6 INTERRUPT PROCESSING

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

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

The system events that can cause interrupts are listed below:

chip-reset - power-up

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

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

Figure 6-1 List of System-Interrupt Events

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

CHIP-RESET

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

NONMASKABLE INTERRUPTS

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

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

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

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

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

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

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

The stage 1 VBLANK processor is executed.

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

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

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

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

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

- An RTI instruction is executed.

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

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

Stage 1 VBLANK Process

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

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

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

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

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

Stage 2 VBLANK Process

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

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

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

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

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

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

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

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

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

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

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

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

MASKABLE INTERRUPTS

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

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

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

- The 6502 A register is pushed to the stack.
- If the interrupt is due to serial I/O bus output ready, then clear the interrupt and jump through global RAM vector VSEROR [020C].
- If the interrupt is due to serial I/O bus input ready, then clear the interrupt and jump through global RAM vector VSERIN [020A].
- If the interrupt is due to serial I/O bus output complete, then clear the interrupt and jump through global RAM vector VSEROC [020E].
- If the interrupt is due to POKEY timer #1, then clear the interrupt and jump through global RAM vector VTIMR1 [0210].
- If the interrupt is due to POKEY timer #2, then clear the interrupt and jump through global RAM vector VTIMR2 [0212].
- If the interrupt is due to POKEY timer #4, then clear the interrupt. The service routine contains a bug, and falls into the following test.
- If pressing a keyboard key caused the interrupt (other than [BREAK], [START], [OPTION], or [SELECT]); then clear the interrupt and jump through global RAM vector VKEYBD [0208].
- If pressing the [BREAK] key caused the interrupt; then clear the interrupt. Set the BREAK flag BRKKEY [0011] to zero, proceed to clear the following:

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

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

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

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

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

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

INTERRUPT INITIALIZATION

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

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

Figure 6-2 Interrupt RAM Vector Initialization

System initialization sets the interrupt enable status as follows:

NMI VBLANK enabled, display list disabled.

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

SYSTEM TIMERS

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

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

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

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

USAGE NOTES

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

POKEY Interrupt Mask

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

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

```
; EXAMPLE OF INTERRUPT ENABLE

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

; EXAMPLE OF INTERRUPT DISABLE

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

Figure 6-3 POKEY Interrupt Mask Example

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

Setting Interrupt and Timer Vectors

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

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

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

JSR    SETVBV
```

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

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

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

Stack Content at Interrupt Vector Points

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

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

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

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

Miscellaneous Considerations

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

Restrictions on Clearing of "I" Bit

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

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

Interrupt Process Time Restrictions

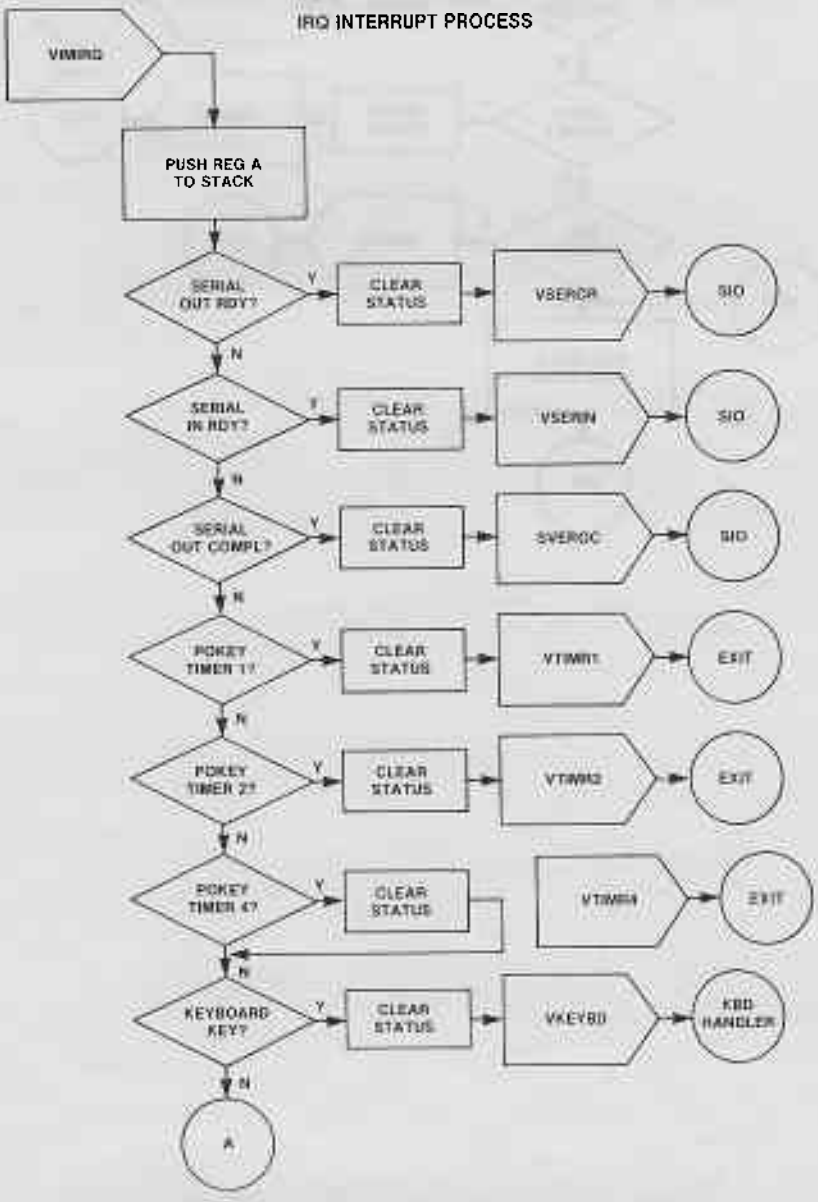
You should not write an interrupt routine that exceeds 400 msec. when added to the stage 1 VBLANK, if the serial I/O is being used. The SIO sets the CRITIC flag while serial bus I/O is in progress.

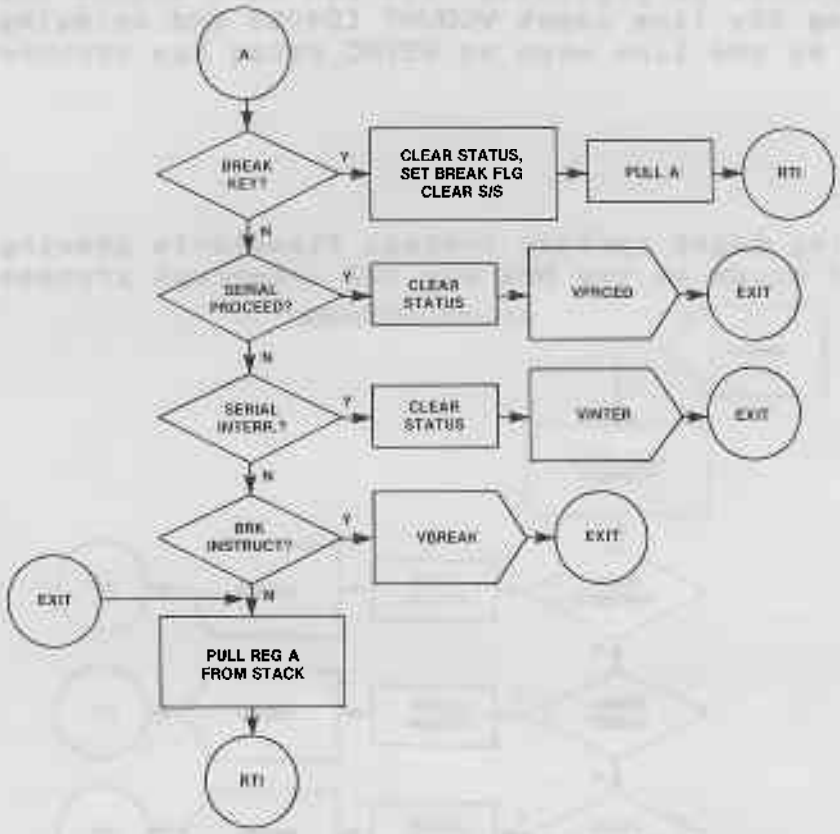
Interrupt Delay Due to "WAIT FOR SYNC"

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

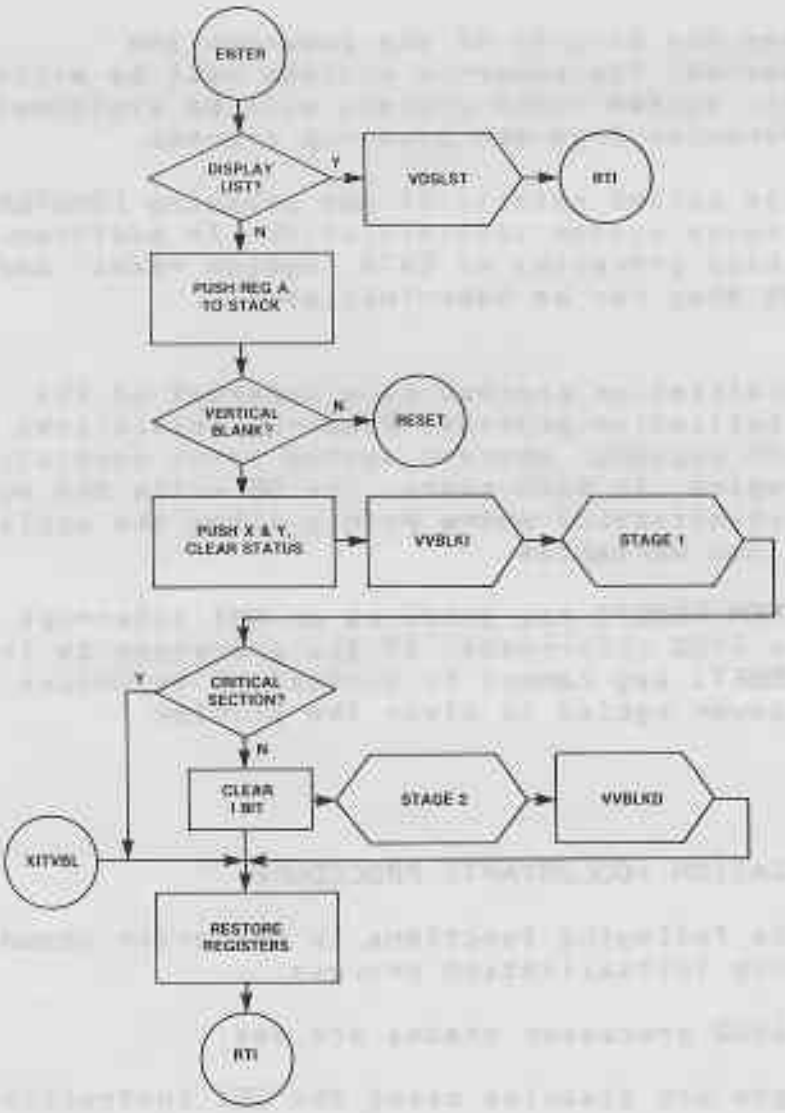
FLOWCHARTS

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




A
BREAK KEY?
Y
CLEAR STATUS, SET BREAK FLG CLEAR S/S
PULL A
RTI
N
SERIAL PROCEED?
Y
CLEAR STATUS
VPRCED
EXIT
N
SERIAL INTERR.?
Y
CLEAR STATUS
VINTER
EXIT
N
BRK INSTRUCT?
Y
VBREAK
EXIT
EXIT
N
PULL REG A FROM STACK
RTI

NMI INTERRUPT PROCESS



7 SYSTEM INITIALIZATION

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

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

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

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

POWER-UP INITIALIZATION (COLDSTART) PROCEDURE

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

1. The following 6502 processor states are set:

- IRQ interrupts are disabled using the SEI instruction.
- The decimal flag is cleared using the CLD instruction.
- The stack pointer is set to $FF.

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

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

   - Cartridge address BFFC = 00?
   - The memory at BFFC is not RAM?
   - Bit 7 of the byte at BFFD = 1?

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

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

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

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

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

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

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

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

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

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

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

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

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

- The following resident routines are called for initialization:

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

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

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

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

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

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

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

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

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

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

for details of the cassette-boot operation.

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

    There are no cartridges in the slots.

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

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

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

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

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

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

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

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

SYSTEM RESET INITIALIZATION (WARMSTART) PROCEDURE

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

A. Same as power-up step 1.

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

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

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

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

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

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

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

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

8 FLOATING POINT ARITHMETIC PACKAGE

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

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

The package provides the following operations:

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

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

```
FR0 [00D4] =     6-byte internal form of FP number.
FR1 [00E0] =     6-byte internal form of FP number.
FLPTR [00FC] =   2-byte pointer (lo,hi) to a FP.
                 number.
INBUFF [00F3] = 2-byte pointer (lo,hi) to an ASCII text
                 buffer.
CIX [00F2] =     1-byte index, used as offset to buffer
                 pointed to by INBUFF.
LBUFF [0580] =   result buffer for the FASC routine.
```

FUNCTIONS/CALLING SEQUENCES

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

ASCII to Floating Point Conversion (AFP)

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

Calling sequence:

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

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

FR0 = floating point number.
CIX = the buffer offset to the first byte after the ASCII
      number.
```

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

Floating Point to ASCII Conversion (FASC)

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

Calling sequence:

FR0 = floating point number.

```
JSR      FASC [D8E6]
```

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

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

Integer to Floating Point Conversion (IFP)

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

Calling sequence:

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

```
JSR      IFP [D9AA]
```

FR0 = floating point representation of integer.

Floating Point to Integer Conversion (FPI)

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

Calling sequence:

FR0 = floating point number.

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

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

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

Floating Point Addition (FADD)

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

Calling sequence:

```
     FR0 = floating point number.
     FR1 = floating point number.

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

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

Floating Point Subtraction (FSUB)

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

Calling sequence:

```
     FR0 = floating point minuend.
     FR1 = floating point subtrahend.

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

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

Floating Point Multiplication (FMUL)

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

Calling sequence:

```
     FR0 = floating point multiplier.
     FR1 = floating point multiplicand.

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

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

Floating Point Division (FDIV)

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

Calling sequence:

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

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

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

Floating Point Logarithms (LOG and LOG10)

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

Calling sequence:

```
     FR0 = floating point number.

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

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

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

The logarithm of a number Z is computed as follows:

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

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

Floating Point Exponentiation (EXP and EXP10)

Function: This routine exponentiates.

Calling sequence:

```
     FR0 = floating point exponent (Z).

     JSR       EXP [DDC0] for e ** Z
or
     JSR       EXP10 [DDCC] for 10 ** Z
     BCS       overflow.

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

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

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

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

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

Floating Point Polynomial Evaluation (PLYEVL)

Function: This routine performs an n degree polynomial evaluation.

Calling sequence:

```
     X,Y = pointer (X = LSB) to list of FP coefficients (A(i))
           ordered from high order to low order (six bytes per
           coefficient).
     A = number of coefficients in list.
     FR0 = floating point independent variable (Z).

     JSR       PLYEVL [DD40]
     BCS       overflow or other error.

     FR0 = result of A(n)*Z**n + A(n-1)*Z**n-1 ... + A(1)*Z +
           A(0).
     FR1 is altered.
```

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

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

Clear FRO (ZFRO)

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

Calling sequence:

```
JSR     ZFRO [DA44]

FRO = zero.
```

Clear Page Zero Floating Point Number (ZF1)

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

Calling sequence:

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

JSR     ZF1 [DA46]

Zero-page FP number(X) = zero.
```

Load Floating Point Number to FRO (FLDOR and FLDOP)

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

Calling sequences:

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

JSR     FLDOR [DD89]
```

or

```
FLPTR = pointer to FP number.

JSR     FLDOP [DD8D]

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

Load Floating Point Number to FR1 (FLD1R and FLD1P)

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

Calling sequences:

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

Store Floating Point Number From FR0 (FSTOR and FSTOP)

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

Calling sequence:

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

Move Floating Point Number From FR0 to FR1 (FMOVE)

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

Calling sequence:

```
JSR     FMOVE [DDB6]
```

FR1 = FR0 (FR0 remains unchanged).

RESOURCE UTILIZATION

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

00D4 through 00FF
057E through 05FF

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

IMPLEMENTATION DETAILS

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

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

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

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

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

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

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

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

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

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

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

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

9 ADDING NEW DEVICE HANDLERS/PERIPHERALS

This section describes the interface requirements for a nonresident Device Handler that is to be accessed via the Central I/O utility (CIO). The Serial bus I/O utility (SIO) interface is defined for those handlers that utilize the Serial I/O bus.

The I/O subsystem is organized with three levels of software between you and your hardware: The CIO, the individual device handlers, and the SIO.

The CIO performs the following functions:

Logical device name to Device Handler mapping (on OPEN).

I/O Control Block (IOCB) maintenance.

Logical record handling.

User buffer handling.

The device handlers are below CIO. They perform the following functions:

Device initialization on power-up and system reset.

Device-dependent support of OPEN and CLOSE commands.

Byte-at-a-time data input and output.

Device-dependent special operations.

Device-dependent command support.

Device data buffer management.

The SIO is at the bottom level (for Serial I/O bus peripheral handlers). It performs the following functions:

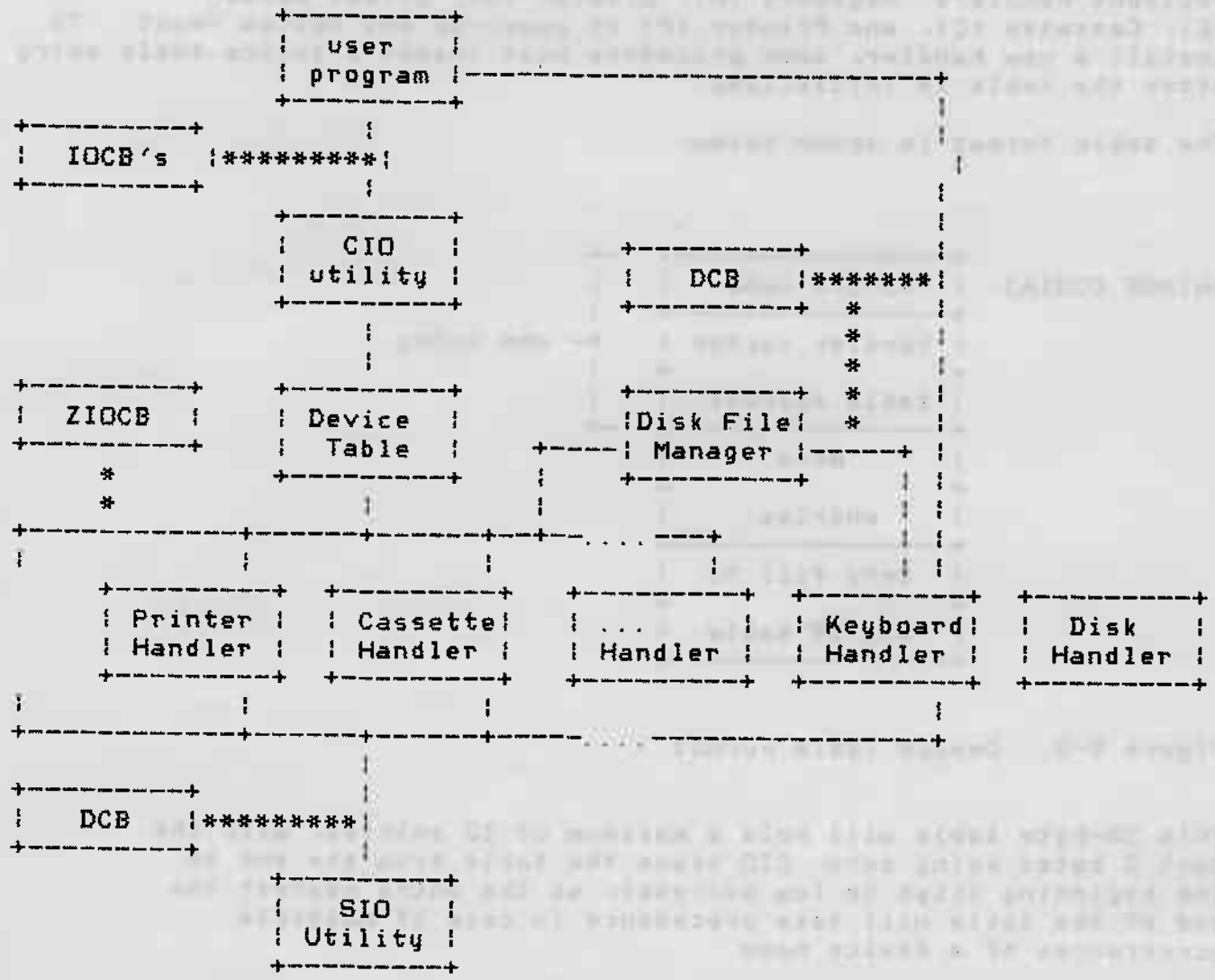
Control of all Serial bus I/O, conforming to the bus protocol.

Bus operation retries on errors.

Return of unified error statuses on error conditions.

A separate control structure is used for communication at each interface, as follows:

User/CIO	I/O Control Block (IOCB)
CIO/Handler	Zero-page IOCB (ZIOCB)
Handler/SIO	Device Control Block (DCB)



Where: ---- shows a control path.
**** shows the data structure required for a path.

Note the following:

1. The Keyboard/Display/Screen Editor handlers don't use SIO.
2. The Diskette Handler cannot be called directly from CIO.
3. The DCB is shown twice in the diagram.

Figure 9-1 I/O Subsystem Flow Diagram

DEVICE TABLE

The device table is a RAM-resident table that contains the single-character device name (e.g. K, D, C, etc). and the handler address for each of the handlers known to CIO. The table is initialized to contain entries for the following resident handlers: Keyboard (K), Display (S), Screen Editor (E), Cassette (C), and Printer (P) at power-up and system reset. To install a new handler, some procedure must insert a device table entry after the table is initialized.

The table format is shown below:

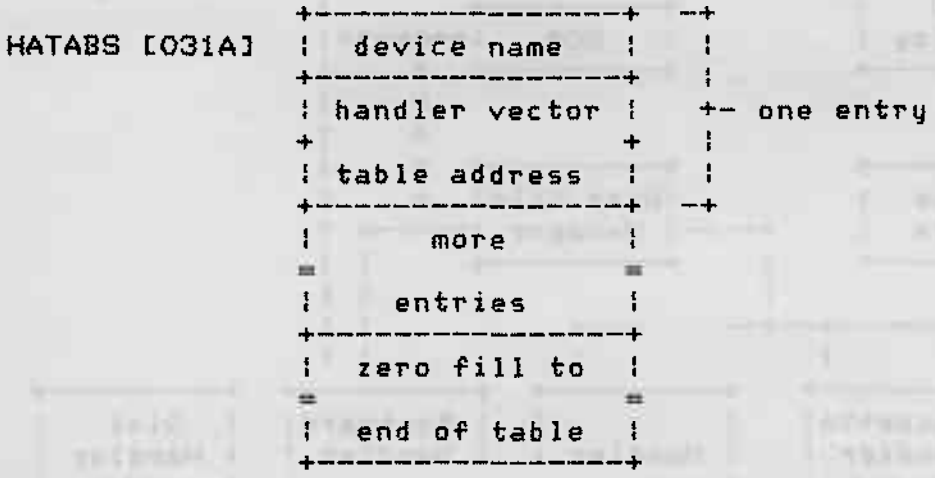


Figure 9-2 Device Table Format

This 38-byte table will hold a maximum of 12 entries, with the last 2 bytes being zero. CIO scans the table from the end to the beginning (high to low address); so the entry nearest the end of the table will take precedence in case of multiple occurrences of a device name..

The device name for each entry is a single ATASCII character, and the handler address points to the handler's vector table, that will be described in the following section.

CIO/HANDLER INTERFACE

This section describes the interface between the Central I/O utility and the individual device handlers that are represented in the Device Table (as described in the preceding section).

Calling Mechanism

Each handler has a vector table as shown below:

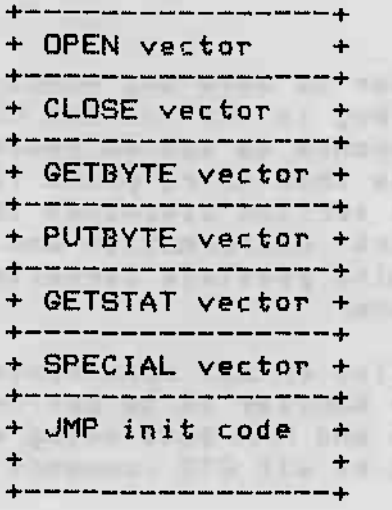


Figure 9-3 Handler Vector Table

The device table entry for the handler points to the first byte of the vector table.

The first six entries in the table are vectors (lo,hi) that contain the address - 1 of the handler routine that handles the indicated function. The seventh entry is a 6502 JMP instruction to the handler initialization routine. CIO uses only the addresses contained in this table for handler entry. Each user/CIO command translates to one or more calls to one of the handler entries defined in the vector table.

The vector table provides the handler addresses for certain fixed functions to be performed to CIO. In addition, operation parameters also must be passed for most functions. Parameter passing is accomplished using the 6502 A, X, and Y registers and an IOCB in page 0 named ZIOCB [0020]. In general, register A is used to pass data, register X contains the index to the originating IOCB, and register Y is used to pass status information to CIO. The zero-page IOCB, is a copy of the originating IOCB; but in the course of processing some commands, CIO can alter the buffer address and buffer length parameters in ZIOCB, but not in the originating IOCB (see Section 5 for information relating to the originating IOCB).

See Appendix B for the standard status byte values to be returned to CIO in register Y.

The following sections describe the CIO/handler interface for each of the vectors in the handler vector table.

Handler Initialization

NOTE: This entry doesn't appear to have any function for nonresident handlers due to a bug in the current OS -- the device table is cleared in response to system reset as well as power-up. This prevents this entry point from ever being called. The rest of this section discusses the intended use of this entry point. Conformation would be in order to allow compatibility with possible corrected versions of the OS in the future.

The entry was to have been called on all occurrences of power-up and system reset; the handler is to perform initialization of its hardware and RAM data using a routine that assures proper processing of all CIO commands that follow.

Functions Supported

This section describes the functions associated with the first six vectors from the handler vector table. This section also presents a brief, device-independent description of the CIO/handler interface and recommended actions for each function vector.

OPEN

This entry is called in response to an OPEN command to CIO. The handler is expected to validate the OPEN parameters and perform any required device initialization associated with a device OPEN.

At handler entry, the following parameters can be of interest:

X = index to originating IOCB.
Y = $92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).
ICBALZ/ICBAHZ [0024/0025] = address of device/filename specification.
ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler attempts to perform the indicated OPEN and indicates the status of the operation by the value of the Y register. The responsibility for checking for multiple OPENs to

the same device or file, where it is illegal, lies with the handler.

CLOSE

This vector table entry is called in response to a CLOSE command to CIO. The handler is expected to release any held resources that relate specifically to that device/filename, and for output files to:

1) send any data remaining in handler buffers to the device,

2) mark the end of file

3) update any associated directories, allocation maps, etc.

At handler entry, the following parameters can be of interest:

X = index to originating IOCB.
Y = $92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).
ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler attempts to perform the indicated CLOSE and indicates the status of the operation by the value of the Y register.

CIO releases the associated IOCB after the handler returns, regardless of the operation status value.

GETBYTE

This vector table entry is called in response to a GET CHARACTERS or GET RECORD command to CIO. The handler is expected to return a single byte in the A register, or return an error status in the Y register.

At handler entry, the following parameters can be of interest:

X = index to originating IOCB.
Y = $92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).
ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler will obtain a data byte directly from the device or from a handler-maintained buffer and return to CIO with the byte in the A register and the operation status in the Y register.

Handlers that do not have short timeouts associated with the reading of data (such as the Keyboard and Cassette Handlers), must monitor the [BREAK] key flag BRKKEY [0011] and return with a status of $80 when a [BREAK] condition occurs. See Appendix L, E5; and Section 12 for a discussion of [BREAK] key monitoring.

CIO checks for reads from device/files that have not been opened or have been opened for output only; the handler will not be called in those cases.

PUTBYTE

This entry is called in response to a PUT CHARACTERS or PUT RECORD command to CIO. The handler is expected to accept a single byte in the A register or return an error status in the Y register.

At handler entry, the following parameters can be of interest:

X = index to originating IOCB.
Y = $92 (status = function not implemented by handler).
A = data byte.

ICDNOZ [0021] = device number (1-4, for multiple device handlers).
ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler sends the data byte directly to the device, or to a handler-maintained buffer, and returns to CIO with the operation status in the Y register. If a handler-maintained buffer fills, the handler will send the buffered data to the device before returning to CIO.

CIO checks for WRITEs to device/files that have not been opened, or have been opened for input only. The handler will not be called in those cases.

Now that the normal operation of PUTBYTE has been defined, a special case must be added. Any handler that will operate within the environment of the ATARI 8K BASIC language interpreter has a different set of rules. Because BASIC can call the handler PUTBYTE entry directly, without going through CIO, the zero-page IOCB (ZIOCB) can or may not have a relation to the PUTBYTE call. Therefore, the handler must use the outer level IOCB to obtain any information that would normally be obtained from ZIOCB. Note also that the OPEN protection normally provided by CIO is bypassed (i.e. PUTBYTE to a non-OPEN device/file and PUTBYTE to a read-only OPEN).

GETSTAT

This entry is called in response to a GET STATUS command to CIO. The handler is expected to return four bytes of status to memory or return an error status in the Y register.

At handler entry, the following parameters can be of interest:

X = index to originating IOCB. Y = $92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).
ICBALZ/ICBAHZ [0024/0025] = address of device/filename specification.
ICAX1Z/ICAX2Z
[002A/002B] = device-specific information.

The handler gets device status information from the device controller and puts the status bytes in DVSTAT [02EA] through DVSTAT+3, and finally returns to CIO with the operation status in register Y.

The IOCB need not be opened nor closed in order for you to request CIO to perform a GET STATUS operation; the handler must check where there are restrictions. See Section 5 for a discussion of the CIO actions involved with a GET STATUS operation using both open and closed IOCB's, and note the impact of this operation on the use of the buffer address parameter.

SPECIAL

This handler entry is used to support all functions not handled by the other entry points, such as diskette file RENAME, display DRAW, etc. Specifically, if the IOCB command byte value is greater than $0D, then CIO will use the SPECIAL entry point. The handler must interrogate the command byte to determine if the requested operation is supported.

At handler entry, the following parameters can be of interest:

X = index to originating IOCB.
Y = $92 (status = function not implemented by handler).

ICDNOZ [0021] = device number (1-4, for multiple device handlers).
ICCOMZ [0022] = command byte.
ICBALZ/ICBALH [0024/0025] = buffer address.
ICBLLZ/ICBLHZ [0028/0029] = buffer length.
ICAX1Z/ICAX2Z [002A/002B] = device-specific information.

The handler will perform the indicated operation, if possible, and return to CIO with the operation status in register Y.

The IOCB need not be opened nor closed in order for you to request CIO to perform a SPECIAL operation; the handler must check where there are restrictions. See Section 5 for a discussion of the CIO actions involved with a SPECIAL operation using both open and closed IOCB's, and note the impact of this on the use of the buffer address parameter.

Error Handling

Error handling has been simplified somewhat by having CIO handle outer level errors and having SIO handle Serial bus errors, leaving the handler to process the remaining errors. These errors include:

- out-of-range parameters.
- [BREAK] key abort.
- Invalid command.
- Read after end of file.

The current handlers respond to errors using the following guidelines:

- They keep the recovery simple (and therefore predictable and repeatable).
- They Do not interact directly with you for recovery instructions.
- They lose as little data as possible.
- They make all attempts to maintain the integrity of file oriented device storage -- this involves the initial design of the structural elements as well as error recovery techniques.

Resource Allocation

Nonresident handlers needing code and/or data space in RAM should use the techniques listed below, to assure nonconflict with other parts of the OS, including other nonresident handlers.

Zero-Page RAM

Zero-page RAM has no spare bytes, and even if there were, there is no allocation scheme to support multiple program assignment of the spares. Therefore, the nonresident handler must save and restore the bytes of zero-page RAM it is going to use. The bytes to use must be chosen carefully, according to the following criteria:

The bytes cannot be accessed by an interrupt routine.

The bytes cannot be accessed by any noninterrupt code between the time the handler modifies the bytes and then restores the original values.

A simple save/restore technique would utilize the stack in a manner similar to that shown below:

```
        LDA     COLCRS          ; (for example)
        PHA                     ; SAVE ON STACK.
        LDA     COLCRS+1
        PHA

        LDA     HPOINT          ; HANDLER'S POINTER.
        STA     COLCRS
        LDA     HPOINT+1
        STA     COLCRS+1

        XXX     (COLCRS),Y      ; DO YOUR POINTER THING.

        PLA                     ; RESTORE OLD DATA.
        STA     COLCRS+1
        PLA
        STA     COLCRS
```

Note that the Display Handler or Screen Editor should not be called before restoring the original value of COLCRS, because COLCRS is a variable used by those routines.

Nonzero-Page RAM

There is no allocation scheme to support the assignment of fixed regions of nonzero-page RAM to any specific process, so the handler has three choices:

1. Make a dynamic allocation at initialization time by altering MEMLO [02E7].

2. Include the variables with the handler for RAM-resident handlers. This still involves altering MEMLO at the time the handler is booted.

3. If the handler replaces one of the resident handlers (by removing the resident handler's entry in the device table), then the new handler can use any RAM that the

formerly resident handler would have used.

Stack Space

In most cases, there are no restrictions on the use of the stack by a handler. However, if the handler plans to push more than a couple dozen bytes to the stack; then it should do a stack overflow test, and always leave stack space for interrupt processing.

HANDLER/SIO INTERFACE

This section describes the interface between serial bus device handlers and the serial bus I/O utility (SIO). SIO completely handles all bus transactions following the device-independent bus protocol. SIO is responsible for the following functions:

Bus data format and timing from computer end.

Error detection, retries and statuses.

Bus timeout.

Transfer of data between the bus and the caller's buffer.

Calling Mechanism

SIO has a single entry point SIOV [E459] for all operations. The device control block (DCB) [0300] contains all parameters passed to SIO. The DCB contains the following bytes:

DEVICE BUS ID -- DDEVIC [0300]

The bus ID of the device is set by the handler prior to calling SIO (see Appendix I).

DEVICE UNIT # -- DUNIT [0301]

This byte indicates that of n units of a given device type to access, and is set by the handler prior to calling SIO. This value usually comes from ICDNOZ. SIO accesses the bus device whose address is equal to the value of DDEVIC plus DUNIT minus 1 (the lowest unit number is normally equal to 1).

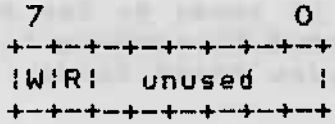
DEVICE COMMAND -- DCOMND [0302]

The handler sets this byte prior to calling SIO. It will be sent to the bus device as part of the command frame. See Appendix I for device command byte values.

DEVICE STATUS -- DSTATS [0303]

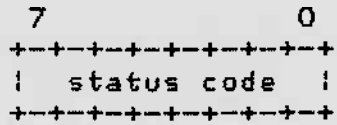
This byte is bidirectional. The handler will use DSTATS to indicate to SIO what to do after the command frame is sent and acknowledged. SIO will use it to indicate to the handler the status of the requested operation.

Prior to an SIO call:



Where: W,R = 0,0 indicates no data transfer is associated with the operation.
0,1 indicates a data frame is expected from the device.
1,0 indicates a data frame is to be sent to the device.
1,1 is invalid.

After an SIO call:



See Appendix C for the possible SIO operation status codes.

HANDLER BUFFER ADDRESS -- DBUFLO/DBUFHI [0304/0305]

The handler sets this 2-byte pointer. It indicates the source or destination buffer for device data or status information.

DEVICE TIMEOUT -- DTIMLO [0306]

The handler sets this byte. It specifies the device timeout time in units of 64/60 of a second. For example, a count of 6 specifies a timeout of 6.4 seconds.

BUFFER LENGTH/BYTE COUNT -- DBYTLO/DBYTHI [0308/0309]

The handler sets this 2-byte count for the current operation, and indicates the number of data bytes to be transferred into or out of the buffer. This parameter is not required if the STATUS byte W and R bits are both zero. These values indicate that no data transfer is to take place.

> WARNING: There is a bug in SIO that causes incorrect actions when the last byte of a buffer is in a memory address ending in $FF, such as 13FF, 42FF, etc.

AUXILIARY INFORMATION -- DAUX1/DAUX2 [030A/030B]

The handler sets these 2-bytes. The SIO includes them in the bus command frame; they have device-specific meanings.

Functions Supported

SIO does not examine the COMMAND byte it sends to the device, because all bus transactions are expected to conform to a universal protocol. The protocol includes three forms, stated below (as seen from the computer):

Send command frame.

Send command frame and send data frame.

Send command frame and receive data frame.

The values of the W and R bits in the status byte select the command form.

Error Handling

SIO handles most of the serial bus errors for the handler, as indicated below:

Bus timeout -- SIO provides a uniform command frame and data frame ACK byte timeout of 1/60 of a second - 0 / + 1/60. The handler specifies the maximum COMPLETE byte timeout value in DTIMLO.

Bus errors -- SIO detects and reports UART overrun and framing errors. The sensing of these errors in any received byte will cause the entire associated frame to be considered bad.

Data frame checksum error -- SIO validates the checksum on all received data frames and generates a checksum for all transmitted frames.

Invalid response from device -- In addition to the error conditions stated above, SIO checks that the ACK and COMPLETE responses are proper (ACK = $41 and COMPLETE = $43). ACK stands for acknowledge.

Bus operation retries -- SIO will attempt one complete command retry if the first attempt is not error free, where a complete command try consists of up to 14 attempts to send (and acknowledge) a command frame, followed by a single attempt to

receive the COMPLETE code and possibly a data frame.

> NOTE: There is a bug in the retry logic for data writes, such that if the command frame is acknowledged by the controller, but the data frame is not acknowledged, then SIO will retry indefinitely.

Unified error status codes -- SIO provides device-independent error codes (see Appendix C).

SERIAL I/O BUS CHARACTERISTICS AND PROTOCOL

This section describes:

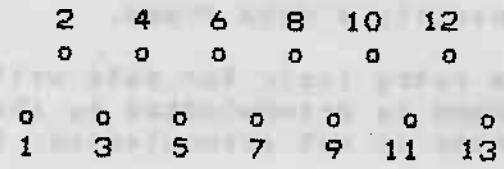
- The electrical characteristics of the ATARI 400 and ATARI 800 Home Computers serial bus
- The use of the bus to send bytes of data,
- The organization of the bytes as "frames" (records),
- The overall command sequences that utilize frames and response bytes to provide computer/peripheral communication.

Hardware/Electrical Characteristics

The ATARI 400 and the ATARI 800 Home Computers communicate with peripheral devices over a 19,200 baud asynchronous serial port. The serial port consists of a serial DATA OUT (transmission) line, a serial DATA IN (receiver) line and other miscellaneous control lines.

Data is transmitted and received as 8 bits of serial data (LSB sent first) preceded by a logic zero start bit and succeeded by a logic one stop bit. The serial DATA OUT is transmitted as positive logic (+4v = one/true/high, 0v = zero/false/low). The serial DATA OUT line always assumes its new state when the serial CLOCK OUT line goes high; CLOCK OUT then goes low in the center of the DATA OUT bit time.

An end view of the Serial bus connector at the computer or peripheral is shown below (the cable connectors would of course be a mirror image):



where: 1 = computer CLOCK IN.
2 = computer CLOCK OUT.
3 = computer DATA IN.
4 = GND.
5 = computer DATA OUT.
6 = GND.
7 = COMMAND-.
8 = MOTOR CONTROL.
9 = PROCEED-.
10 = +5v/READY.
11 = computer AUDIO IN.
12 = +12v.
13 = INTERRUPT-.

Figure 9-4 Serial Bus Connector Pin Descriptions

CLOCK IN is not used by the present OS and peripherals. This line can be used in future synchronous communications schemes.

CLOCK OUT is the serial bus clock. CLOCK OUT goes high at the start of each DATA OUT bit and returns to low in the middle of each bit.

DATA IN is the serial bus data line to the computer.

Pin 4 GND is the signal/shield ground line.

DATA OUT is the serial bus data line from the computer.

Pin 6 GND is the signal/shield ground line.

COMMAND- is normally high and goes low when a command frame is being sent from the computer.

MOTOR CONTROL is the cassette motor control line (high=on, low= off).

PROCEED- is not used by the present OS and peripherals; this line is pulled high.

+5v/READY indicates that the computer is turned on and ready. This line can also be used as a +5 volt supply of 50ma current rating for ATARI peripherals only.

AUDIO IN accepts an audio signal from the cassette.

+12V is a +12 volt supply of unknown current rating for ATARI peripherals only.

INTERRUPT- is not used by the present OS and peripherals; this line is pulled high.

There are no pin reassignments made in the Serial bus cable, so pin 3, the computer's DATA IN line, is the peripheral's data output line; and similarly for pin 5.

Serial Port Electrical Specifications

Peripheral input:

V1H = 2.0v min.
V1L = 0.4v max.

I1H = 20ua. max. @ V1H = 2.0v
I1L = 5ua. max. @ V1L = .4v

Peripheral output (open collector bipolar):

VOL = 0.4v max. @ 1.6 ma.
VOH = 4.5v min. with external 100Kohm pull-up.

Vcc/READY input:

V1H = 2.0v min. @ I1H = 1ma. max.
V1L = 0.4v max.
Input goes to logic zero when open.

Bus Commands

The bus protocol specifies that all commands must originate from the computer, and that peripherals will present data on the bus only when commanded to. Every bus operation will go to completion before another bus operation is initiated (no overlap). An error detected at any point in the command sequence will abort the entire sequence.

A bus operation consists of the following elements:

Command frame from the computer.

Acknowledgement (ACK) from the peripheral.

Optional data frame to or from the computer.

Operation complete (COMPLETE) from the peripheral.

Command Frame

The serial bus protocol provides for three types of commands: 1) data send, 2) data receive and 3) immediate (no data -- command only). There is a common element in all three types, a command frame consisting of five bytes of information sent from the computer while the COMMAND- line is held low. The format of the command frame is
shown below:

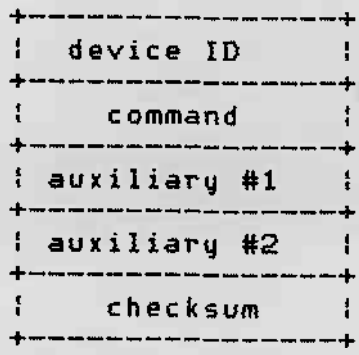


Figure 9-5 Serial Bus Command Frame Format

The device ID specifies that of the serial bus devices is being addressed (see Appendix I for a list of device IDs).

The command byte contains a device-dependent command (see Appendix I for a list of device commands).

The auxiliary bytes contain more device-dependent information.

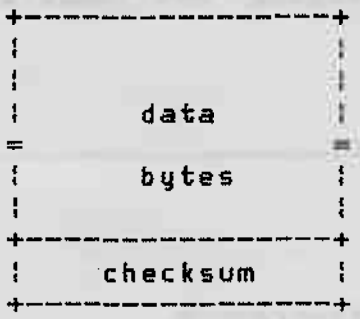
The checksum byte contains the arithmetic sum of the first four bytes (with the carry added back after every addition).

Command Frame Acknowledge

The peripheral being addressed would normally respond to a command frame by sending an ACK byte ($41) to the computer; if there is a problem with the command frame, the peripheral should not respond.

Data Frame

Following the command frame (and ACK) can be an optional data frame that is formatted as shown below:



This data frame can originate at the computer or at the device controller, depending upon the command. Current device controllers expect fixed-length data frames as does the computer, where the data frame length is a fixed function of the device type and command.

The checksum value in the data frame is the arithmetic sum of all of the frame data preceding the checksum, with the carry from each addition being added back (the same as for the command frame).

In the case of the computer sending a data frame to a peripheral, the peripheral is expected to send an ACK if the data frame is acceptable, and send a NAK ($4E), or do nothing if the data frame is unacceptable. See the first flowchart in Section 9.

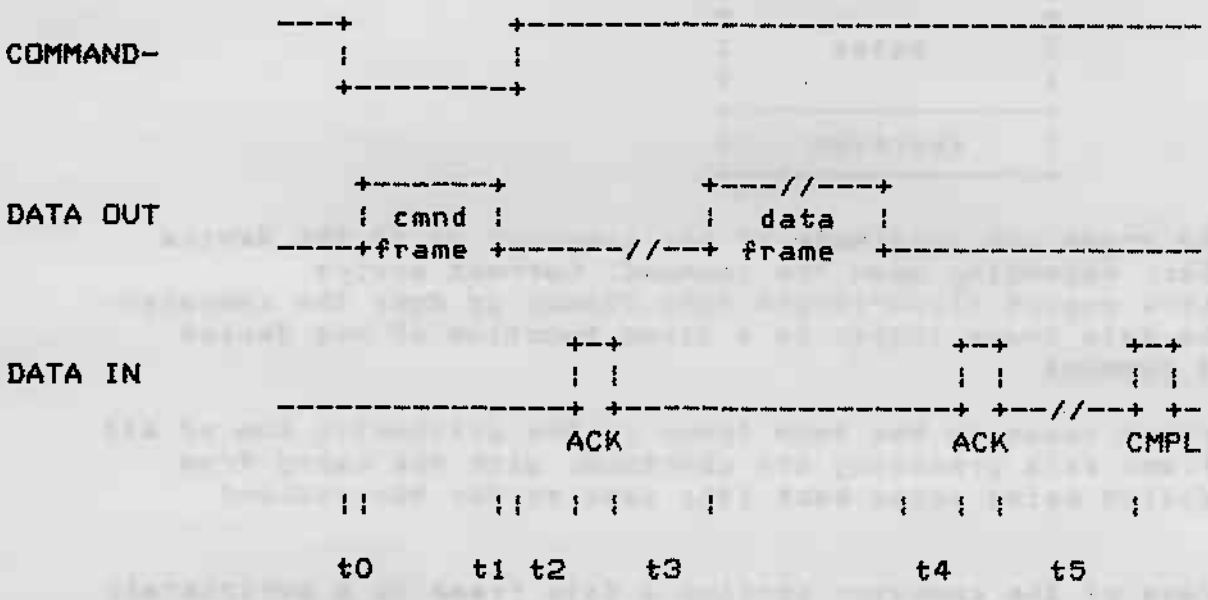
Operation Complete

A peripheral is also expected to send an operation-COMPLETE byte ($43) at the time the commanded operation is complete. The location of this byte in the command sequence for each command type is shown in the timing diagrams in Section 9. If the operation cannot go to normal, error-free completion, the peripheral should respond with an ERROR byte ($45) instead of COMPLETE.

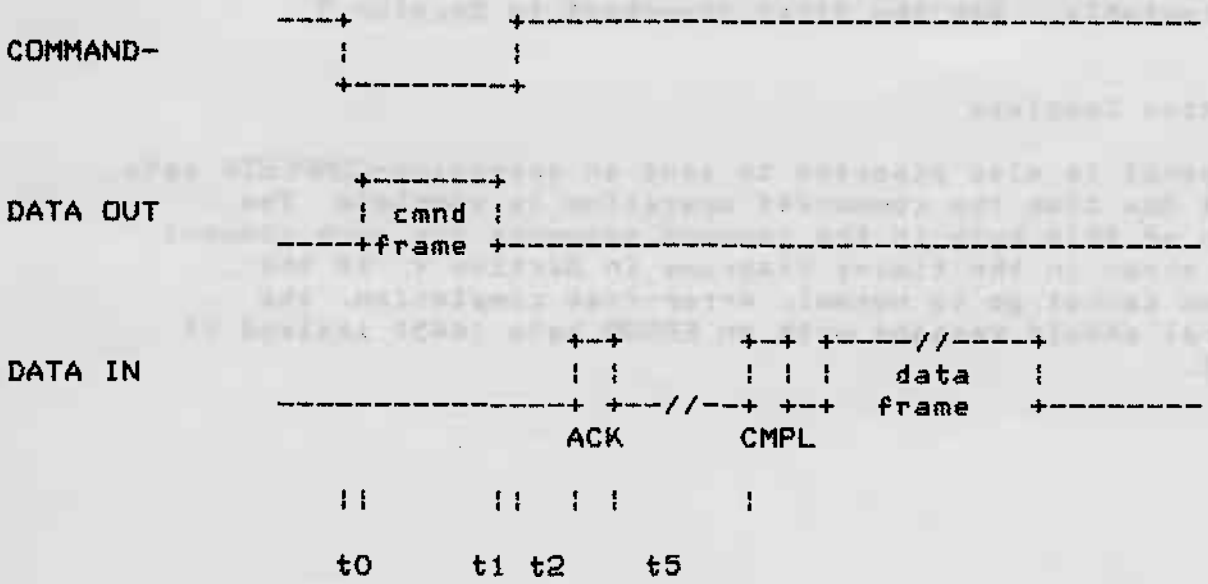
Bus Timing

This section provides timing diagrams for the three types of command sequences: data send, data receive, and immediate.

DATA SEND sequence:



DATA RECEIVE sequence:



IMMEDIATE sequence:

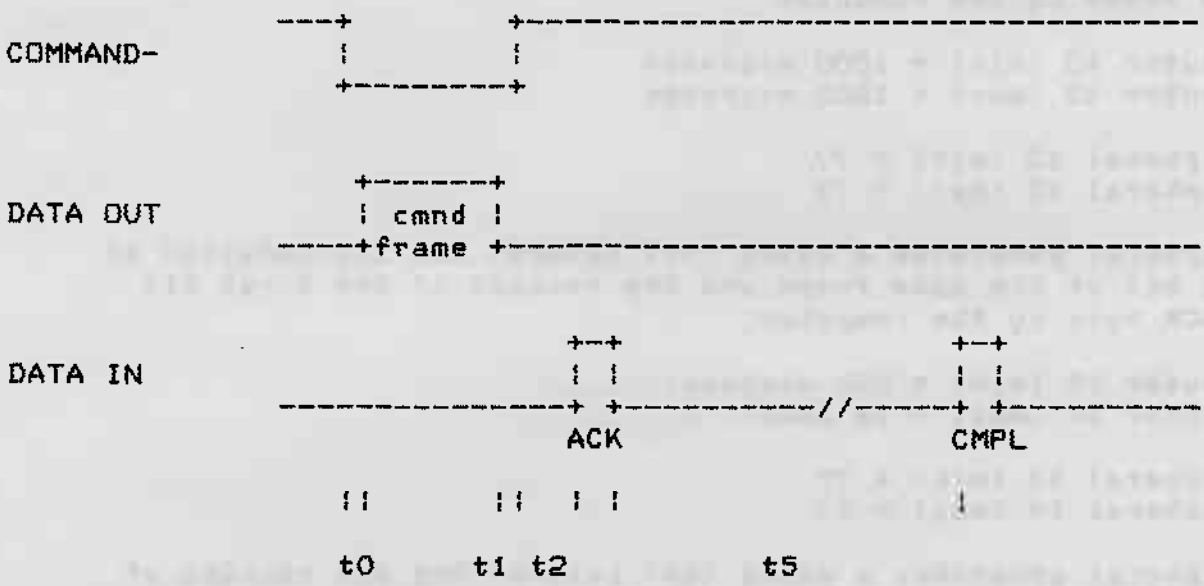


Figure 9-6 Serial Bus Timing Diagram

The computer generates a delay (t0) between the lowering of COMMAND- and the transmission of the first byte of the command frame.

computer t0 (min) = 750 microsec.
computer t0 (max) = 1600 microsec.

peripheral t0 (min) = ??
peripheral t0 (max) = ??

The computer generates a delay (t1) between the transmission of the last bit of the command frame and the raising of the COMMAND- line.

computer t1 (min) = 650 microsec.
computer t1 (max) = 950 microsec.

peripheral t1 (min) = ??
peripheral t1 (max) = ??

The peripheral generates a delay (t2) between the raising of COMMAND- and the transmission of the ACK byte by the peripheral.

computer t2 (min) = 0 microsec.
computer t2 (max) = 16 msec.

peripheral t2 (min) = ??
peripheral t2 (max) = ??

The computer generates a delay (t3) between the receipt of the last bit of the ACK byte and the transmission of the first bit of the data frame by the computer.

computer t3 (min) = 1000 microsec.
computer t3 (max) = 1800 microsec.

peripheral t3 (min) = ??
peripheral t3 (max) = ??

The peripheral generates a delay (t4) between the transmission of the last bit of the data frame and the receipt of the first bit of the ACK byte by the computer.

computer t4 (min) = 850 microsec.
computer t4 (max) = 16 msec.

peripheral t4 (min) = ??
peripheral t4 (max) = ??

The Peripheral generates a delay (t5) between the the receipt of the last bit of the ACK byte and the first bit of the COMPLETE byte by the computer.

computer t5 (min) = 250 microsec.
computer t5 (max) = 255 sec. (handler-dependent)

peripheral t5 (min) = ??
peripheral t5 (max) = N/A

HANDLER ENVIRONMENT

Nonresident handlers can be installed in at least three different manners:

1. As booted software from diskette or cassette.

2. Resident in a cartridge (A or B).

3. Downloaded from a serial bus device.

This section will discuss the basic mechanisms for handler installation for these environments. In order to fully utilize the information in this section, you must have read and understood the following sections:

Bootable Handler

The diskette- or cassette-booted software will insert the handler's vector table pointer and name to the device table whenever the booted software's initialization entry point is entered (on power-up and system reset). Remember that both power-up and system reset clear the device table of all but the resident handler entries.

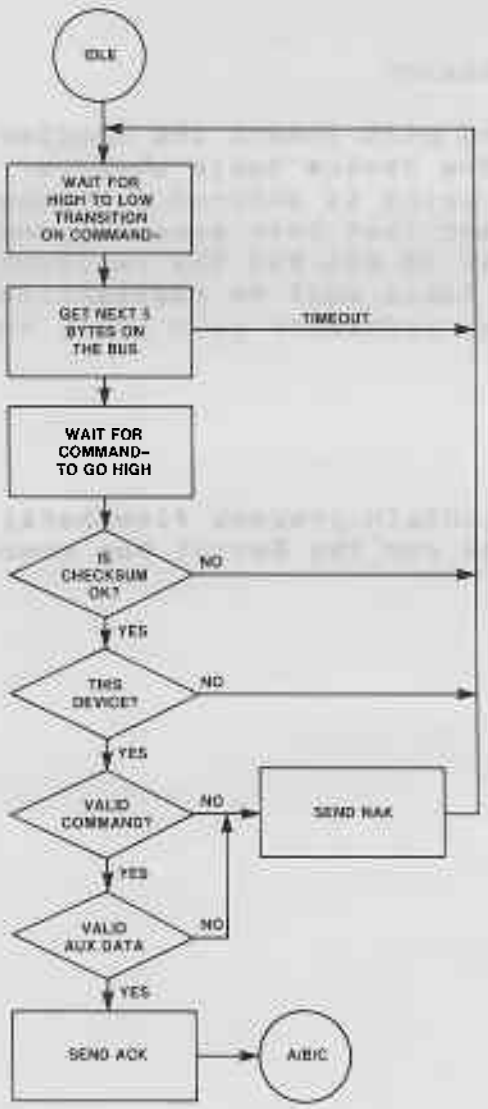
Cartridge Resident Handler

The cartridge software will insert the handler's vector table pointer and name to the device table whenever the cartridge's initialization entry point is entered (on power-up and system reset). Remember that both power-up and system reset clear the device table of all but the resident handler entries; therefore the device table must be reestablished by the handler-initialization procedure upon every entry.

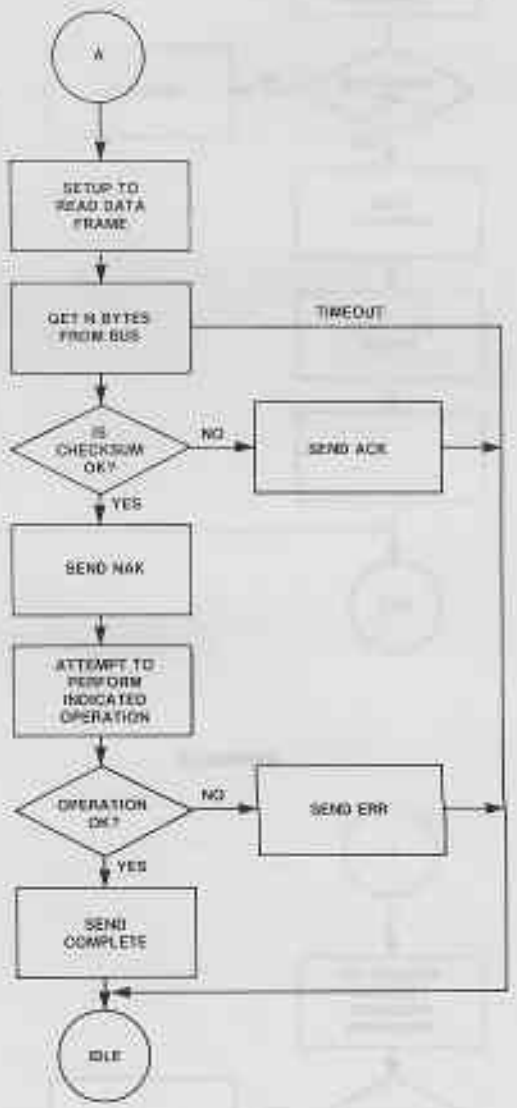
FLOWCHARTS

The following pages contain process flowcharts showing the SIO and peripheral actions for the Serial bus command forms.

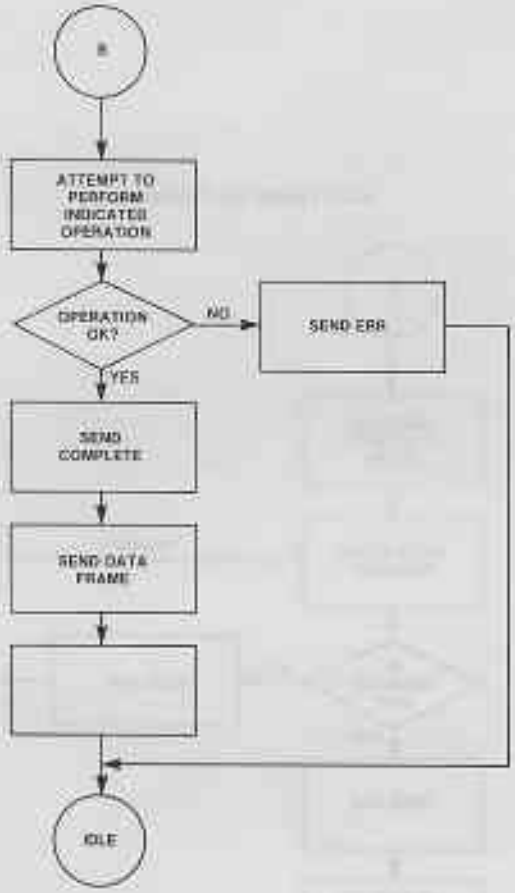
PERIPHERAL'S COMMAND FRAME PROCESSING



DATA FRAME TO PERIPHERAL



DATA FRAME TO COMPUTER



IMMEDIATE