2600/7800 DEVELOPMENT KIT

CARE AND FEEDING INSTRUCTIONS

A full, complete 2600/7800 development system includes the following:

HARDWARE:

- 520ST or 1040ST COMPUTER (or Mega ST))
- SH204 HARD DISK
- SF354 or SF314 FLOPPY DISK (or equivalent)
- SM124 or SC1224 MONITOR (or equivalent)
- 7800 GAME MACHINE
- TRANSFER CABLE
- DEVELOPMENT RAM CARTRIDGE
- TELEVISION (not furnished by Atari)
- EPROM CARTRIDGE (for appropriate game system)

SOFTWARE:

- DEVELOPMENT KIT DISK:
  - 40COL.S (Sample source code for 40-column text display)
  - ASM.BAT (Submit file for the batch utility)
  - BATCH.TTP (Batch utility: executes programs in submit file)
  - CONV65.PRG (Converts .0 file to S-record format ".SR")
  - DLOAD.TTP (Downloads code to the 7800 and acts as terminal)
  - K.TTP ("Kermit" - for inter-computer communication)
  - MAC.PRG (cross-assembler)
  - MARIA.S (Recommended 7800 equates file)
  - STELLA.S (Recommended 2600 equates file)
  - ME.TTP (Micro-Emacs editor)
  - WAIT.PRG (Allows messages to remain on screen until <CR>)
- HARD DISK BOOT DISK (comes with hard disk drive)
- DS5027 NEOCHROME (ST graphics program)
- SAMPLE SOURCE CODE (for the appropriate game system)

DOCUMENTATION FOR 2600 AND 7800 DEVELOPMENT:

ASSEMBLER MANUAL
EDITOR MANUAL
STELLA (2600) PROGRAMMER'S GUIDE
2600 GAME STANDARDS AND PROCEDURES
7800 SUPER CART SPEC
7800 PROGRAMMING GUIDE
SARA PROGRAAMING INSTRUCTIONS
MOTOROLA S RECORDS FORMAT
7800 GAME STANDARDS AND PROCEDURES
NOTE FREQUENCIES FOR ATARI 2600/7800 SOUND SYSTEM

7800 SCHEMATIC
2600 SCHEMATIC
DESCRIPTION OF 7800 "PRO" DEVELOPMENT SYSTEM UPGRADE (debugger commands)

...AND ANY OTHER DOCUMENTS WHICH MAY BE GENERATED FROM TIME TO TIME.

SETTING UP THE DEVELOPMENT SYSTEM:

Set up the ST computer per its instructions.

Connect standard 7800 to power supply, attach 7800 to TV set.

Attach standard parallel transfer cable to printer port of ST and to the 25-pin D connector on the development system cartridge.

Put the development system cartridge in the cartridge port (chips to the rear) and power-up the 7800.

Run DLOAD.TTP on the ST and if properly connected, you will see the sign-on message from the development system cartridge on your ST screen (another message is displayed on the TV screen).

And off you go!

SOME INFORMATION THAT WAS NEVER GIVEN IN ANY OF THE OTHER DOCUMENTATION:

How to use the right and left joystick buttons (7800 games only):
Initialize SWCHB with the following 4 instructions:

```
LDA   #$14
STA   CTLSWB
LDA   #0
STA   SWCHB
```

Read the fire buttons from:

```
INPT0   -  player 0, right button (d7=1 if pushed)
INPT1   -  player 0, left button  (d7=1 if pushed)
INPT2   -  player 1, right button (d7=1 if pushed)
```

INPT3 - player 1, left button (d7=1 if pushed)

How to program for the 2600 on the 7800 Development System:

1. A 2600 "lock-in" must be performed before anything else and is accomplished by these 2 instructions:

```
LDA  #$FD
STA  $08
```

2. Six 4k blocks of memory space ("banks" on the cartridge) are available for use by 2600 game developers. They are:

   5000-5FFF
   7000-7FFF
   9000-9FFF
   B000-BFFF
   D000-DFFF
   F000-FFFF

Since 2600 cartridges have 1,2,4, or 8 banks of 4k each, only cartridges of up to 4 banks in size may be developed with this system. 8 bank cartridges will need the dedicated 2600 development system when it comes out. It is recommended that the F000-FFFF bank not be used during game development (except to store hardware vectors and to execute the 2600 lock-in described above) since 122 bytes at FF80-FFF9 are dedicated to 7800 encription signatures & the development system requires this area for system code. F000-FF7F may be used, but it cannot be "packed" full of code as would be possible in the EPROM cartridge. Also, keep in mind that the development system does not emulate bank switching or SARA ram accesses precisely. Since the 7800 development system was not expressly designed to run in 2600 mode, some problems have been observed when it is used in 2600 mode. Most development system boards are tested to see how they perform in 2600 mode. Some work fine but others only allow load & go with no debugger communications once 2600 mode has been locked-in. If the debugger fails to work once the 2600 program has been started (either no communications or ?? responses to legitimate debugger commands), you may have a board or base-unit (or combination thereof) which cannot tolerate 2600 mode. Try a different base-unit or different development system cart.

Summary of differences between dev system as 2600 & 2600 EPROM cartridge:

Development system as 2600:	EPROM cartridge:
1. Code in bank F000-FFFF must not reside at FF80-FFF9	Bank F000-FFFF may be used in its entirety.
2. Bank switching timing can be simulated but other bank switching side-effects cannot	Bank switching must be used for access to other 4k blocks. Bank 0 code cannot access bank 1 data.

be directly observed. For example, code executing from bank 0 can access data from bank 1.	When attempted, this is usually quite evident as a bug.
3. SARA ram accesses can be simulated provided that read & write accesses are in the same 128 byte block.	SARA ram must be read at F080-F0FF and written to at F000-F07F.
4. The two instruction 2600 lock-in code described above must preceed all other code.	The two instruction 2600 lock-in code is not necessary and uses 4 bytes of ROM that could be used otherwise.

Bank switching is used for 2600 cartridges larger than 4K. When it was first designed (in the stone age of 1977), the 2600 base-unit brought only enough address lines out to the cartridge to address 4k. Now that ROM has become so cheap, 2600 Bank switching necessary to address larger carts. This is accomplished by reading a "magic" location. Normally, a LDA ABSOLUTE is executed followed by a JMP ABSOLUTE. A copy of these two instructions is found at the same offset in both the bank being switched to and the bank being switched from. In addition, at power-up, the programmer cannot assume which bank will get control first. All banks must vector RESET thru proper bank switching code to the bank with the start-up code. The magic addresses to be read for bank switching purposes (when applicable) are:

cart size:	1 bank	2 bank	4 bank	8 bank	
	none	$FFF8	$FFF6	$FFF2	lowest bank #
		$FFF9	$FFF7	$FFF3	|
			$FFF8	$FFF4	|
			$FFF9	$FFF5	V
				$FFF6	highest bank #
				$FFF7	
				$FFF8	
				$FFF9	

For more details, see sample source code.

Available PAL Colors: (2600):

0,2,4,5,6,7,A,B,C,D   Lums are the same

Conversions

1-2
2-4
3-6
4-6
5-A
6-C
7-C
8-B
9-D
A-D
B-7
C-5
D-5
E-5
F-4

Odd or even numbered scan lines may alter the colors to black and white. (2600 PAL conversion)

Some previously released 7800 cartridges used additional RAM provided on the cartridge itself. This has become prohibitively expensive and is not allowed for future game development. Exceptions will be made only if cleared in writing by Atari management first.

When sending EPROMS to Atari, the following information should be provided (on a 1/2" x 3/4" label that does not cover up the printing on the EPROM):

GAME NAME (including system - may be abbreviated)
DATE
CHECKSUM
BANK #

For example:	7800 BALLBL.	2600 SOCC
	9/28/87	9/28/87
	E2D8	300B HI
	BANK67	

The preferred form of game milestone submission to Atari is sending the source code on floppy disk together with the .O files generated by that source. A .BAT file containing the command line for DLOAD.TTP would be a convenience as well. Source code is manditory for final submissions before game release.

If there are any questions regarding the use of the software or hardware, call John Feagans at Atari: (408) 745-4923. He and engineer Jose "Beam-Me-Up" Valdes are also available through CompuServe for 24-hour Q&A service (see CompuServe booklet for details).

Dave Staugas may also be reached for questions at (408) 745-2267.

Description of the Atari 7800 "Pro" Development System Upgrade

Document revision date: 13-July-88

New features (vs. previous development system):

- Downloads proceed up to 6 times faster than the previous development system.

- Programs to download need not be converted to .SR (S-records) format via the CONV65.PRG. Object files straight from the assembler (MAC.PRG .O files) may be used directly, saving the CONV65.PRG step and increasing download speeds by a factor of 2. .O files must be used if symbolic references are desired with the debugger.

- An on-board symbolic debugger is included with trace, go with breakpoint, list (disassembly), set (change memory), register change, and dump memory commands currently available.

- Communications between ST and 7800 base unit are accomplished via a new bi-directional parallel I/O port on the development cartridge making the joystick ports free on the base unit for, of all things, joysticks!

- Any production 7800 base unit may be used with the development system "cartridge" (no need to modify the base-unit ROM) since the development system cartridge ROM is encrypted to "pass" the encryption test of the base unit.

- Programs (whether single or multiple bank) may be loaded without use of a Load/Run switch since the develpment system cartridge manages memory automatically during download.

- New board has been designed especially for development system use and will (hopefully) prove more reliable than the previous kludge-board.

- Checksum is computed for S-records after reading data back from RAM rather than simply adding up the data as received and then storing to RAM as was done with the previous system. A similiar method is used with .O files except a 16-bit checksum is used to improve reliability detection.

Items you should have with this new Development System Upgrade:

1. Single board (large) cartridge with parallel port.

2. Parallel ribbon cable.

3. Diskette with DLOAD.TTP program.

Using the Development System Upgrade:

To use the Atari 7800 "Pro" development system, simply plug the supplied 7800 development card into any 7800 base unit and connect the parallel cable between your ST computer's printer port and the 7800 development card. Power-up the base unit and wait about 2 seconds ("ATARI" & Fuji is displayed on TV screen while decryption is performed). The blue screen with the 7800 sign-on message should now be displayed. If your base unit has the old transfer program ROM, you will need to depress the 7800 reset button to start the new debug cartridge ROM.

Now, run DLOAD.TTP on the ST. If an .SR file or .0 file is to be downloaded, type it's name on the command line when invoking DLOAD.TTP. If the file type is omitted, the program will first look for an .0 file on the current directory. If an .0 is not found, an attempt will be made to load an .SR file. Multiple files of either type may be listed on the command line separated by spaces or commas, and they will be loaded in the order they appear there. Only .0 files contain symbols that can be used with the symbolic debugger so this file type is preferrable when debugging is to be performed. The type of download to be performed is determined by the file-type (.0 or .SR) so be sure that the format of each file is identified by its proper .0 or .SR.

For game programs that do not require multiple banks, bank zero will be used automatically and no relocating of S-records is necessary. Multiple bank programs will still require a separate assembly for each bank as with the old development system but no relocating is necessary. If you are loading .0 files (w/symbols), the following two lines of "code" should be included in each source file (bank #3 example is shown):

```
BANKNBR    =    3        ;this equate tells DLOAD.TTP to load in bank #3
DUMMY      =    BANKNBR
```

Due to a quirk of MADMAC (the Atari cross-assembler), the symbol BANKNBR will not be included in the .0 file symbol table unless it is used as well as being defined. So, use it in another dummy equate to force it's inclusion into the symbol table. Another quirk of MADMAC that I have observed; symbols starting with upper-case "L" are not included in the .0 file! Still another bug observed in MADMAC: If an entire page of 6502 memory is initialized to 00's, the assembler omits the page in the .0 file entirely. The symbol BANKNBR (must be all upper-case) should not be used in any other way. If this symbol is not found, bank #0 will be assumed. If you prefer to use S-records, the way to switch banks is via a new S-record type that is used expressly for bank switching. A summary of the three S-record types understood by the development system is as follows (blank spaces are added for clarity and should not be present in actual S-record):

Example	S-record description
S 1 23 8000 78 56 09 ... 7A	S Starting byte of every S-record 1 Record type 1 for download data 23 Byte count in hex (add 3 to include addr & checksum) 8000 Starting load address, this record 78 56 09 etc. is 32 bytes of dload data 7A is the checksum byte when added to the data bytes, 2 address bytes and the byte count byte should equal $FF (ignoring overflo).
S 3 05	S Starting byte of every S-record 3 Record type 3 for switching banks 05 A switch to bank #5 is performed. (no checksum needed)
S 9	S Starting byte of every S-record 9 Record type 9 for terminating this download. (This record is automatically appended after each download)

Once the file(s) have downloaded, DLOAD.TTP becomes a terminal of sorts that allows the programmer to inspect 7800 system RAM/ROM and debug his program. The prompt is a tilde ("~") indicating the debugger is ready to receive commands from the ST keyboard. The commands for this debugger are modeled after those of SID.PRG which comes with the ST development system.

Some of the commands accept addresses or data as arguments. Any such address or data may be expressed in the following 3 ways:

As a hex constant (examples--FCB2 40A AB).

As a hex constant preceeded by a bank number & colon. The bank number (0-7) is only meaningful when applied to addresses in the 8000-BFFF range where bank switching can occur. (examples--0:9A04 5:8000 2:BADD)

As a symbol. A symbol must be preceeded by a period (e.g. ".start") and must be found in the programmers symbol table from the .O file(s) that were downloaded. (examples-- .start .main .loop)

Commands are "sent" when the <cr> is entered and may be editted with backspace before that time. If a command calls for long type-out, the user may suspend/resume the type-out with cntrl-S/cntrl-Q (xon/xoff) sequences. Striking any other key will cancel a long type-out.

Commands currently available:

~g[xxxx][,yyyy]

Go (execute) starting from current PC or at optional xxxx address until optional yyyy breakpoint is reached.

~x[PC | A | X | Y | S | B | P |]

eXamine and change registers command. If the optional register name is omitted, the contents of all registers are displayed. If a register name is given, that register alone is displayed and the user may input a change to the contents of that register. "PC" is a 16 bit value, while "A","X","Y","S", and "P" are 8-bit values. "B" is the current bank number and should be in the range 0-7.

~d[xxxx][,yyyy]

Dump memory starting from last dumped address or optional xxxx address until optional yyyy address.

~l[xxxx][,yyyy]

List (disassemble) memory starting from last listed address or optional xxxx address until optional yyyy address.

~t[w | xxxx]

Trace (execute) one instruction starting from current PC or if optional count xxxx is entered, trace xxxx program instructions. If the instruction to trace is a JSR, ~tw will execute the entire routine called by the JSR and break upon return.

~sxxxx

Set (change) memory command. The address xxxx to set must be provided.

The current contents of the memory address indicated is displayed and the user may enter a new value or skip to the next address with a <cr>. To exit this mode, type a period then <cr>.

```
Example:                                    remarks:

          ~sD000                            change memory starting at D000
          D000 E2 42                        contents E2 changed to 42
          D001 78 <cr>                      contents 78 left alone
          D002 60 20                        contents 60 changed to 20
          D003 CA .<cr>                     exit x command
          ~
```

~z

Shut-up sound command. When breaking into an executing program, the sound latches may be left in an annoying state. The sounds of silence may be had with the z command. Zeros are written to all six sound registers.

~n
~-n

Enable or disable Display List NMI processing while in "system" mode. A minus sign followed by "n" will force any DLI's that occur while the debugger is processing user commands to merely RTI with no further processing. "n" without the minus enables DLI's in system mode but with 71 cycles of overhead added. When executing the users program using the Go command, the state of this flag has no effect--DLI's will execute normally with no added overhead.

Additional notes on debugger use:

If it is desired to break into an executing program on the 7800 development system, the user need merely type cntrl-C at the ST keyboard. An IRQ is generated and the program's state can be examined. To resume, simply restart with the "G" command. The IRQ will not work however if the user's program executes an SEI instruction (set interrupt disable). Change all SEI's to CLI's while debugging, then when it's time to burn EPROM's, change them back to SEI. Since the 7800 target system normally has no external IRQ connected, it probably won't hurt to run your program with IRQ enabled anyway. When a running game has been broken into in this way, the main program has stopped, but DLI's will continue to be processed. However, 71 cycles of overhead is added to the DLI routine which may be unacceptable for some applications. If this is the case, use the -n command to stop display list processing (DLI vector will just point to an RTI). To return to the ST's desktop (or command line) while a downloaded game is running, use the <esc> key. This is the way to exit DLOAD.TTP at any time.

When invoking DLOAD.TTP, the command line may include (in addition to the above described list of files to download) the 1st debugger command to be issued after downloading has finished. In this way, it is possible to "load & go" by automaticaly issuing the "g" command. This initial comand should be preceeded by a minus "-" and should be the last item on the command line. An example of the DLOAD.TTP command line using this feature:

```
DLOAD.TTP file0 file1 file2 -g
```

Occassionally, communications seems to hang up between ST and 7800 development system. This can often be broken thru by use of cntrl-C. As with any alpha release software, bugs and other annoyances will be present in this, the first release of the Atari 7800 Pro Development System Upgrade, which you will probably discover for yourself. I would like to hear about any problems you may be having or requests for features not found here. You may contact me, Dave Staugas, at [408] 745-2267.

Now let's get going and create 7800 games that eat Nintendo alive!

7800 SUPER CART.

A One Megabit ROM in the 7800 super cart will be organized as one fixed 16K x 8 space ($C000 to $FFFF) and seven 16K x 8 banked slots ($8000 to $BFFF). Changing banks is achieved by writing to any location from $8000 to BFFF with the appropriate bank number.

The Bank data should be organized as follows:

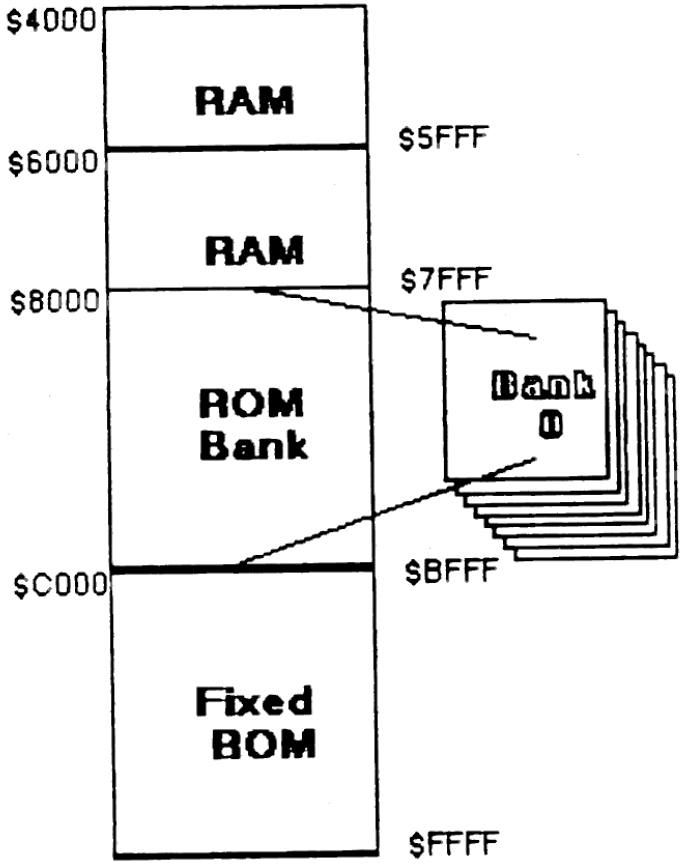
Bank	Data (binary)
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111 (same as fixed bank at $C000-$FFFF)

The 7800 Super Cart. is designed for various sizes of ROMs and will support an 8K x 8 SRAM. Below is a jumper chart showing all supported configurations of the cartridge:

U1 Description	U4 Options	w1	w2	w3	w4	w5	w6	w7	w8	U5	U3	U2
512k ROM/EPROM	Empty	--	++	--	++	--	--	--	--	++	++	++
cs=low,bankswitched	16k/64k SRAM	--	++	--	++	--	--	--	--	++	++	++
	128K ROM	--	++	--	++	--	--	--	--	++	++	++
1024k ROM/EPROM	Empty	--	++	--	--	++	--	--	--	++	++	++
cs=low,bankswitched	16k/64k SRAM	--	++	--	--	++	--	--	--	++	++	++
	128k ROM	--	++	--	--	++	--	--	--	++	++	++
256k ROM	Empty	xx	--	--	αα	αα	++	--	++	--	--	--
cs=high,not banked	16k/64k SRAM	--	--	--	αα	αα	++	--	++	--	++	++
	128k ROM cs0	--	--	--	αα	αα	++	--	++	--	++	++
	128k ROM cs1	--	xx	++	αα	αα	++	--	++	--	--	++
512k ROM (48k net)												
cs=low,not banked	Empty	++	++	--	++	--	--	++	++	--	--	++
256k ROM/EPROM	Empty	--	++	--	++	--	--	--	++	--	++	--
cs=low,not banked	16k/64k SRAM	--	++	--	++	--	--	--	++	--	++	++
	128k ROM cs0	--	++	--	++	--	--	--	++	--	++	++
	128k ROM cs1	not supported										

-- open	++ connected
xx don't care	αα TBD

7800 Cart. Memory Map

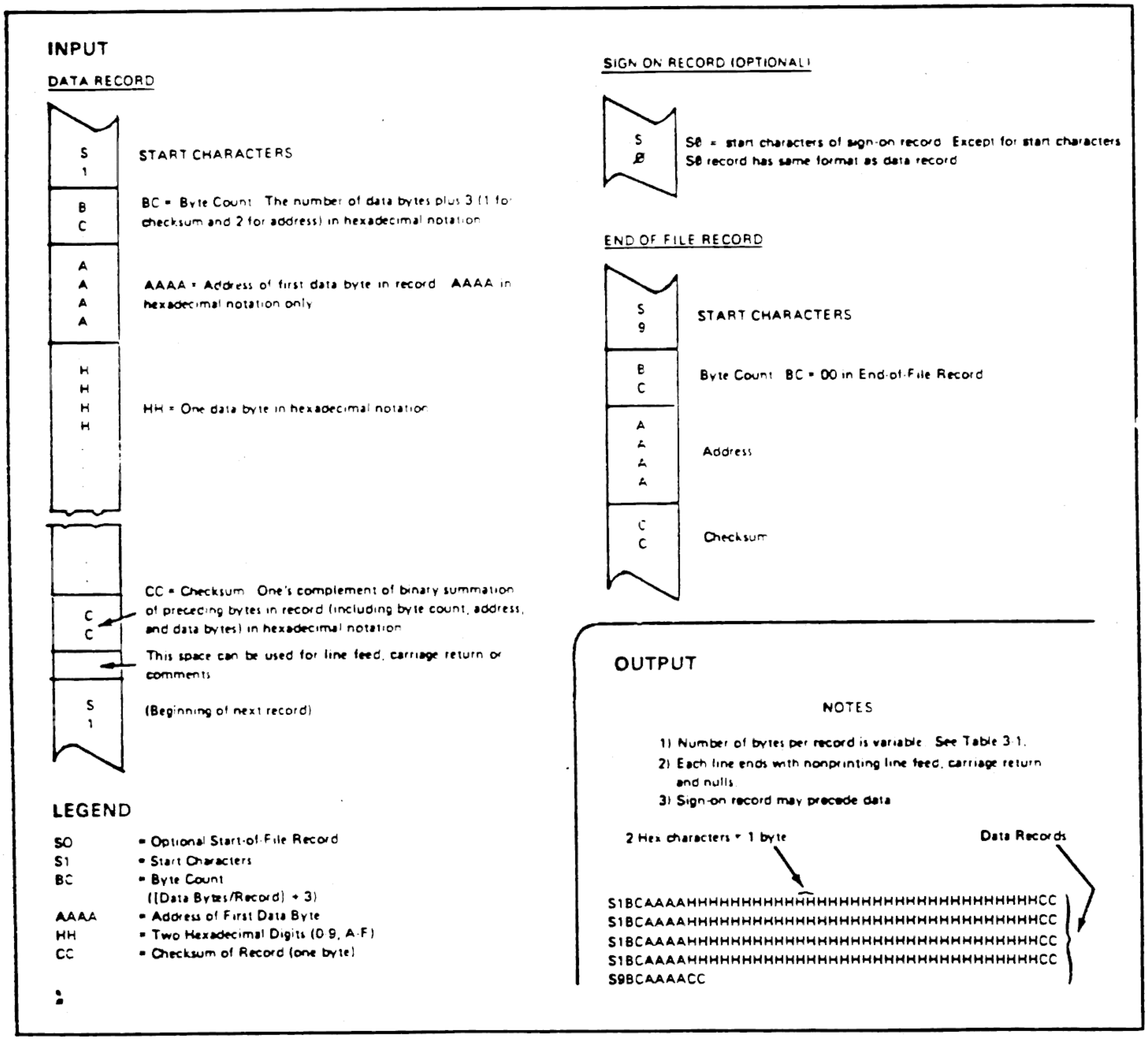


MOTOROLA "S" RECORDS

- **MOTOROLA EXORCISER FORMAT, CODE 82**

Motorola data files may begin with a sign-on record, which is initiated by the code S0. Valid data records start with an 8-character prefix and end with a 2-character suffix. Figure A-11 demonstrates a series of valid Motorola data records.

Each data record begins with the start characters "S1"; the programmer will ignore all earlier characters. The third and fourth characters represent the byte count, which expresses the number of data, address and sum-check bytes in the record. The address of the first data byte in the record is expressed by the last 4 characters of the prefix. Data bytes follow, each represented by 2 hexadecimal characters. The number of data bytes occurring must be 3 less than the byte count. The suffix is a 2-character checksum.



INPUT
DATA RECORD
S 1
START CHARACTERS
B C
BC = Byte Count The number of data bytes plus 3 (1 for checksum and 2 for address) in hexadecimal notation
A A A A
AAAA = Address of first data byte in record AAAA in hexadecimal notation only
H H H H
HH = One data byte in hexadecimal notation
C C
CC = Checksum One's complement of binary summation of preceding bytes in record (including byte count, address, and data bytes) in hexadecimal notation
This space can be used for line feed, carriage return or comments
S 1
(Beginning of next record)
SIGN ON RECORD (OPTIONAL)
S Ø
SØ = start characters of sign-on record Except for start characters SØ record has same format as data record
END OF FILE RECORD
S 9
START CHARACTERS
B C
Byte Count BC = 00 in End-of-File Record
A A A A
Address
C C
Checksum
LEGEND
SO = Optional Start-of-File Record
S1 = Start Characters
BC = Byte Count ([Data Bytes/Record] + 3)
AAAA = Address of First Data Byte
HH = Two Hexadecimal Digits (0-9, A-F)
CC = Checksum of Record (one byte)
OUTPUT
NOTES
1) Number of bytes per record is variable See Table 3-1.
2) Each line ends with nonprinting line feed, carriage return and nulls.
3) Sign-on record may precede data
2 Hex characters = 1 byte
Data Records
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S9BCAAAACC

Figure A-11. Specifications for Motorola Data Files

NOTE FREQUENCIES FOR ATARI 2600/7800 SOUND SYSTEM

	NOTES	04,05	0CH,0DH	01	6,7,9,0AH	0FH	02,03	08
	C 10							
	B 9	$00_(-104)						
	B flat 9							
	A 9 15080 CPS							
	G sharp 9							
	G 9							
9	F sharp 9							
	F 9							
	E 9							
	E flat 9							
	D 9							
	C sharp 9							
	C 9							
	B 8	$01_(-052)						
	B flat 8							
	A 8 7040 CPS							
	G sharp 8							
	G 8							
8	F sharp 8							
	F 8							
	E 8	$02_(-041)	$00_(-040)					
	E flat 8							
	D 8							
	C sharp 8							
	C 8							
	B 7	$03_(-026)						
	B flat 7							
	A 7 3520 CPS							
	G sharp 7							
	G 7	$04_(-004)						
7	F sharp 7							
	F 7							
	E 7	$05_(-010)	$01_(-020)					
	E flat 7							
	D 7							
	C sharp 7	$06_(+025)						
	C 7			$00_(0000)				
	B 6	$07_(-013)						
	B flat 6							
	A 6 1760 CPS	$08_(-015)	$02_(-016)					
	G sharp 6							
	G 6	$09_(-002)						
6	F sharp 6	$0A_(-053)						
	F 6	$0A_(+031)						
	E 6	$0B_(-010)	$03_(-010)					
	E flat 6	$0C_(-037)						
	D 6	$0C_(+033)						
	C sharp 6	$0D_(+013)	$04_(0000)					
	C 6	$0E_(0000)		$01_(0000)				
	B 5	$0F_(-006)			$00_(-025)			
	B flat 5	$10_(-009)						
	A 5 880 CPS	$11_(-008)	$05_(-008)					
	G sharp 5	$12_(-004)						
	G 5	$13_(+001)						
5	F sharp 5	$14_(+008)	$06_(+007)					
	F 5 $15_(+015)	$16_(-016)		$02_(-001)				
	E 5	$17 _(-005)	$07_(-005)					
	E flat 5	$18_(+006)						
	D 5	$1A(-006)	$08_(-006)					
	C sharp 5	$1B_(+006)						
	C 5	$1D_(0000)	$09_(0000)	$03_(0000)				

	NOTES	04,05	0CH,0DH	01	6,7,9,0AH	0FH	02,03	08
	B 4	$1F_(-035)			$01_(+013)			
	B flat 4		$0A_(+009)					
	A 4 440 CPS		$0B_(-004)					
	G sharp 4			$04_(+003)				
	G 4		$0C_(+010)					
4	F sharp 4		$0D_(+003)					
	F 4		$0E_(-001)	$05_(-001)				
	E 4		$0F_(-002)		$02_(+008)	$00_(+008)		
	E flat 4		$10_(-004)					
	D 4		$11_(-003)	$06_(+006)				
	C sharp 4		$12_(-002)					
	C 4		$13_(-000)	$07_(0000)				
	B 3		$14_(+003)		$03_(+007)			
	B flat 3		$15_(+004)	$08_(-001)				
	A 3 220 CPS		$17_(-002)					
	G sharp 3		$18_(+002)	$09_(+002)	$04_(-005)			
3	G 3		$1A_(-003)					
	F sharp 3		$1B_(+001)	$0A_(+005)				
	F 3		$1D_(0000)	$0B_(0000)				
	E 3		$1F_(-001)	$0C_(-003)	$05_(+004)	$01_(+004)		
	E flat 3							
	D 3			$0D_(+003)	$06_(-002)			
	C sharp 3			$0E_(+001)				
	C 3			$0F_(0000)				
	B 2			$10_(0000)	$07_(+003)			
	B flat 2			$11_(0000)				
	A 2 110 CPS			$12_(0000)	$08_(+002)	$02_(+002)		
	G sharp 2			$13_(+001)	$09_(-002)			
	G 2			$14_(+001)				
2	F sharp 2			$16_(-001)	$0A_(0000)			
	F 2			$17_(0000)				
	E 2			$18_(+001)	$0B_(+002)	$03_(+002)		
	E flat 2			$1A_(0000)	$0C_(0000)			
	D 2			$1B_(+001)	$0D_(-001)			
	C sharp 2			$1D_(0000)	$0E_(-002)	$04_(-002)		
	C 2			$1F_(0000)	$0E_(+002)	$04_(+002)	$00_(+002)	$00_(-004)
	B 1				$0F_(+002)			
	B flat 1				$10_(+001)			
	A 1 055 CPS				$11_(+001)	$05_(+001)		
	G sharp 1				$13_(-001)			
	G 1				$14_(-001)	$06_(-001)		
1	F sharp 1				$15_(0000)			
	F 1				$16_(+001)	$07_(-001)		
	E 1				$17_(+001)	$07_(+001)		
	E flat 1				$19_(0000)	$08_(-001)		
	D 1				$1B_(0000)	$08_(+001)		
	C sharp 1				$1C_(0000)	$09_(-001)		
	C 1				$1E_(0000)	$09_(+001)	$01_(+002)	$01_(-002)
	B 0							
	B flat 0							
	A 0 027 CPS							
	G sharp 0							
	G 0							
0	F sharp 0							
	F 0							
	E 0							
	E flat 0							
	D 0							
	C sharp 0							
	C 0							