to (WNDWDTH)-1. After the screen output routine STOADV places a character into the screen area as part of normal character output, CH is then incremented and compared to WNDWDTH. If CH is not low then a carriage return will be simulated. Note that CH is used for echoing keyboard input to the screen by the Monitor GETLN etc. routines.

PAGE ZERO USAGE BY THE MONITOR

37	$25	CV	Vertical screen position (line number) for next character to be written to the screen: Range is 0 to 23 ($17). The content of CV is relative to the top of the screen, not to the top of the Scroll Window. It may be set by loading the desired line number into A-reg and calling TABV. It may be set by POKEing the line number into CV and then calling VTAB. Actual storage of a character into the screen area includes use of BASL,H for line number, not CV. The calls above to VTAB or TABV are to set BASL,H from CV for immediate future reference. CV is never compared to WNDTOP, so no output character will automatically place it in the Scroll Window. If CV is below WNDBTM it will remain on current line as carriage returns go by while the contents of the Scroll Window will be scrolled for each.
38 39	$26 $27	GBASL GBASH	Memory address within the screen area of the left end point of the desired line for LORES plot. This field is set by GBASCALC routine to the memory location appropriate for the line number specified in the A-reg.
40 41	$28 $29	BASL BASH	This two byte field is the memory address for the left end character position of the current text line, within the Scroll Window. The contents are a function of CV and WNDLFT. This field is set by the BASCALC routine to point to the memory address for the left end of the line specified in the A-reg. This call to BASCALC is usually accomplished by the VTAB routine, which then adds (WNDLFT) to BASL,H to point to the left end of the line within the window.
42 43	$2A $2B	BAS2L BAS2H	This two byte field is used as a work area only during a scroll operation. It is the destination line pointer used as each line is moved to the position above current.
44	$2C	H2	Right end point of a horizontal line being drawn by HLINE routine: Range is 0 to 39 ($27). This byte is set by the calling program before HLINE is called.
"	"	LMNEM	Low byte of two byte pointer (LMNEM,RMNEM) used by Disassembler for index to mnemonics table.
"	"	RTNL	Save area used by Instruction Trace routine.

PAGE ZERO USAGE BY THE MONITOR

45	$2D	V2	Bottom point of a vertical line being drawn by VLINE routine: Range is 0 to 19 for mixed screen, 0 to 23 ($17) for full screen graphics. This byte must be set before VLINE is called. Note that this byte is used when the Clear Screen (CLRSCR) uses VLINE to clear the screen.
"	"	RMNEM	Used with LMNEM as table index for mnemonic table by Disassembler.
"	"	RTNH	Used with RTNL as save area by Instruction Trace routine.
46	$2E	MASK	With this label, this location is used as a $0F or $F0 by PLOT depending on whether the point is on the high side or the low side of the two plot lines represented by the GBASL,H pointer. Each character of the form (GBASL),Y contains two points on the screen, one above the other. MASK is used to set the appropriate one while leaving the other unchanged.
"	"	FORMAT	Using this label, the Disassembler uses this byte as temporary storage for the code which indicates the format of the instruction for display purposes.
"	"	CHKSUM	This byte is used during cassette tape read to continually accumulate the checksum which will be "compared" to that generated during the write operation which created the record. This byte is initialized to zero at the beginning of a tape read. As each byte is stored into memory it is Exclusively ORed against CHKSUM. After the last byte has been stored, one more byte is read from the tape and exclusively ored against CHKSUM. If the result in the A-reg is zero, a good read may be assumed. As this result is not finally stored back into CHECKSUM, that field cannot be used by the calling program to determine success or failure of the read. See page 38 for a way.
47	$2F	LASTIN	With this label the RDBIT routine uses this byte as a work area to determine whether the sense of input from the cassette tape input register has changed.
"	"	LENGTH	This field is set by the Disassembler to indicate the length of the instruction. After output of the disassembled instruction, PCADJ uses this value to update PCL,H. Instruction trace also uses this field to indicate number of bytes to move the the instruction trace execution work area.
"	"	SIGN	After a call to MULPM or DIVPM (signed 16 bit multiply or divide) the $01 bit of this byte is set if result is to be complemented by calling program.

PAGE ZERO USAGE BY THE MONITOR

48	$30	COLOR	This byte contains the code for the color of points to be placed on the screen in graphics mode. The SETCOL routine is entered with a value in the low order 4 bits of the A-reg. This value is then placed in both the high and low nibbles of COLOR. COLOR is then used with MASK in setting the value of the byte in the screen area to accomplish setting a particular point to a certain color.
49	$31	MODE	This byte is used by the Monitor Command processing routines to indicate disposition of hex information in the input line. For example, a hex address followed by a colon causes setting of MODE so that during further processing of the input line each blank encountered signifies end of a hex value to be placed in memory.
50	$32	INVFLG	This byte is a mask used by COUT1 to cause characters written to the screen area to display white on black (INVFLG=$FF) or black on white (INVFLG=$3F) or blinking (INVFLG=$7F).
51	$33	PROMPT	This byte contains the prompt character which is written to the screen by the Monitor GETLN routine in preparation for reading a line of characters from the keyboard. When the RESET key is pressed the Monitor places an asterisk in this location as the prompt character for the Monitor.
52	$34	YSAV	This byte is a save area used by the Monitor Command Processor. The Y-reg is used by the Command Processor in indexing through the input line. When a command has been decoded the Y-reg is saved at YSAV before going to the selected service routine. On return to the Command Processor the Y-reg is reloaded from here before transfer of control to NXTITM to continue scanning of the input line.
53	$35	YSAV1	This byte is a save area for the Y-reg across a call to the screen output routines. Y-reg is saved and restored in the COUT1 routine.
54 55	$36 $37	CSWL CSWH	This two byte field contains the address of the routine which is to receive and dispose of output characters. When the RESET key is pressed this field is initialized to point to COUT1 to send output characters to the screen. Entering a Monitor Command nPc (n=port number, Pc=Control-P) will cause the Monitor to set CSWL,H to Cn00. The routine at that location will then receive (in the A-reg) each byte "written" through COUT, which is a JPI (CSWL).

PAGE ZERO USAGE BY THE MONITOR

56 57	$38 $39	KSWL	This two byte field contains the address of the user input routine. It is set by RESET key processing to point to KEYIN which gets its input from the keyboard. The Monitor Command nKc (n=port number, Kc=control K) causes the setting of KSWL,H to Cn00. This routine is then called any time the Monitor or executing program asks for another byte of input by calling RDKEY or one of the routines which calls RDKEY.
58 59	$3A $3B	PCL PCH	This field is a save and control area for the Program Counter. In addition to the Monitor uses described below, this field is the index to next address used by the Mini Assembler (at F666). This field is set by execution of a BRK instruction (but sometimes incorrectly I believe) to provide for indication of where the BRK was encountered. This field is set by Monitor Commands L, G, S, T. . It is updated as required by the routines supporting commands L (disassemble), S (single step), and T (trace). This is the field by means of which control is transferred to the desired memory location for Monitor commands G, S, T. Updating of this field is handled by the PCADJ routine using LENGTH previously set by instruction disassembly.
60 61 . . thru 67	$3C $3D $43	XQT/XQTNZ	This field is 8 bytes long and overlays A1L,H, A2L,H, A3L,H and A4L,H. This field is a work area for Instruction Step/Trace. The instruction next to be executed is moved here and modified under some conditions.

PAGE ZERO USAGE BY THE MONITOR

60 61	$3C $3D	A1L A1H	Multipurpose Monitor work area: Clobbered by Instruction Trace; see XQT above. When the Monitor begins processing a command, MODE is initialized to zero. As the input line is scanned, hex digits are first placed into A2L,H. From there they are moved also to A1L,H and A3L,H as long as MODE remains zero. When a plus, minus, colon, or period is encountered, MODE is modified to indicate which, and A1L will continue to contain the value terminated by the + or - or : or . . A1L,H is the primary index for the BLANK Monitor command, memory examine or display. A1L,H contains the minuend for a subtract Monitor command. A1L,H is the source field pointer during a Monitor MOVE command. A1L,H is one of the two indices used in the Monitor VERIFY command. A1L,H is the source field from which PCL,H is set on L, S, T, and G Monitor commands, if an address is specified for those commands. If no address is used in the input line then PCL,H is used from the residue of the last command which maintained/used it. A1L,H is the memory pointer used for cassette tape READ and WRITE Monitor operations.
62 63	$3E $3F	A2L A2H	Multipurpose Monitor work area: Clobbered by Instruction Trace; see XQT above. This field is the receiving field into which hex data is stored from the input area during Monitor Command processing. When the command itself is encountered, A2L,H contains the last parameter entered. While MODE contains zero (until a plus, minus, colon, or period is encountered) A2L,H is continually copied into A1L,H and A3L,H. If a "less than" sign is encountered, A2L,H is immediately copied to A4L,H and A5L,H. A2L,H is used to terminate examine (memory display), tape write, tape read, memory move, and memory verify operations. A2L,H contains the subtrahend in a Monitor subtract command operation. A2L,H contains the augend in a Monitor add command operation. A2L is the source field and A3L,H is maintained as the pointer for the Monitor store command. A2L contains the port number in an input port select or output port select (control K or P) command.

PAGE ZERO USAGE BY THE MONITOR

64	$40	A3L	Multipurpose Monitor work area:
65	$41	A3H	Clobbered by Instruction Trace; see XQT above.
			A1L,H and A3L,H are both filled from A2L,H during Monitor Command processing scan of the input line as described above regarding A1L,H.
			A3L,H is used as the destination pointer during Monitor store command processing.
			A3L,H is used as a work area by the Register Display routine which is called by the control-E Monitor command or as part of single step or trace operation.
66	$42	A4L	Multipurpose Monitor work area:
67	$43	A4H	Clobbered by Instruction Trace; see XQT above.
			This field (and A5L,H) are loaded from A2L,H during Monitor Command Processor scan of the input area when a "less than" sign is encountered.
			A4L,H is the receiving field pointer during a Monitor MOVE command execution.
			A4L,H is the second field pointer during a Monitor VERIFY operation.
68	$44	A5L	Multipurpose Monitor work area:
69	$45	A5H	This field is not within the bounds of the area of XQT, which overlays A1L thru A4H.
			This field is filled from A2L,H as described above for A4L,H.
Note; A5H and ACC are same address.			This field is not referenced by any Monitor Command.
69	$45	ACC	This register save area is used primarily for support of single cycle and trace operations. In addition, the registers are stored here by BRK instruction handling, and the registers are reloaded from here (except S from SPNT) before going to the address specified by a GO Monitor command.
70	$46	XREG	
71	$47	YREG	
72	$48	STATUS	
73	$49	SPNT	
			During instruction step or trace, the registers are loaded from here before execution of the instruction being traced, and stored here again after said execution.
			The Register Display function of the Monitor, as called by control-E or as part of single step or trace prints the contents of this area as the register contents.
74	$4A	unused	
75	$4B	unused	
76	$4C	unused	
77	$4D	unused	
78	$4E	RNDL	Random number field, 16 bits:
79	$4F	RNDH	This field is continually counted up by the KEYIN routine while testing for key pressed. Thus, the results are effectively random as it does not take long to overflow and start over. There is no other reference to this field within the Monitor.

PAGE ZERO USAGE BY THE MONITOR

80	$50	ACL	These three two byte fields are used only by the 16 bit multiply and divide routines, which themselves are not called from anywhere in the Monitor. Therefore, these fields are used only if a user program makes use of the multiply and divide routines.
81	$51	ACH	
82	$52	XTNDL	
83	$53	XTNDH	
84	$54	AUXL	
85	$55	AUXH	

The section on Data Manipulation Functions contains a full description of the multiply and divide routines.

For Multiply, place the two factors in ACL,H and AUX,H, call the appropriate routine (MULPM at FB60 or MUL at FB63), and find the results in ACL,H-XTNDL,H. In order of significance, place the factors in $51,50 and $55,54 and find the results in $53,52,51,50.

For Divide, the entry points are DIVPM at FB81 or DIV at FB84. Place the divident in ACL,H-XTNDL,H and the divisor in AUXL,H. The results will be quotient in ACL,H and remainder in XTNDL,H.
In order of significance, the dividend is in $53,52,51,50, the divisor in $55,54, the quotient in $51,50, and the remainder in $53,52.

OVERVIEW OF KEYBOARD INPUT AND TEXT MODE SCREEN OUTPUT

The default operation of the screen is as a scrolling device. That is, new data is entered or output at the bottom of the screen and all above is shifted up line by line until the oldest information disappears off the top of the screen. However, it is also possible, with a little extra work in the user program, to use the screen as a formated display. Following is a description of the ramifications of that type of use, and suggested solutions to the troublesome situations encountered.

Characters generated by the user program for display on the screen are handed to the Monitor one character at a time. The screen output handlers check for control character vs. display character, and operate in accordance with what they find. For example, output of a carriage return character or line feed character while the cursor is on the bottom line of the screen will cause a scroll operation to take place. If the screen is being used with a format instead of as a scroll device, then the program can easily avoid output of a carriage return or line feed when the cursor is on the bottom line of the screen.

The easiest way for the user program to read information from the keyboard is to call the Monitor at the point where it will read in a line (up to a carriage return) before returning control to the calling program. When this is done, the input information is always available at the same place in memory. There is, however, a conflict between this type of a call and using the screen as a format type display. While the Monitor is receiving the keyboard input, it "echoes" the characters to the screen at the current cursor location. When end of input is signaled by a carriage return, the Monitor clears the cursor current line from cursor to right end of the line. Thus, the user program must make sure that before asking for input from the keyboard the cursor is placed where there is no data to the right.

It is possible to divide the screen into scroll area and non-scroll area. Many complications arise from this method of operation, so the recommended solution to the format display problem is to leave the screen full scroll and avoid scroll services when they are not desirable.

In the section on Screen Output Without the Scroll Window will be found the entry points and qualifiers for using a divided display.

KEYBOARD INPUT ROUTINES OF THE MONITOR

The Monitor routines supporting keyboard input are designed to do the following; echo the keyboard input to the screen (through COUT) at the current cursor position, and store the entered characters in the keyboard input area ($200-$2FF), for the convenience of the calling program. The executing program may position the cursor anywhere (in the scrolling window) before calling the Monitor keyboard input routines. On entry of a Carriage Return from the keyboard, the Monitor keyboard input routines will cause return of control back to the calling program with the character count in the X-reg, and a Carriage Return in the input area as a terminator. The program need not look into the screen refresh memory to determine what was entered.

The routines described below are included in the address table.

GETLNZ	Entry at this point causes output of a Carriage Return (through COUT) before going to GETLN to write prompt and read data.
GETLN	Entry at this point is with the cursor properly positioned (CV, BASL,H, and CH) as described in the section regarding Text Mode Output Within the Scroll Window. GETLN prints the Prompt character and initializes X-reg for indexed storage of the input characters in the Input Area. Control then goes to NXTCHAR.
NXTCHAR	This is the top point in the character input loop. RDCHAR is called to get a character into the A-reg. On return the A-reg is tested for presence of the Ctrl U (right arrow) and if so the A-reg is loaded from screen memory assuming that the Y-reg contains the same value as CH. If the A-reg value is $E0 or greater, the lower case letter is then converted to upper case. The character is then stored from the A-reg to the Input Area. If the character is a Carriage Return, it is printed through COUT and the RTS exit of COUT will then give control back to the calling program with X-reg indicating the input character count +1. That is, the input is in memory locations $200 through $200,X where 200,X contains a Carriage Return.
RDCHAR	This routine calls RDKEY to get the next character placed into the A-reg. If, on return, it is found that the Escape key has been pressed, control is passed to the ESC1 routine for Escape key processing. After Escape key and the key following have been read and processed control returns to the RDCHAR routine as if it were just being entered.
RDKEY	This routine picks up and saves the character in the screen area at BASL,H CH (leaving Y-reg containing the contents of CH). It then changes that character to blinking to indicate current cursor position. This routine asks for the next input character to be placed in the A-reg by doing an indirect jump via KSWL,H, which is normally pointing at KEYIN. Return is therefore to the caller of RDKEY, not to the RDKEY routine itself.

KEYIN This is the routine which gets the next input key from the keyboard hardware. There are two required actions and two extra actions taken by this routine. The required actions are reading the keyboard input buffer over and over again until it is determined (by presence of the $80 bit) that a character has indeed been read, and the keyboard strobe is hit to prepare for the next keyboard input.

The auxilliary actions taken by this routine are first, to count up the random number field, ignoring overflow, and second, restoring to the screen area the character modified by the RDKEY routine to remove the blink.

Return to caller of RDKEY is accomplished by an RTS.

ESC1 This routine is called by the RDCHAR routine if an Escape key is found in the A-reg by that routine. ESC1 actually gets control after the next key is provided by RDKEY. ESC1 calls the appropriate Scroll Window service routine based on the input character. The RTS at the end of such routine results in control returning to RDCHAR at its normal entry point.

When this routine is called, the A-reg contains the character designating the action to be taken, and Carry is set. If Carry is not set at time of entry into this routine, the function accomplished is that of the character one less than the contents of the A-reg.

USER PROGRAM CALLS TO MONITOR KEYBOARD INPUT ROUTINES _ ACTUAL KEYBOARD INPUT

The following paragraphs describe how to set up for calls to the various entry points in the Monitor for keyboard input, and what the results will be.

GETLNZ

X-reg, Y-reg, and A-reg are insignificant.
CH is insignificant.
CV should point to a line in the Scroll Window.
BASL,H is insignificant.

Results:
Keyed information is in $200 through $200,X where $200,X contains Carriage Return.
A-reg contains a Carriage Return.
X-reg contains the number of characters read excluding the terminating Carriage Return.
Y-reg contains contents of WNDWDTH.
CH contains zero.
CV contains line pointer, current value.
BASL,H contains memory address corresponding to CV and WNDLFT.
Screen line is blanks to the right of the end of echoed input.

GETLN

Setup:
X-reg, Y-reg, and A-reg are insignificant.
CV and BASL,H should be compatible, pointing in the Scroll Window.
CH indicates where on that line the prompt character is to be placed, to be followed by echoed keyboard input.

Results are the same as for GETLNZ above.

NXTCHAR

Enter here to bypass print of Prompt character.

Setup:
X-reg should be set to zero to begin storing of input at $200.
Y-reg and A-reg are insignificant.
CV and BASL,H should be compatible, pointing in the Scroll Window.
CH indicates where echoing of keyboard input is to start (should be less than WNDWDTH).

Results are the same as for GETLNZ above.

NOTE: For all the above, Escape Key functions are as defined in the Reference Manual. Also, Ctrl U (right arrow for cursor movement) picks up screen contents and uses it as keyboard input.

RDCHAR Entry here gives the calling program each character as entered, except that the Escape function is supported, hidden from the calling program. Entry here bypasses Monitor service in support of the cursor right arrow key, and the input characters will not be stored in the Input Area. Conversion to upper case is bypassed. The calling program will have to take appropriate action on Carriage Return.

Blink of the current cursor position on the screen will still happen, but the Monitor service of echoing the key to the screen (and advancing CH to the next position) is bypassed.

Monitor support for the cursor left arrow (backspace) is bypassed. Cancel input line Monitor support is bypassed.

Setup:
X-reg is insignificant and will not be clobbered.
Y-reg is insignificant.
A-reg is insignificant.
CV and BASL,H should be compatible, pointing in the Scroll Window.
CH indicates horizontal position in the Scroll Window where the cursor will be indicated by blinking.

Results:
On return to the caller A-reg will contain the key value.
Y-reg will contain contents of CH.
X-reg will contain same value as at input.
CV, CH, BASL,H will have changed only if an Escape key sequence has been performed.

RDKEY Entry here bypasses Escape key Monitor support, and all the other Monitor support bypassed by RDCHAR entry. Functions still supported are to set the indicated cursor position to blinking until a key is pressed, at which time the position is restored to its previous state.

Setup:
X-reg, Y-reg, and A-reg are insignificant.
CV and BASL,H should be compatible, pointing in the Scroll Window.
CH indicates horizontal position where cursor will blink.

Results:
A-reg contains the input character (which may be Escape key or any control key or any character).
X-reg is unchanged from the call.
Y-reg contains contents of CH.
CV, CH, BASL,H remain unchanged.

NOTE: For all the above, the random number at RNDL,H is continually incremented while KEYIN routine is testing for key pressed.

KEYIN This is the keyboard physical read routine executed by the normal setting of KSWL,H.

Setup:
X-reg is unused and unaffected across this call.
A-reg input to this routine is stored at (BASL),Y when a key is pressed, before the A-reg is filled from the keyboard register.
Y-reg is used for storing A-reg in screen area to (BASL),Y.
CH and CV are not referenced, but should be appropriately set.
BASL,H are used as indicated for A-reg above.

Result:
On return to the caller, only the A-reg has been changed. It contains the input from the keyboard register.

KEYIN replacement;

There are cases in which it is desirable to replace the physical keyboard input routine with a routine which either reads from the keyboard and preprocesses the input, or gets the information to feed to the reading program from some other source than the keyboard. The requirements of such a program in replacing the KEYIN routine are as follow. Placing the program/routine into effect is accomplished by storing the entry point in KSWL,H.

The replacement routine should manage the following resources as indicated.

A-reg Store the A-reg at (BASL),Y , then load the A-reg from whatever source is to be used.

X-reg Must be unaltered. Save on entry and restore on exit if it must be used by the replacement routine.

Y-reg Use as indicated above for A-reg. It must not be changed on return from contents on entry, so save and restore if it must be used otherwise. (This caution is not required, however, if the source of the input prevents Escape key and Ctrl U (cursor advance arrow) from being entered. In such case, Y-reg is expendable.)

CV and BASL,H and CH
These are all used for echoing the "keyboard" input, so the replacement routine should either leave them alone or manipulate them in an intelligent manner.

NOTE: On replacing the pointer to KEYIN at KSWL,H, it is generally safer to pick up and store the current contents of KSWL,H in a save area before placing the address of your routine, and then restore KSWL,H from that save area when taking the replacement routine out of service.

NOTE ALSO: If you replace the contents of KSWL,H with the address of your routine while using DOS, expect the unexpected. DOS uses both CSWL,H and KSWL,H, and periodically restores them to appear the way DOS likes to see them regardless of current contents.

KEYBOARD INPUT MONITOR ROUTINE ADDRESSES

There are many points in Keyboard Service which a user program could usefully call. However, because they are generally different entry points in a continuous string of instructions, and all instructions after the point of entry will be used, this table of addresses is in Monitor sequence, rather than in sequence by potential usability.

Function	Hex Addr	+DEC Addr	-DEC Addr	Monitor Label	Regs Destroyed
Set screen to blink at cursor saving original character in A-reg. from (BASL),Y	FD0C	64780	-756	RDKEY	
Jump Indirect (KSWL) to KEYIN	FD18	64792	-744		
Increment random number at RNDL,H while polling keyboard register.	FD1B	64795	-741	KEYIN	
Store A-reg to (BASL),Y (clear blink set by RDKEY routine).	FD26	64806	-730		
Load A-reg from Keyboard register	FD28	64808	-728		
Clear keyboard strobe					
Return (RTS)					
Call RDKEY for Escape key service	FD2F	64815	-721	ESC	
Call ESC1 with char in A-reg to do indicated function.	FD32	64818	-718		
Call RDKEY to get next char into A	FD35	64821	-715	RDCHAR	
Compare to $93. =, br to ESC to call for next char and do ESC					
Return to caller (RTS)					
Using character in A-reg, br to routine for Escape key service.	FC2C	64556	-980	ESC1	

@	HOME	clear scroll window
A	ADVANCE	cursor right
B	BS	cursor left
C	LF	cursor down one line
D	UP	cursor up one line
E	CLREOL	clear to end of line
F	CLREOP	clear to end of window

other - ignore it, just RTS.

PRIMARY CALL POINTS ARE ON NEXT PAGE.

BASL,H	$28-29
KSWL,H	$38-39

KEYBOARD INPUT MONITOR ROUTINE ADDRESSES, continued.

(Logically speaking, the place to start below is GETLNZ, but the sequence of presentation here is the sequence of Monitor instructions because of heavy use of "fall into" next code segment.)

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Echo keyboard input thru COUT to screen, from IN,X , with INVFLG temporarily set to $FF.	FD3D	64829	-707	NOTCR	
Pick up char from IN,X; if $88 goto BCKSPC if $98 goto CANCEL if X-reg (input index) greater than $F7 fall into FD5C. Otherwise to NOTCR1, bypass Bell	FD4D	64845	-691		
Sound bell if X indicates 248+ input characters.	FD5C	64860	-676		
Increment X If X not zero goto NXTCHAR If X=0 fall into CANCEL	FD5F	64863	-673	NOTCR1	
Load $DC (backslash) into A-reg to indicate cancelled input.	FD62	64866	-670	CANCEL	
Call COUT to print A-reg then fall into GETLNZ	FD64	64868	-668		
Print Carriage Return thru COUT	FD67	64871	-665	GETLNZ	
Load PROMPT into A-reg	FD6A	64874	-662	GETLN	
Call COUT to print A-reg	FD6C	64876	-660		
Load X-reg with $01 for passage thru backspace operation.	FD6F	64879	-657		
If X=0 goto GETLNZ to start over. else decrement X, fall into NXTCHAR	FD71	64881	0655	BCKSPC	
Call RDCHAR to get next character If character gotten is $95 (ctrlU cursor right arrow) pick up screen character from (BASL),Y to replace it. If A-reg greater than$DF then AND against $DF to make upper case	FD75	64885	-651	NXTCHAR	
Store A-reg to input area at IN,X Compare to Carriage return. Goto NOTCR (above) if not. Else, call CLREOL to clear rest of line, then print carriage return thru COUT, with included RTS to return to caller of keyboard input.	FD84	64900	-636	ADDINP	

```
IN      =$200
INVFLG  is at $32
```

18

PAGE ZERO LOCATIONS REGARDING KEYBOARD INPUT AND SCREEN OUTPUT

Text mode output to the screen can be done within the Scroll Window or without the Scroll Window. Text mode output outside of the Scroll Window is sometimes a little difficult. Keyboard input is echoed to the screen using Monitor support for the Scroll Window.

This table of page zero locations used for keyboard input, Scroll Window output, and outside of Scroll Window output is provided for quick reference. A more complete description of each field may be found in the section on Page Zero Usage by the Monitor.

Loc Dec	Loc Hex	Monitor Label	Range Dec	Range Hex	Definition - description
32	20	WNDLFT	0-39	0-27	Left edge of Scroll Window
33	21	WNDWDTH	1-40	1-28	Width of Scroll Window
34	22	WNDTOP	0-22	0-16	Top line of Scroll Window
			20-22	14-16	if mixed graphics and text
35	23	WNDBTM	1-24	1-18	Bottom line of window
36	24	CH	0-39	0-27	Horizontal distance from left edge of window to cursor position
37	25	CV	0-23	0-17	Line number of cursor position, relative to top of screen, not top of Scroll Window
38-39	26-27	GBASL GBASH			Memory address of left end of screen line for LORES graphics or text outside of window
40-41	28-29	BASL BASH			Memory address corresponding to contents of CV and WNDLFT
50	32	INVFLG	=255 =127 =63	=FF =7F =3F	Display next character white on black Display next char. blinking. Display next char. black on white.
54-55	36-37	CSWL CSWH			Address of routine for character output to screen, normally COUT1.
56-57	38-39	KSWL KSWH			Address of routine for keyboard physical input, normally KEYIN.

OUTPUT TO THE SCREEN, TEXT MODE

There are many points in the Monitor where a user program can call for service. This section is therefore subdivided to facilitate finding the entry point for the function desired.

Screen Format Control
identifies the entry points by means of which display operation (full text, full graphics, mixed LORES graphics and text, display page), Scroll Window setup, and character display mode (black on white or white on black or blinking) are established or modified.

Window Data Manipulations
describes Monitor calls which clear all or part of the Scroll Window, set parts of the window to some user specified value, or cause conditional or unconditional scrolling of the window.

Cursor Control
describes the ways and means of moving the cursor relative to its current position, or moving it to some location independent of its current position.

General Data Output
describes the Monitor entry points by means of which to output user program generated data to the screen or to the current output device if CSWL has been modified. Also, entry points are described by means of which standard types of output (blanks, bell code, carriage return) can be transmitted to the output device (generally screen).

Special Data Output
describes Monitor entry points for printing the contents of certain fields in certain ways, generally used for program development or programming aid programs. Entry points are indicated for printing the contents of certain registers in hex, printing special or general parts of memory in hex, and calling on the Monitor register display routine.

Text Output Without the Scroll Window
describes the entry points used for placing characters on the screen outside of the Scroll Window, and for reading the keyboard when echo to the Scroll Window is not to be performed.

Any entry points which would seem to belong in more than one place will, of course, be found in each applicable place.

SCREEN FORMAT CONTROL

This page identifies the places in the Monitor which control the display mode of operation and the Scroll Window configuration.

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Entry point from RESET key function (STATUS byte cleared, ignore) and	FB2F	64303	-1233	INIT	A
Clear HIRES graphics mode and	FB33	64307	-1229		A
Set display page 1 and	FB36	64310	-1226		A
Set TEXT mode and	FB39	64313	-1223	SETTXT	A
Load 0 into A-reg for WNDTOP and branch to SETWND below.	FB3C	64316	-1220		A
Set color graphics mode and	FB40	64320	-1216	SETGR	A
Set mixed graphics/text mode and	FB43	64323	-1213		A
Call CLRTOP to clear graphics and	FB46	64326	-1210		A,Y
Load 20 ($14) into A-reg for set of WNDTOP. Fall into SETWND	FB49	64329	-1207		A
Set top line of window (WNDTOP) from A-reg, 0 or 20 or user set. and	FB4B	64331	-1205	SETWND	A
Load A-reg with 0 for WNDLFT and	FB4D	64333	-1203		A
Store A-reg to WNDLFT and	FB4F	64335	-1201		A
Load A-reg with 40 for WNDWDTH and	FB51	64337	-1199		A
Store A-reg to WNDWDTH and	FB53	64339	-1197		A
Load A-reg with 24 ($18) for WNDBTM	FB55	64341	-1195		A
Store A-reg to WNDBTM and	FB57	64343	-1193		A
Load A-reg with 23 for VTAB and	FB59	64345	-1191		A
Store A-reg to CV and	FB5B	64347	-1189	TABV	A
Jump to VTAB to set BASL,H and RTS.					
Load Y-reg with $FF for INVFLG and br to SETIFLG	FE84	65156	-380	SETNORM	Y
Load Y-reg with $3F for INVFLG and br to SETIFLG	FE80	65152	-384	SETINV	Y
Store Y-reg in INVFLG and RTS $FF from SETNORM = white on black $3F from SETINV = black on white $7F from user set = blinking	FE86	65158	-378	SETIFLG	Y
Set CSWL,H to point to COUT1	FE93	65171	-365	SETVID	A,X,Y

STATUS	$48	WNDTOP	$22
CV	$25	WNDLFT	$20
BASL,H	$28-29	WNDWDTH	$21
INVFLG	$32	WNDBTM	$23

SCREEN FORMAT CONTROL BY POKE/STore

In many cases, the routine in the Monitor described on the previous page exists because the Monitor itself requires the capability indicated. Often, calling the Monitor for a specific control function is doing it the hard way. This table indicates other ways of accomplishing the same results.

Function	Method
Set LORES Graphics display mode	POKE -16304,0 or STA C050
Set TEXT display mode	POKE -16303,0 or STA C051
Set LORES mode to Full Screen	POKE -16302,0 or STA C052
Set MIXED LORES and TEXT mode	POKE -16301,0 or STA C053
Set display to page 1 (normal)	POKE -16300,0 or STA C054
Set display to page 2 (alternate)	POKE -16299,0 or STA C055
Clear HIRES graphics mode	POKE -16298,0 or STA C056
Set HIRES graphics mode	POKE -16297,0 or STA C057
Set top line of Scroll Window	POKE 34 ($22) with line number, 0-23
Set left edge of Scroll Window	POKE 32 ($20) with column no., 0-39
Set width of Scroll Window	POKE 33 ($21) with no. columns, 1-40, Left edge + width not to exceed 40.
Set bottom line of Scroll Window	POKE 35 ($23) with line number, 1-24, Bottom must be greater number than top.
Set Normal (white on black)	POKE 50,255 or store $FF in $32.
Set Blink for future output	POKE 50,127 or store $7F in $32.
Set Reverse (black on white)	POKE 50,63 or store $3F in $32.

SCROLL WINDOW DATA MANIPULATION ENTRY POINTS

This table describes three types of Scroll Window data manipulation entry points. The first is Monitor label ESC1, Escape Key Processor, because it transfers control to a number of the other entry points depending upon the A-reg contents. Second is points supporting clearing or setting parts to the screen to a particular value. Third is points causing conditional or unconditional scrolling of the window.

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Call screen data manipulation If A = @ goto HOME A ADVANCE B BS C LF D UP E CLREOL F CLREOP other, RTS back	FC2C	64556	-980	ESC1	A,Y
Clear from line (CV) col (CH) to end of window.	FC42	64578	-958	CLREOP	A,Y
Clear from ln (CV) col (Y) to end of window	FC44	64580	-956		A,Y
Clear to EOP from ln (A) col (Y)	FC46	64582	-954	CLEOP1	A,Y
Clear Scroll Window to blanks, set cursor to top left of window	FC58	64600	-936	HOME	A,Y
Set CH=0, CV=(A), clear to EOPage.	FC5A	64602	-934		A,Y
Clear line from cursor ((BASL),CH)	FC9C	64668	-868	CLREOL	A,Y
Clear line from cursor ((BASL),Y)	FC9E	64670	-866	CLEOLZ	A,Y
Set character in A-reg from cursor ((BASL),Y) to EOLine	FCA0	64672	-864	CLEOL2	A,Y
Clear line (BASL), then set BASL,H from CV, WNDLFT	FC95	64661	-875	SCRL3	A,Y
Clear line from cursor ((BASL),Y then set BASL,H from CV, WNDLFT	FC97	64663	-873		A,Y
Following points are sequential, enter at any point to do following.					
Zero to A-reg for CH	FC62	64610	-926	CR	A,Y
Set A-reg to CH	FC64	64612	-924		A,Y
Increment CV	FC66	64614	-922	LF	A,Y
Compare CV to WNDBTM, set BASL,H if ok, call scroll function if required.	FC68	64616	-920		A,Y
Scroll the window, lines (CV) thru (WNDBTM)	FC70	64624	-912	SCROLL	A,Y
Scroll window, lines (A)-(WNDBTM)	FC72	64626	-910		A,Y

CH	$24	Cursor horizontal posit, relative to WNDLFT
CV	$25	Cursor vertical, rel to top of screen, not window.
BASL,H	$28-29	Memory address of screen line from (CV) and WNDLFT set by BASCALC routine and VTAB routine.

CURSOR POSITION CONTROL

In general, the Cursor is at the position indicated by the contents of CV (line number relative to top of screen) and CH (column number relative to left margin of the Scrolling Window). The memory location of the cursor is the sum of the contents of BASL,H (which contains the address of the leftmost character of the line within the Scrolling Window) and the contents of CH. Normally, then, BASL,H contains an address computed from the contents of CV and WNDLFT. However, if either CV or WNDLFT is changed without recomputing BASL,H then the different routines of the Monitor may come up with unpredictable (or at least undesired) results.

In the following table, the description includes indication of which of the cursor address fields is being used for what. Note, for example, that at FC95 the line indicated by BASL,H is cleared, and then BASL,H is recomputed from CV, WNDLFT for future references.

<u>The ESC1 and VIDOUT routines</u> are included in the table because they can be made to call (goto) the other entry points by giving them the appropriate A-reg contents on entry. VIDOUT is the routine which handles CR, backspace, and line feed when such characters are sent through COUT1 (generally through COUT). ESC1 is the routine called by the Monitor keyboard input routines to handle the first character after recognition of an Escape key. Thus it has four way cursor movement capability, as well as being able to call for clearing the screen line to the right of the cursor, clearing also all lines from there to the bottom of the window, or clearing the entire Scroll Window.

The next group of points contains <u>those which affect</u> (clear) <u>data</u> on the screen as well as move the cursor.

The third group is entry points supporting <u>movement of the cursor</u> relative to its current position

The fourth group supports positioning the cursor at a desired location without reference to its current position. To do this, the program should set CV and CH and then call the appropriate routine to set BASL,H.

CURSOR POSITION CONTROL continued

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Call screen/cursor manipulation	FC2C	64556	-980	ESC1	A,Y
If A = @ goto HOME — clear the scroll window, cursor to top left.					
A ADVANCE — cursor right one space					
B BS — cursor left one space (to right end of line above if necessary and possible)					
C LF — cursor down one line, scroll if necessary.					
D UP — cursor up one line if possible.					
E CLREOL — clear line to right of cursor.					
F CLREOP — clear from cursor to end of window.					
other — RTS; ignore the entry.					
Place character in screen memory or Process Control Character	FBFD	64509	-1027	VIDOUT	A,Y
If (A) GT $9F or LT $80 goto STOADV					
If A = $8D goto CR	FC04	64516	-1020		A,Y
$8A LF					
$88 BS					
$87 sound bell					
other ignore it - RTS					
Clear Scroll Window, set cursor to top left corner of window.	FC58	64600	-936	HOME	A,Y
Set CV from A-reg; Clear to bottom of window. Set CH=0	FC5A	64602	-934		A,Y
Clear line (BASL,H), whole line then set new BASL,H from CV, WNDLFT.	FC95	64661	-875	SCRL3	A,Y
Load Y from CH and	FBF0	64496	-1040	STOADV	A,Y
Store A-reg to screen at (BASL),Y and	FBF2	64498	-1038		A
Increment CH and	FBF4	64500	-1036	ADVANCE	A
Compare (CH) with (WNDWDTH) and goto CR if CH not less. Else return (RTS)	FBF6	64502	-1034		A
Move cursor left one column, to right end of line above if required and possible.	FC10	64528	-1008	BS	A
Move cursor up one line (if possible)	FC1A	64538	-998	UP	A
Load 0 to A-reg for CH (Car.RET) and	FC62	64610	-926	CR	A
Store A-reg to CH and	FC64	64612	-924		A
Increment CV and	FC66	64614	-922	LF	A
Compare CV to WNDBTM, If CV not less decrement CV and do scroll. If CV less goto VTABZ to set BASL,H and return.	FC68	64616	-920		A

BASL,H	$28-29	WNDLFT	$20
CV	$25	WNDWDTH	$21
CH	$24	WNDTOP	$22
		WNDBTM	$23

CURSOR POSITION CONTROL continued

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Place cursor at line (A), col (CH) (store A to CV and comp BASL,H by JMP to VTAB)	FB5B	64347	-1189	TABV	A
Set BASL,H from CV and WNDLFT by call BASCALC and add WNDLFT	FC22	64546	-990	VTAB	A
Set BASL,H from A-reg and WNDLFT by call BASCALC and add WNDLFT	FC24	64548	-988	VTABZ	A
Set BASL,H to memory address for left character of line in A-reg (not left character of window)	FBC1	64449	-1087	BASCALC	A

BASL,H	$28-29	WNDLFT	$20
CH	$24	WNDWDTH	$21
CV	$25	WNDTOP	$22
		WNDBTM	$23

GENERAL TEXT OUTPUT

The preferred method of sending text to the screen is by loading the character desired into the A-reg and calling COUT to handle it from there. The reason this is preferred is that if it is (later) desired to send the output to some other device than the screen, this can be managed by changing CSWL,H to point at the program supporting such other device. There are times, however, when it is desired to write to the screen regardless of the setting of CSWL. COUT1 is the entry point for screen only output, where support is desired for reverse video display, or blinking characters by means of setting INVFLG. COUTZ is an entry point later in the processing, wherein INVFLG is no longer of interest.

Output to the screen may be written via these alternate entry points. However, note that the Monitor will still use COUT for the keyboard input echo function.

Following are addresses of the above mentioned locations, and a few other entry points which will output the specified character(s) without the calling program having to load them into the A-reg before the call. Such points mentioned here are useful for general user programs. The section on Special Text Output contains more points of less usefullness.

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Print a byte to specified (CSWL) output device, normally COUT1.	FDED	65005	-531	COUT	A
Print a byte to the screen after AND with INVFLG (if not control)	FDF0	65008	-528	COUT1	A
Print a byte to the screen	FDF6	65014	-522	COUTZ	none
Print Carriage return thru COUT	FD8E	64910	-626	CROUT	A
Print three blanks thru COUT	F948	63816	-1720	PRBLNK	A,X
Print (X) blanks thru COUT	F94A	63818	-1718	PRBL2	A,X
Print A-reg followed by (X)-1 blnks	F94C	63820	-1716	PRBL3	A,X

Note 1; In each case, the character printed goes to line (CV) col (CH), after which CH is advanced.

Note 2; The following control characters are effective - have the expected effect.

$8D	carriage return
$8A	line feed; cursor down one line, may scroll.
$88	backspace
$87	sound the bell.

Any other character in the range of $80 thru $9F is dropped. It does not cause cursor motion or screen memory modification.

SPECIAL TEXT OUTPUT

The general method of output from a program to the screen is for the program to load the desired characters, one at a time, into the A-reg and call COUT. There are, however, a number of places in the Monitor where either data in registers or storage is printed in a different way, or specific data is printed without regard to the input registers. In all cases below, COUT is used for passing the data to the output device. Many of the points below are handy for patching into a program being developed in order to facilitate determining why there is a difference between actual and intended operation of the program. Many of these, then, will be repeated in the section on Debugging Aids.

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Print "ERR" and sound bell	FF2D	65325	-211	PRERR	A
Sound bell	FF3A	65338	-198	BELL	A
Print Carriage Return	FD8E	64910	-626	CROUT	A
Print hex, right nibble of A-reg	FDE3	64995	-541	PRHEX	A
Print hex, A-reg	FDDA	64986	-550	PRBYTE	A
Print hex, Y reg and X-reg	F940	63808	-1728	PRNTYX	A,X
Print hex, A and X regs	F941	63809	-1727	PRNTAX	A,X
Print hex, X reg	F944	63812	-1724	PRNTX	A,X
Print three blanks	F948	63816	-1720	PRBLNK	A,X
Print (X) blanks	F94A	63818	-1718	PRBL2	A,X
Print char in A-reg & (X)-1 blanks	F94C	63820	-1716	PRBL3	A,X
Print Car Ret, then hex of Y & X regs, then print minus sign	FD96	64918	-618	PRXY2	A
Print hex Y&X regs and minus sign	FD99	64921	-615		A
Print CR, then hex of A1H,A1L, then minus sign.	FD92	64914	-622	PRA1	A,X,Y
Print hex of memory from xxxx thru xxx7 where xxxx is contents of A1L,H	FDA3	64931	-605	XAM8	A (Y must be 0 on entry)
Print hex contents of memory from (A1L,H) thru (A2L,H)	FDB3	64947	-589	XAM	A (Y=0 before call)
Save all registers at $45-$49	FF4A	65354	-182	SAVE	none
Display register contents as saved by SAVE routine from $45-$49.	FAD7	64215	-1321	REGDSP	A,X

CHARACTER OUTPUT WITHOUT THE SCROLL WINDOW

If all or part of the screen is to be used in a direct addressing manner, it is necessary to avoid certain Monitor services. In general, the Scroll Window services provided by the Monitor are;

1. Scroll all text in the window up one line if a carriage return or line feed takes the cursor down thru the bottom line.
2. Automatically assume carriage return if window width is exceeded.
3. Place cursor at left edge of the Scroll Window instead of at the left edge of the screen on a carriage return.
4. Support screen clear functions;
   A. Clear the window, place cursor at top left corner.
   B. Clear the window from current cursor position.
   C. Clear line to the right of cursor position.

When using all or part of the screen as a random access display, these automatic services need be avoided.

If the full screen is to be used as a random access display, without a portion being used as a working Scrolling Window, the problem is not too difficult. Consider leaving the whole screen <u>defined</u> as the Scroll Window.

1. The scroll operation only occurs if a Carriage Return or line feed or exceeding window width occurs on the bottom line of the Scroll Window. Avoid this by not having the program output CR or LF or excessive data on the bottom line of the screen, and by keeping the cursor away from the bottom line of the screen during keyboard input operations.
2. The full screen is defined as the Scroll Window by the Monitor when the RESET key is pressed. A user program can restore the window parameters to this configuration if they have been altered by calling Set Normal Scroll Window at $FB3C or 64316 or -1220.
3. Preceed the output of each string of characters with a Monitor call to set line number (TABV or VTAB) in CV and BASL,H, and a POKE or STore of character number in CH.
4. Output the string of characters by the same means as if operating with scroll services, being careful not to unintentionally exceed window width or output carriage returns. Depending on your screen design, you may intentionally do each of these.

Note that program output of a carriage return does not clear the line to the right of that carriage return, but keyboard input of a carriage return does (if reading the keyboard is being done by the Monitor get-line routines).

CHARACTER OUTPUT WITHOUT THE SCROLL WINDOW continued

If part of the screen is to be allocated as an operating Scroll Window while the remainder of the screen is to be directly accessed, then a different (lower) level of Monitor services must be called upon.

One approach toward supporting such a divided screen is to use the Scroll Window for data input, using the Monitor get-input-line services, and use Scroll Window support for whatever output the program intends to put there, and then use parts of LORES graphics support for placing characters on the screen outside the Scroll Window, as described below. The aim here is to leave support of cursor position (zero page fields CV, CH, and BASL,H) up to the Monitor, and use other methods/fields for placing characters outside the Scroll Window.

To place characters on the screen outside the scroll window,

1. with the line number in the A-reg, call GBASCALC ($F847, +63559, -1977) to set GBASL,H ($26-27) to point to the memory address of the left character position of the indicated line.
2. With Y-reg indicating horizontal position on the line, store the desired character at (GBASL),Y.

Note that this technique does not interfere with LORES plotting if the screen is being used in mixed mode because PLOT calls always set GBASL,H as required without regard to possible previous contents.

CHARACTER OUTPUT WITHOUT THE SCROLL WINDOW continued

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
OUTSIDE OF SCROLL WINDOW					
Compute memory address for line in A-reg; set GBASL,H	F847	63559	-1977	GBASCALC	A
INSIDE SCROLL WINDOW					
Write byte in A-reg to screen at cursor (CV),(CH) using INVFLG and supporting cursor move.	FDF0	65008	-528	COUT1	none
Write byte in A-reg to screen at (CV),(CH) with cursor move but not INVFLG.	FDF6	65014	-522	COUTZ	none
Clear Scroll Window to blanks, cursor to top left corner	FC58	64600	-936	HOME	A,Y
Set CV from A-reg, clear window to end of window.	FC5A	64602	-934		A,Y
Place cursor at line (A) col (CH) setting CV and BASL,H from A	FB5B	64347	-1189	TABV	A
Set BASL,H from CV (and WNDLFT)	FC22	64546	-990	VTAB	A
Set BASL,H from (A) and WNDLFT without regard to CV	FC24	64548	-988	VTABZ	A
Set BASL,H to left end of screen line (not window line) in A-reg	FBC1	64449	-1087	BASCALC	A

WNDLFT	$20
WNDWDTH	$21
WNDTOP	$22
WNDBTM	$23
CH	$24
CV	$25
GBASL,H	$26-27
BASL,H	$28-29
INVFLG	$32

LORES PLOTTING

In standard (or low resolution) plotting mode, the graphic area of the screen is 40 points wide and either 40 points high with 4 lines of text below or 48 lines high. The same memory area is used for low resolution plotting as is used for text output to the screen. However, in the graphics mode, each character position contains information for two plot points, one immediately above the other. Thus, 20 text lines are used to display 40 graphics lines in the mixed mode, and 24 text lines are used to display 48 graphics lines in the mixed mode.

There are four bits allocated for each point, by means of which the point may be displayed in any of 16 colors.

The Monitor contains routines supporting the following functions;

Set display mode to mixed graphics and text,

Clear the graphics part of the screen (in whole or in limited part),

Set a color control byte to be used for each plot point established until another color is selected,

Plot a single point at an indicated vertical/horizontal position,

Plot a horizontal line from one vertical/horizontal point to a vertical point,

Plot a vertical line from given start point to specified end horizontal,

Return to requesting program the color value of the point at a specified coordinate.

There are limitations on some of these functions which may not always be desirable. For example, using the entry point which sets mixed graphics and text includes clearing the graphics part of the screen, setting the Scroll Window to be the entire remainder of the screen, and moving the cursor (straight down from current position) to the bottom line of the screen. In addition, there is no Monitor entry point for setting full screen graphics mode. However, the display mode controls are easily set in any desired fashion merely by poking or storing into the appropriate memory locations, so this is certainly no major problem.

Various page zero locations are used for low resolution graphics mode.

GBASL,H	\$26-\$27	is set by the GBASCALC routine to the memory address of the plotting line specified.
COLOR	\$30	contains the selected color value in both high and low nibbles of the byte.
MASK	\$2E	is used internally by the plot routines as \$F0 or \$0F to set either the high or low nibble of the receiving byte depending on whether the graphics line is the top or bottom of the two displayed from that "text" line.
H2	\$2C	is the end of line position for horizontal line drawing.
V2	\$2D	is the end of line position for vertical line drawing.

LORES PLOTTING continued

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Plot a point at line (a) col (Y) leaving GBASL,H and MASK set	F800	63488	-2048	PLOT	A
Plot a point, line per GBASL,H and MASK, col in Y	F80E	63502	-2034	PLOT1	A
Draw horizontal line at (A) from (A) thru (H2)	F819	63513	-2023	HLINE	A,Y
Draw horizontal line at line indicated by GBASL,H & MASK from (Y) thru (H2)	F81C	63516	-2020	HLINE1	A,Y
Plot vertical line at (Y) from (A) thru (V2)	F828	63528	-2008	VLINE	A
Plot vertical line at (Y) from (A)+1+carry thru (V2)	F826	63526	-2010	VLINEZ	A
Plot vertical line at (Y) from (A)+1 thru (V2)	F82D	63533	-2003		A
Clear full (48 lines) screen	F832	63538	-1998	CLRSCR	A,Y
Clear graphics area (40 lines)	F836	63542	-1994	CLRTOP	A,Y
Clear graphics partial from line 0 thru (Y), 40 col wide	F838	63544	-1992	CLRSC2	A,Y
Clear graphics partial from line 0 to (V2) 40 col wide	F83A	63546	-1990		A,Y
Clear graphics partial, top left lines 0 thru (V2), cols 0 thru (Y)	F83C	63548	-1998	CLRSC3	A,Y
Set COLOR for following points to (A)	F864	63588	-1948	SETCOL	A
Change COLOR to (COLOR)+3	F85F	63583	-1953	NXTCOL	A
Load color of point (A),(Y) to A	F871	63601	-1935	SCRN	A
Set GBASL,H from (A). (A)=line/2	F847	63559	-1977	GBASCALC	A
Set Color Graphics display mode and following also done;	FB40	64320	-1216	SETGR	A
Set graphics mode to Mixed and	FB43	64323	-1213		A
Clear graphics part of scrn and	FB46	64326	-1210		A
Load $14 to A for WNDTOP and	FB49	64329	-1207		A
Store (A) to WNDTOP and	FB4B	64331	-1205	SETWND	A
Load 0 to (A) for WNDLFT and	FB4D	64333	-1203		A
Store (A) to WNDLFT and	FB4F	64335	-1201		A
Load $28 to (A) for WNDWDTH and	FB51	64337	-1199		A
Store (A) to WNDWDTH and	FB53	64339	-1197		A
Load $18 to (A) for WNDBTM and	FB55	64341	-1195		A
Store (A) to WNDBTM and	FB57	64343	-1193		A
Load $17 to (A) for TABV and Go to TABV to set BASL,H	FB59	64345	-1191		A

GBASL,H	$26-27
H2	$2C
V2	$2D
MASK	$2E
COLOR	$30

DATA MANIPULATION FUNCTIONS

There are a number of routines in the Monitor which may be called by user programs to perform often needed tasks. These may be divided into two categories; those which are used by the Monitor to perform Monitor commands, and those provided specifically for user programs but not "used" by the Monitor. The routines described in this section are listed here for quick reference.

- MOVE bytes from (A1) thru (A2) to (A4)
  - Increment A1 and A4 with compare A1:A2
  - Increment A1 with compare A1:A2
- MULtiply two bytes by two bytes giving four byte result
- DIVide four or two bytes by two bytes giving two byte result
- SAVE 6502 registers (for reload or display)
- RESTORE 6502 registers from save area (except S)

The addresses of these routines are in the table below. The user is cautioned, however, to pay close attention to the descriptions of Multiply and Divide on the next page with regard to scaling and signs.

Function	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Multiply signed fields leaving sign in LSB of SIGN	FB60	64352	-1184	MULPM	A,X,Y
Multiply fields unsigned 51,50 * 55,54 = 53,52,51,50	FB63	64355	-1181	MUL	A,X,Y
Divide signed fields leaving sign in SIGN LSB (from 51,55)	FB81	64385	-1151	DIVPM	A,X,Y
Divide unsigned fields 53,52,51,50 / 55,54 = 51,50	FB84	64388	-1148	DIV	A,X,Y
Set absolute values for ACL,H and AUXL,H leaving resulting sign in LSB of SIGN (called by MULPM and DIVPM)	FBA4	64420	-1116	MD1	A,X,Y
Move bytes in memory to (A4L,H) from (A1L,H) thru (A2L,H) (call with Y=0)	FE2C	65068	-468	MOVE	A
Increment pointer A4L,H <u>and</u>	FCB4	64692	-844	NXTA4	A
Increment pointer A1L,H with set of carry if resulting A1L,H not less than A2L,H	FCBA	64698	-838	NXTA1	A
Save 6502 regs A,X,Y,P,S at $45-49	FF4A	65354	-182	SAVE	A,X
Restore 6502 regs A,X,Y,P from $45-48	FF3F	65343	-193	RESTORE	A,X,Y,P

ACC	$45	A1L,H	$3C,3D
XREG	$46	A2L,H	$3E,3F
YREG	$47	A4L,H	$42,43
STATUS (P reg)	$48	ACL,H	$50,51
SPNT (S reg)	$49	XTNDL,H	$52,53
		AUXL,H	$54,55

DATA MANIPULATION FUNCTIONS

MULTIPLY routine

Set Multiplier in $55,$54 (MSB,LSB)
Set Multiplicand in $51,$50 (MSB,LSB)
Clear $52, $53 to zero.
Call/JSR FB60 or FB63 (-1184 or -1181) (MULPM or MUL) depending on sign conventions/requirements.

Result, in order of significance most to least, is in $53, $52, $51, $50. This result is positive. If one of the two inputs (but not both) was negative, then SIGN (at $2F) contains an 01 bit, indicating that the result should be complemented by the user program before further use.

Examples:

Called	Inputs				Output				
routine	$51	$50	$55	$54	$53	$52	$51	$50	$2F
MULPM	00	01	00	01	00	00	00	01	00
	00	01	01	00	00	00	01	00	00
	04	00	08	00	00	20	00	00	00
	FC	00	08	00	00	20	00	00	01
	FC	00	F8	00	00	20	00	00	02
	7F	FF	7F	FF	3F	FF	00	01	00
	80	00	02	00	01	00	00	00	01
	80	00	80	00	40	00	00	00	02
MUL	00	01	00	01	00	00	00	01	
	00	01	01	00	00	00	01	00	
	04	00	08	00	00	20	00	00	
	FC	00	08	00	07	E0	00	00	
	FC	00	F8	00	F4	20	00	00	
	00	FC	00	F8	00	00	F4	20	
	80	00	02	00	01	00	00	00	
	80	00	80	00	40	00	00	00	
	12	34	56	78	06	26	00	60	

DATA MANIPULATION FUNCTIONS

DIVIDE routine

This routine accomplishes the division of the number in bytes (most to least significant) $53,52,51,50 by the number in bytes $55,54, leaving the quotient in $51,50 and remainder in $53,52.

If the contents of $53,52 is larger than the contents of $55,54 then the result will not fit in the quotient bytes - overflow has resulted.

With regards to scaling, looking at the four byte dividend as an integer value and the divisor in $55,54 as an integer, the quotient and remainder fields are also integers.

Sign can be a problem if the DIVPM entry point is used. The sign bit of the dividend is the $80 bit of byte $51. If the intended divide is two bytes by two bytes (with $53,52 cleared before divide) then signed fields division is supported, with the sign bit being the LSB of $2F. If the call is to DIVPM, then if $2F contains $01 complement the result before using it.

When using unsigned divide, entry point DIV, then the divide is a 32 bit field by a 16 bit field with a 16 bit result.

Examples:

Called	Inputs		Outputs		
Routine	Dividend	Divisor	Quotient	Remainder	Sign
	53 52 51 50	55 54	51 50	53 52	2F
DIVPM	00 40 00 00	08 00	08 00	00 00	00
	00 00 00 08	00 04	00 02	00 00	00
	00 01 00 00	00 02	80 00	00 00	00
	00 00 00 03	00 02	00 01	00 01	00
	00 00 30 00	02 00	00 18	00 00	00
	00 00 30 00	20 00	00 01	10 00	00
	00 00 33 33	00 22	01 81	00 11	00
	00 10 40 00	04 00	04 10	00 00	00
	00 20 80 00	08 00	04 10	00 00	01
	00 20 $\overline{82}$ 00	$\overline{08}$ 00	04 0F	06 00	$\overline{01}$
	00 10 $\overline{41}$ 00	$\overline{04}$ 00	04 10	01 00	$\overline{00}$
DIV	00 80 00 00	80 00	01 00	00 00	
	00 00 80 00	08 00	00 10	00 00	

SPEAKER (BELL) USE THROUGH THE MONITOR

There are many ways to use the speaker in the APPLE. One of these ways is to signal program events. The Monitor contains a routine which supports this use by toggling the speaker at 1 khz for .1 second. This is the "beep" heard when the RESET key is pressed or at completion of a tape record read or write.

The APPLE does not contain the only speaker in town. That is, some printers which attach to the APPLE make a sound of some type when presented with the BELL code.

There are two basic ways to call for the BELL from a user program. If it is intended to sound the Bell in the APPLE regardless of output device in use, then
  CALL -1059 (or CALL 64477) or JSR FBDD expecting destruction of A,Y.

If it is intended to sound the alarm of the APPLE if screen is the print device or speaker in the printer, whichever applicable, then
  CALL -198 (or CALL 65338) or JSR FF3A expecting destruction of A-reg.

CASSETTE TAPE INPUT AND OUTPUT

There are two primary entry points in the Monitor with regard to reading and writing tape. They are READ and WRITE. The requirements for call, etc. for these are described below. There are a number of other routine entry points which are used by the Monitor on bit and byte basis. These are described below to the extent of location in the Monitor and indication of which APPLE programs call them, but I do not have the specifications with regard to timing between the calls. That is beyond the scope of this work.

As you will have found by now, some tape files are composed of one record, and some of two records. For example, LOADing a BASIC program results in two beeps, signaling the completions of the reads of two separate records from the tape. The first of these records is two bytes long, and contains the length of the second record. When the Monitor has satisfied BASIC's read of the first record, BASIC uses the record length indicated in that record to tell the Monitor the start and end points in memory to use for the second record. Each call to READ or WRITE in the Monitor accomplishes only one record input or output.

APPLESOFT programs are also SAVEd as two record sets. However, the first record is three bytes long; two bytes of length and one byte set to $55. Some other programs write a longer (but fixed length) first record containing length of the second record and other information about it.

WRITE	FECD	65229	-307

Before entry at this point, set the first byte address in A1L,H (3C,3D) and the last byte address at A2L,H (3E,3F). The Monitor will write ten seconds of continuous tone (header) followed by the contents of memory as specified, followed by one byte of checksum (result of exclusive OR of all the bytes written to the tape).

READ	FEFD	65277	-259

Before entry at this point, set the first byte address into A1L,H (3C,3D) and the last byte address at A2L,H (3E,3F). The Monitor reads the data from the tape, storing it into memory in the specified locations, and maintaining a running Exclusive OR result in the field called CHECKSUM ($2E). When the last specified memory location has been filled from the tape, the Monitor reads one more byte and compares it with the contents of CHECKSUM. If equal, the Monitor sounds a beep and returns to calling program. If not equal, the Monitor writes "ERR" through COUT (to the screen) before sounding the beep and returning.

If it is desired to have the calling program determine whether the tape was read successfully or not, then some special actions must be taken. One method is to compare the contents of CH ($24) before the tape read with the contents after. If they are equal, "ERR" was not written to the screen. If cursor horizontal position (CH) has changed across the call to READ, then ERR must have been written to the screen. If this condition is encountered, the program can then ask the operator to try the tape read operation again.

CASSETTE TAPE INPUT AND OUTPUT continued

The following entry points / routines are described as to function, but not documented for use. For some of them, timing is critical, and the documentation for using them would depend on how they were to be used.

HEADR	FCC9	64713	-823

This routine writes the synchronization monotone which is the first part of every tape record. When the WRITE routine calls HEADR it loads a $40 into the A-reg causing a 10 second header to be written. The READ routine also calls HEADR to delay from first detection of data coming in from the tape to the first point at which reading for 0/1 detection begins. This routine is not called by BASIC or APPLESOFT, but it is called by the Programmer's Aid #1 Tape Verify routine.

RD2BIT	FCFA	64762	-774

This routine causes looping with decrementing of Y-reg until the hardware has indicated two transitions of the tape input register. The routine RDBIT is called twice for this purpose. Contents of the Y-reg on return compared with contents on entry indicate the length of time it took for the transitions.

This routine is called from within the Monitor by the READ routine to delay entering data transfer mode until tape input is available. READ calls HEADR for the 3.5 second delay on return from its call to RD2BIT. This routine is also called from APPLESOFT, and from Tape Verify and Shape Table Load in the Programmer's Aid #1.

RDBIT	FCFD	64765	-771

This routine loops with decrementing of the Y-reg while testing the tape input register for change from zero to one or one to zero. Bit value of zero or one is then determined from the residual count in the Y-reg. This routine is called from within the Monitor routines RD2BIT and READ. It is also called by Programmer's Aid #1 Tape Verify.

RDBYTE	FCEC	64748	-788

This routine has the obvious function of calling bit reading until a byte is accumulated, and then returning to caller with the byte in hand. In addition to being called from the Monitor READ routine, it is also called by Shape Table Load.

WRBIT	FCD6	64726	-810

This routine accomplishes writing a bit to the tape when called by either the HEADR routine or the WRBYTE routine.

WRBYTE	FEED	65261	-275

When called to write a byte to the tape, this routine uses WRBIT to write ten bits to the tape. The only caller I have found is WRITE in the Monitor.

MACHINE LANGUAGE PROGRAM DEVELOPMENT AIDS

There are many routines in the Monitor which can be helpful when developing machine language programs. Some of these are routines to be used in the finished program, like the Monitor MOVE routine possibly. The others in this list are general, special, or very special screen output routines, and some data manipulation routines.

FUNCTION	Hex Addr	+Dec Addr	-Dec Addr	Monitor Label	Regs Destroyed
Write byte in A to screen at CV,CH	FDED	65005	-531	COUT	A
Print Carriage Return thru COUT	FD8E	64910	-626	CROUT	A
Print three blanks thru COUT	F948	63816	-1720	PRBLNK	A,X
Print (X) blanks thru COUT	F94A	63818	-1718	PRBL2	A,X
Print character in A followed by (X)-1 blanks	F94C	63820	-1716	PRBL3	A,X
Print BELL code thru COUT	FF3A	65338	-198	BELL	A
Print "ERR" and BELL thru COUT	FF2D	65325	-211	PRERR	A
Print low nibble of A as hex char	FDE3	64995	-541	PRHEX	A
Print A-reg as 2 hex nibbles	FDDA	64986	-550	PRBYTE	A
Print hex of Y,X regs	F940	63808	-1728	PRNTYX	A
Print hex of A,X regs	F941	63809	-1727	PRNTAX	A
Print hex of X reg	F944	63812	-1724	PRNTX	A
Print CR, then hex of Y,X regs, then minus sign or dash	FD96	64918	-618	PRYX2	A
Print hex of Y,X regs, then dash	FD99	64921	-615		A
Print CR, hex of A1H,A1L, and dash	FD92	64914	-622	PRA1	A,X,Y
Print memory as hex with preceeding address from mmmm to mmm7 where mmmm is contents of A1L,H on entry.	FDA3	64931	-605	XAM8	A (Y=0)
Print memory as hex from (A1L,H) to (A2L,H) inclusive.	FDB3	64947	-589	XAM	A (Y=0)
Save A,X,Y,P,S regs at $45-49	FF4A	65354	-182	SAVE	A,X
Display registers with names from $45-49 as SAVEd, with preceeding carriage return.	FAD7	64215	-1321	REGDSP	A,X
Display regs as above without CR	FADA	64218	-1318	RGDSP1	A,X
Restore regs A,X,Y,P not S from $45	FF3F	65343	-193	RESTORE	A,X,Y,P
Restore regs (call RESTORE), then JMP (PCL) to continue execution	FEB9	65209	-327		A,X,Y,P
Execute one instruction at (PCL,H) with register restore before, register save after, update of PCL,H, and display of instruction and display of result regs.	FA43	64067	-1469	STEP	
Move memory contents to (A4L,H) from (A1L,H) thru (A2L,H)	FE2C	65068	-468	MOVE	A (Y=0)
Compare memory contents (A4L,H) to (A1L,H) thru (A2L,H), differences are displayed.	FE36	65078	-458	VFY	A (Y=0)
Increment A4L,H ($42,43) <u>and</u>	FCB4	64692	-844	NXTA4	A
Inc A1L,H (3C,D), set carry if A2L,H not less than A1L,H	FCBA	64698	-838	NXTA1	A

PADDLES AND BUTTONS AND ANNUNCIATOR OUTPUT

The APPLE has a Game I/O connector with hardware support for four digital outputs, three digital inputs, and four analog inputs (called paddles). The Monitor contains support for reading the paddles, which consists of writing a strobe to start the paddle timer and then reading the paddle timer of interest with incrementing of Y-reg until the paddle detector comes true. The Monitor does not contain support for the digital outputs or digital inputs. Access to the digital I/O ports is accomplished by PEEKing or POKEing the appropriate address (or LDx or STx).

To use the Monitor support to read the setting of a paddle, call

PREAD at (FB1E, 64286, -1250) with Paddle number (0-3) in X-reg

and on return the "value" of the paddle will be found in the Y-reg. The A-reg is destroued in the process.

Direct reading of the paddles may be accomplished by accessing the paddle trigger and then reading the appropriate paddle input address repeatedly while counting until the value read no longer has the $80 bit set.

Game I/O Addresses Table

Start Paddle Timers	C070	49264	-16272	
Paddle 0 timer	C064	49252	-16284	negative until
Paddle 1 timer	C065	49253	-16283	timer
Paddle 2 timer	C066	49254	-16282	expires
Paddle 3 timer	C067	49255	-16281	
Paddle 0 switch	C061	49249	-16287	negative
Paddle 1 switch	C062	49250	-16286	indicates
Paddle 2 switch	C063	49251	-16285	button is pushed
Clear Annunciator 0 output	C058	49240	-16296	POKE/STore
Set Annunciator 0 output	C059	49241	-16295	zero to
Clear Annunciator 1 output	C05A	49242	-16294	appropriate
Set Annunciator 1 output	C05B	49243	-16293	address
Clear Annunciator 2 output	C05C	49244	-16292	
Set Annunciator 2 output	C05D	49245	-16291	
Clear Annunciator 3 output	C05E	49246	-16290	
Set Annunciator 3 output	C05F	49247	-16289	

NOTE FOR PADDLE READ;
Rapid reading of a second paddle after a first paddle may produce incorrect results. This caution comes from the APPLESOFT manual originally.

MISCELLANEOUS MONITOR SUPPORT

WAIT FCA8 64680 -856

A call to this routine will cause a loop for a predictable length of time, such as is used by the Monitor with regards to using the speaker as a Bell. It may be usable, for example, in writing data to a lower speed device like a printer or typewriter.

My analysis of the code indicates that the time between the call WAIT (JSR) and the end of the RTS of WAIT is

$2.5A^2 + 13.5A + 13$ machine cycles of 1.023 microseconds.

The following table indicates delay times in the WAIT routine from a number of values of A (the A-reg contents when WAIT is called).

A-reg Decimal	Time in Seconds
1	.000029667
2	.00005115
3	.000077748
4	.000109461
5	.000146289
6	.000188232
7	.00023529
8	.000287463
9	.000344751
17	.000987195
18	.001090518
19	.001198956
25	.001956999
26	.002101242
31	.002899182
32	.003074115
36	.003824997
37	.004025505
41	.004878687
42	.00510477
45	.005813709
46	.006060252
49	.006830571
50	.007097574
53	.007929273
54	.008216736
55	.008509314
56	.008807007
57	.009109815
58	.009417738
59	.009730776
60	.010048929
73	.014650383
74	.015040146
75	.015435024
85	.019665129
86	.020116272
96	.024909027
97	.025416435
105	.029659839
106	.030213282
122	.03976401
123	.040404408
137	.049907055
138	.050624178
150	.059628624
151	.060412242
162	.06936963
163	.070214628
174	.079847196
175	.080753574
184	.089141151
185	.090098679
195	.099955284
196	.100969077
204	.109263561
205	.110323389
218	.124566618
219	.125698056
239	.149400966
240	.150639819
255	.169836414

EXAMPLE USE OF CONTROL Y WITH PARAMETER.

In the APPLESOFT manual there is a caution that if one paddle is read, another should not be read too quickly. Following is a machine language program with which the interference between the paddles can be demonstrated.

Where y̸ represetns control Y, initiation of the program is by entry of the Monitor command xxxx y̸ where xxxx represents the amount of delay to use between reading paddle 0 and reading paddle 1.

```
2000     LDA #C0          set counter for number of samples to run
2002     STA 4             before clearing screen and starting over
2004     LDA 3C           pick up low part of entered count from A1L
2006     STA 10            and store it for repeated use
2008     LDA 3D           pick up high part of entered count from A1H
200A     STA 11            and store it for repeated use.

200C     LDA 10           pick up low part of count
200E     STA 12            store it in counter for this pass
2010     LDA 11             and the high part as well.
2012     STA 13
2014     LDX #0           set X for paddle 0 read
2016     JSR FB1E         call paddle read
2019     STY 0            store paddle zero value in zero

201B     DEC 12           counting down delay loop
201D     BNE 201B
201F     DEC 13
2021     BMI 201B

2023     LDX #1           set X for paddle 1 read
2025     JSR FB1E         call paddle read
2028     STY 1            store paddle 1 value in 1
202A     LDA 0            pick up paddle zero reading
202C     JSR FDDA         print as hex value
202F     LDA #A0          pick up a blank to print
2031     JSR FDED         print a blank
2034     LDA 1            pick up paddle 1 reading
2036     JSR FDDA         print hex of paddle 1 reading
2039     JSR F948         print three blanks
203C     INC 5            delay to keep paddle 1 from upsetting pdl 0.
203E     BNE 203C
2040     INC 4            is this all before we clear screen and restart?
2042     BNE 200C          NE means no, go back and sample again.

2044     LDA #0           wait awhile before clearing screen.
2046     STA 4
2048     STA 5
204A     INC 4
204C     BNE 204A
204E     INC 5
2050     BNE 204A
2052     JSR FC58         clear the screen.
2055     LDA #C0          restore the per screen counter
2057     STA 4
2059     BNE 200C          and go one more big round.

03F8     JMP 2000
```

SETTING REGISTERS FOR MONITOR CALLS FROM BASIC

Many of the entry points specified in this book require presetting of registers for proper operation. Following is a sample program, written for APPLESOFT, which uses Monitor calls for conversion from decimal to hex.

The theory behind the operation is that on a Monitor G command, the registers are loaded from the SAVE area before going to the location specified in PCL,H. Thus, by poking destination address into PCL,H and the required register contents into XREG, YREG, an entry point in the Monitor Go command processor can be used to pass the registers to a selected routine.

```
10      REM CONVERT DECIMAL INPUT TO HEX OUTPUT
100     INPUT "ENTER NUMBER ";A      read the input
110     IF A=99999 THEN END          provide a way to end the program.
150     C% = A / 256                 isolate the high byte
200     POKE 71,C%                   set YREG for PRNTYX call
300     B% = A / 256                 get remainder from A/256
310     B = B% * 256                 for low byte (XREG) poke
320     B% = A - B
350     POKE 70,B%
400     POKE 59,249                  set PCH to F9
500     POKE 58,64                   set PCL to $40
550     PRINT                        space print
600     CALL 65209                   entry point in GO processor is FEB9
650     PRINT                        a space
700     GOTO 100                     around for another number.
```

BRK INSTRUCTION PROCESSING

When the BRK instruction is executed the 6502 goes through an interrupt procedure which is common with the Interrupt Line of the 6502. The difference between the two is that the P bit $10 is set if a BRK instruction was the cause of the interrupt.

On this type of interrupt the 6502 stacks the current program address and the processor status byte. If the BRK instruction caused the interrupt, the address stored in the stack is the address of the BRK instruction +2.

The 6502 then sets the program counter from memory locations FFFF,FFFE, causing program execution to continue at the address specified in that field. In the APPLE control is thereby transferred to the IRQ routine at $FA86.

The first thing done in the IRQ routine is to save the A-reg by storing it at ACC in the register SAVE area. It then pops the stack into A, and pushes it back to restore the stack pointer. The result is that the P-reg contents stored in the stack by the interrupt process is now in the A-reg where it can be examined. The $10 bit is then tested to determine whether the interrupt was caused by BRK instruction or tickling of the Interrupt line.

If the indication is that the interrupt was not a BRK instruction, the Monitor transfers control to the routine, the address of which the user program must have previously stored at memory locations $3FE,3FF. The Monitor executes a JPI (3FE) to get to that routine.

If a BRK instruction is found to have caused the interrupt, control goes to the BREAK routine at FA92. That routine restores the P-reg from the stack, and then calls SAV1 to save all registers (for display) except A-reg which had already been saved.

The instruction address from the stack is then placed in PCL,H for display on the screen. Note that it is two bytes beyond the BRK which interrupted the processor. The registers are then displayed from the SAVE area, and control is transferred to the Monitor Command Processor.

NOTE that during STEP operations (Single cycle or Trace) the BRK instruction is software detected and handled without an interrupt, and with correct display of instruction and address.

SINGLE CYCLE AND TRACE PECULIARITIES

There are some instructions for which register display is incorrect during STEP processing. The STEP routine detects and gives special attention to JSR, RTS, JMP, JPI, RTI, and BRK instructions. In each case, the register contents are displayed from the SAVE area at $45-49. However, there is no SAVE call after "execution" of these instructions, as there is for normally traced instructions, so the registers displayed are those present in the save area before execution of this instruction.

Therefore, on JSR and RTS, the displayed contents of the S-reg is incorrect. On the first instruction after a JSR or RTS the S-reg displays correctly (unless it is a JSR or RTS).

PROGRAM TO PROGRAM CONTROL INFORMATION TRANSFER

Page zero is heavily used differently by different software products as Integer BASIC, APPLESOFT, the Monitor. Sometimes it is desirable to place a few bytes of data somewhere where it will survive across program loads. As it happens, there are a few bytes available for such use.

CAUTION: I UNDERSTAND THAT DOS USES SOME OF THESE but I don't know which.

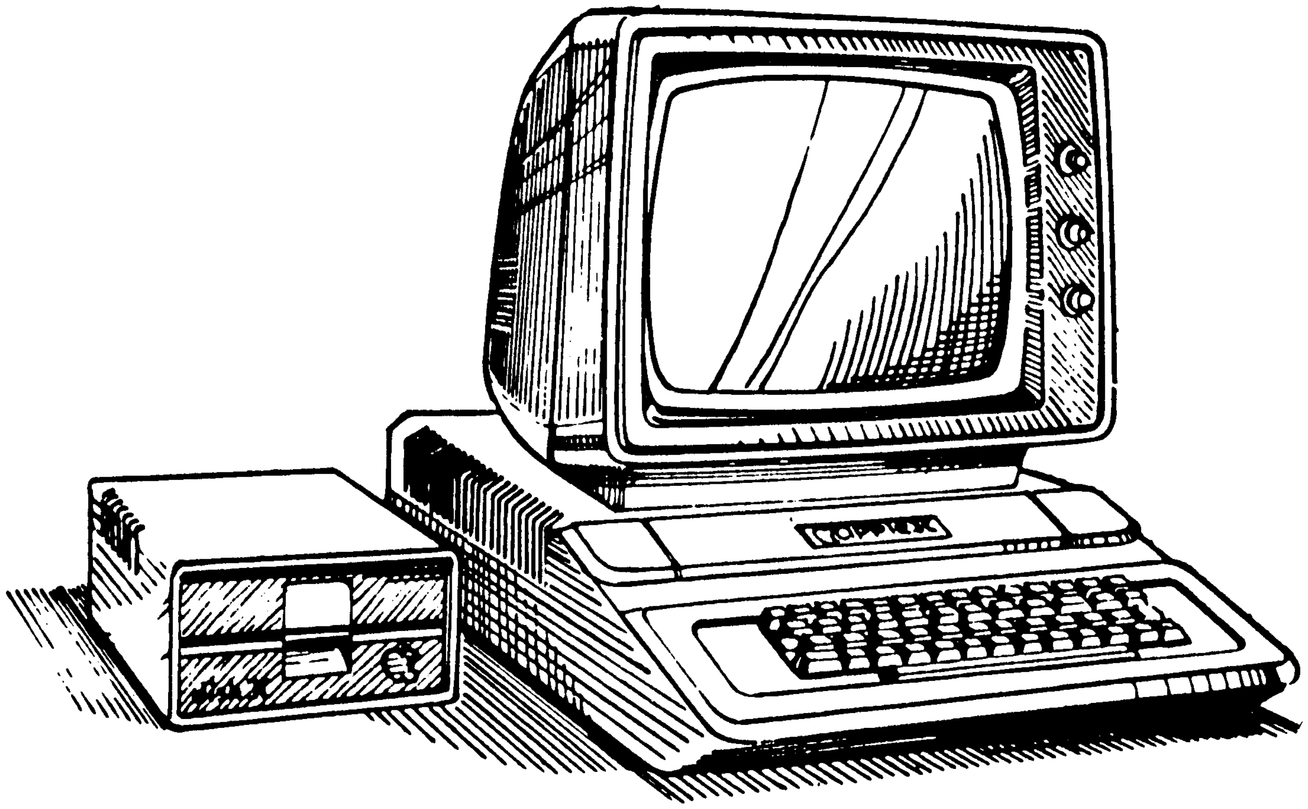
There are 1024 bytes allocated to the primary text page, $400-7FF. 24 lines of 40 characters adds up to 960 bytes. There are 64 bytes in that area which are not clobbered by text or plot or anything else I can find (except DOS, as indicated above). The available bytes are in eight groups of eight bytes.

Hex range	Decimal range
478-47F	1144-1151
4F8-4FF	1272-1279
578-57F	1400-1407
5F8-5FF	1528-1535
678-67F	1656-1663
6F8-6FF	1784-1791
778-77F	1912-1919
7F8-7FF	2040-2047

-END-

The Apple II MONITOR Peeled

The End



William E. Dougherty

May 1979