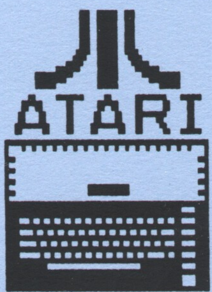
THE COMPLETE AND ESSENTIAL MAP FOR THE XL / XE



Written by
ANDREW C. THOMPSON ©1994

Published and Distributed by
TWAUG PUBLISHING™ ©1994

PART II

APPENDIX A

APPENDIX A1:

BASIC KEYWORDS.
Here's a list of all the Basic keywords in alphabetical order, along with their abbreviations, format and a short explanation:

KEYWORD:	ABBREV:	FORMAT:

ABS P=ABS(Q)
Returns the absolute value of a given value V, which in other words is; remove the minus sign. Where Q is the variable of unknown sign, and P becomes the positive Q.

ADR P=ADR(U$)
Returns the ADRress of the given string U$. Where P becomes the location in memory where the data inside U$ begins.

AND X=P AND Q
Boolean expression. Where X=1 if P AND Q are both positive, 0 otherwise.

ASC X=ASC("U") or X=ASC(U$(1,1))
Where X becomes the ASCII code of the letter U, or of the letter contained in the 1st element of U$.

ATN P=ATN(V)
P becomes the angle in RADians or DEGrees, whose ArcTaNgent is V.

BYE B. BYE
Exits from Basic to the Self-test mode.

CHR$ U$=CHR$(V)
The 1st element of U$ contains the CHaRacter of the ASCII code in V. Reverse of the ASC function.

CLOSE # CL. CLOSE #X
CLOSEs the IOCB OPENed on channel #X.

CLOAD CLOAD
Loads a program that was previously CSAVEd onto cassette. This is standard type of saved file.

CLOG P=CLOG(V)
P becomes the base-10 LOGarithm of the value V.

CLR CLR
CLeaR all string and variable memory reserved with DIM, COM and LET.

COLOR C. COLOR X
Selects X colour register to be used in the next PLOT and DRAWTO. The colour register contains the actual colour to use. See pages 55 - 59.

APPENDIX A1:

COM   COM U$(X) or COM H(L)
Exactly the same as DIM, although this can cause a few unusual bugs.

CONT   CONT
Restarts a program where it exited, either due to the STOP command, the BReaK key or an error. The line number is retained in locations 186 and 187 in LSB/MSB format.

COS   P=COS(X)
P contains the COSine of angle X, where X can be in RADians or DEGrees.

CSAVE   CS.   CSAVE
Saves a program to cassette which can later be CLOADed. This is the standard type of cassette save.

DATA   D.   DATA flint,stone
Marks a list of string or numeric data that can laterwards be READ for miscellaneous use. Each element in a DATA statement must be seperated by a comma.

DEG   DEG
Select DEGrees mode for use with all trigonometric operations.

DIM   DIM U$(X) or DIM H(L)
Allocates X amount of bytes/elements in memory accessible by U$, or allocates L amount of cells identified by H, where 1 cell is 6 bytes.

DOS   DOS
Loads the DOS menu from the disk if it contains the DUP.SYS file, but if DOS is not loaded, then calling DOS has the same function as BYE. DOS jumps through the vector at locations 10 and 11.

DRAWTO   DR.   DRAWTO X,Y
Draws a line from the point of the cursor to the new point using the colour register given by COLOR. Co-ordinate 0,0 is top-left and X is horizontal, while Y is vertical. See PLOT.

END   END
ENDs a program.

ENTER   E.   ENTER"D:TOMMY"
Enters the program TOMMY from the disk unit which was previously saved with LIST. You can also substitute the D with any of E, S, C and K. Though, with K you'll have to press CTRL and 3 to escape the mode.

EXP   D=EXP(P)
D returns the EXPonential of P, where the EXP of a number is the power.

APPENDIX A1:

FOR F. FOR J=S TO F STEP S
Initiates the program loop J, from S TO F in steps of S. STEP S is optional, default STEP is 1. See NEXT.

FRE ? FRE(0)
Returns the amount of free/unused memory available for use.

GET GET #C,X
X contains the ASCII code of the next character read from a file opened on channel C. If the channel is opened on the keyboard, then a keypress is awaited, and X returns the last key pressed.

GOSUB GOS. GOSUB 1000
Branch to a sub-routine in a Basic program placing the address of the GOSUB line on the Basic runtime stack. See RETURN.

GOTO G. GOTO N
Jump to the line whose value is in N.

GRAPHICS GR. GRAPHICS M
Selects GRAPHICS mode M. There are 16 modes to choose from, but M+16 removes the text-window, and M+32 accesses the mode without clearing the memory used by the mode being called.

IF/THEN IF (condition) THEN (action)
IF the condition is true, THEN the action will be carried out. Both of which may be a series of keywords or expressions.

INPUT I. INPUT V or INPUT #X,U$
Awaits keyboard INPUT ended by pressing Return, or brings in a particular amount of bytes (until it finds an EOL; a Return character; 155 $9B) from the device OPENed on channel X, into U$.

INT R=INT(H)
R returns the INTeger part of a fractional/real number stored in H.

LEN L=LEN(U$)
L returns the element length of the string U$.

LET LET V=9
An optional keyword; where it assigns the value 9 to the numeric variable V.

LIST L. LIST or LIST "D:BILL",S,F
LISTs the present listing to the screen, or LISTs lines S to F of the listing to a file called BILL on the disk-drive. S and F are optional. This kind of storage is different to that of SAVE and CSAVE.

LOAD LO. LOAD"D:TED"
LOADs the file TED from device D, which is the disk-drive in this case.

APPENDIX A1:

LOCATE LOC. LOCATE X,Y,Z
Z returns the colour register value stored at screen co-ordinates X,Y.

LOG B=LOG(J)
B returns the natural LOGarithm of the number in J.

LPRINT LP. LPRINT "elpasso"
Outputs the data 'elpasso' to the printer.

NEW NEW
Clears the program and all variables from memory, CLOSEs all IOCB channels except 0, turns all voices off and selects RADians mode.

NEXT N. NEXT K
Marks the end of the FOR/NEXT loop K, whilst also means that the loop must execute all lines between the FOR and NEXT keywords until K supersedes F. See FOR for the F variable.

NOT I=NOT U
Boolean expression. I returns the reverse sign of the number in U. See SGN, also AND.

NOTE NO. NOTE #C,S,B
Returns the last byte number B accessed within the last Sector S accessed in the file OPENed on channel C.

ON ON X GOTO P,Q,R,S
Depending on the value in X, then control will GOTO P, Q, R or S. GOTO P if X=1, or Q if X=2 etc.. GOTO can also be substituted with GOSUB, and the list of destinations can be endless (almost).

OPEN O. OPEN #G,R,S,"D:ROCKY"
OPEN the file 'ROCKY' from the disk-device using channel C. The R variable offers you the type of file access, where 4 is read and 8 is write. While S is an additional variable only used for particular operations (given by R). See the table on page 96.

OR A=U OR I
Boolean expression. A returns a 1 if either of the U or I variables are positive. Returns a 0 only when both U and I are zero or negative.

PADDLE H=PADDLE(P)
H returns the current position of the paddle controller in port P.

PEEK S=PEEK(T)
S returns the contents of memory location T.

PLOT PL. PLOT X,Y
PLOT the present colour at screen co-ordinates X,Y. See COLOR and DRAWTO.

APPENDIX A1:

POINT PO. POINT #C,S,B
Addresses an internal POINTer to the sector and byte within that sector given by variables S and B in the file opened on channel C.

POKE POKE J,R
Replaces the old value in memory location J with the value in R, where R is a number between 0 - 255.

POP POP
Used with GOSUB and FOR/NEXT loops. This removes the last return address placed on the stack, which is normally used to tell Basic what line number to return to when finding either NEXT or RETURN.

POSITION POS. POSITION X,Y
POSITIONs the cursor at the X,Y co-ordinate, ready for subsequent PRINTing.

PRINT PR./? PRINT "pizza" or PRINT #Y;U$
Places the string 'pizza' on screen beginning at the cursor' co-ordinates given by POSITION, or outputs data contained in U$ to the file OPENed on channel Y. A "," can be used instead of the semi-colon ";" to include a tab in the file.

PTRIG W=PTRIG(P)
W returns the status of the paddle trigger P. 0 is pressed.

PUT PUT #L,G
Outputs the byte stored in G to the presently recorded position of the file OPENed on channel L. See NOTE and POINT.

RAD RAD
Selects RADians mode.

READ READ Y or READ U$
READs the present element in a DATA line into Y or U$ depending on the type of element, whether it be numeric or string. See location 182.

REM . REM this-does-that
REMarks in a program listing so that you can REMember what it is supposed to do, which is good when you go back to a listing you may have wrote several years before.

RESTORE RES. RESTORE FREEDOM
Re-addresses a new DATA line given by the variable 'FREEDOM', it also resets the element-number being read in a line to the 1st one. See locations 182, 183 and 184. Also see READ and DATA.

RETURN RET. RETURN
RETURNS from the subroutine to where the GOSUB jumped to by re-instating its address from the top of the stack. See POP also.

RND N=RND(0) or V=INT(RND(0)*W)
N returns a RaNDom number between 0 and .9, while V returns an INTeger number between 0 and W-1.

RUN RUN
Executes a Basic program, clearing all variables and CLOSEing OPENed channels in doing so.

SAVE S. SAVE "D:MONEY"
SAVEs the current program onto the disk-drive under the file-name 'MONEY'. This is also the standard type of disk-file. See LOAD.

SETCOLOR SE. SETCOLOR R,C,L
Sets the colour register R to the colour C and luminance L. See COLOR.

SGN B=SGN(V)
B returns the SiGN of the number V. When V is positive, B returns 1, if V is 0 then B = 0, but if V is negative, then B returns -1.

SIN W=SIN(G)
W returns the SINe of the number G in RADians or DEGrees.

SOUND SO. SOUND C,P,D,V
The sound made is at Pitch P, Volume V using the D distortion in channel C. There are 4 channels, pitch is between 0 - 255, distortion has many variations where a value of 10 is pure tone, and the volume has 16 levels.

SQR V=SQR(T)
V returns the SQuaRe root of the number T.

STATUS ST. STATUS #C,E
E returns the STATUS of the most recent I/O operation on channel C.

STEP See FOR
Optional parameter of the FOR/NEXT loop which specifies the STEP increment of the loop. Default is +1. STEP must be used to perform decrementing loops.

STICK Q=STICK(K)
Q returns the present position of the joystick in port K, where K is 0 or 1. See locations 632 and 633.

STOP STOP
STOP the execution of a program. The line at which it stops can also be continued with CONT.

STR$ H$=STR$(V)
The value V is transfered into the string H$.

STRIG W=STRIG(U)
W returns the status of the joystick trigger in port U, where 1 means released and 0 is pressed.

APPENDIX A1:

THEN See IF
Used with IF.

TO See FOR
Used with FOR/NEXT loops.

TRAP T. TRAP M
Upon the occurence of an error, program control will be passed to the line number given by M. Type TRAP 40000 to turn the TRAP mode off.

USR X=USR(joe,L,M,N) or X=USR(ADR(U$),L,M,N)
Passes control of the Basic program to the machine-language routine beginning at the address given by 'joe'. Parameters L, M and N are optional (including the amount of parameters) and are passed through onto the stack (see locations 256 - 511) in a particular way: firstly, the current location of the Basic program is passed onto the stack, followed by all (if any) of the parameters values in LSB/MSB format (the LSB preceeds the MSB when being pushed on). Once this is done, a single byte is then pushed on top of the stack to represent the amount of parameters passed to the machine-code routine. If there were no parameters passed, then only the Basic return address (2 bytes) and the amount of variables passed (0) would be stacked. Upon return from the routine, you should ensure that the 2 bytes at the top of the stack is the Basic return address. The return instruction is RTS; 96 $60 and not RTI.
The program on page 78 shows USR passing a variable to a machine-code routine.

VAL J=VAL(G$(B))
The reverse of the STR$ function, J becomes the string of number-digits (the value) found, beginning at the Bth element in G$.

XIO XIO F,#Y,I,J,"D:SUE"
A very powerful command that Covers a wide variety of operations which don't utilize a seperate Basic keyword. XIO can perform most of the DOS menu functions, DRAWTO and screen FILLing. See the COMMAND table on page-95. XIO can also be used to create new commands that Basic don't support. A good use for XIO would be to write a new handler device that gives Player/missile graphic commands, such as clearing memory or vertical movement.

APPENDIX A2:

BASIC TOKENIZATION.
A very hidden subject is the tokenization of Basic programs. Probably the best explanation is De Re Atari' one, which is very in depth. Here's a coverage of De Re Atari's explanation.

The visual image of the typical Basic program is quite different to the Basic ROM. To us it appears as 'BASIC', but to the language it is processed as a TOKENIZED program, where each Basic command is recognized as a unique character (a token). When a line of Basic is entered, the language tokenizes your input, checking for legal syntax as it goes. Should this tokenized basic line be without a line number, then it will be executed straight away, but if it has one then it is included into the 'tokenized program'.

The TOKENIZING process converts a line number into a 2-byte (LSB/MSB) integer. If the line is in immediate mode (no line number), then before executing it, a line number of 32768; $8000 is assigned to it. The next token is a byte-count, from the beginning of the line being tokenized to the start of the next line. Obviously, this byte has to be filled-in last of all. After this, Basic then searches through its list of legal commands for the correct token equivalent of the command. If it doesn't find the command, then it is unknown, thus, a syntax error is returned. Ofcourse, with all going fine the next item to be tokenized can be any of 7 different things: a variable, constant, operator, function, double-quote, another statement or just End-Of-Line EOL.

Basic tests to see if the next inputted character is numeric. If not then it compares that character and those following it with the entries from the variable name table (VNT). If this is the 1st line of code entered in the program, then no match will be found. The characters are then compared with the function and operator tables, then should no match be found again, the characters are accepted as a new variable name. All variable names in the variable name table always have the last byte inversed (bit-7 set) to indicate the end of the name. This variable name then has its token (variable number in the table) put into the tokenized line. Note that the variable number token has bit-7 set and is also subtracted by 1, thus the 1st variable token number would be 128; $80 and so on. Should a match be found as a function or an operator, then its token will be placed in the tokenized line.

Double-quotes are tokened with the number 15; $0F, and a byte-count of string characters is included. The actual characters are moved from the input buffer to the output buffer until either the 2nd pair of quotes, or EOL.

APPENDIX A2:

If the next characters in the input buffer are numeric, then they are converted into a 6-byte BCD constant. The token going in the tokenized line becomes 14; $0E and the 6-byte BCD constant follows it. When (if) a colon (:) is encountered, a token of 20; $14 is put in the output buffer and the offset from the beginning of the 1st statement to the start of the 2nd is completed, another byte-count character is set aside and the process is repeated by searching for a command. Eventually, the EOL character will be found whereby a token of 22; $16 is included and the last byte-count character is filled-in. This now completes 1 line where it is then copied into the token program, replacing any line of the same value. All numeric line order is correctly re-organized, thus, contracting/expanding the program as necessary.

If the line was immediate, then this is where it executes it. All immediate mode lines have the number 32768; $8000, so as you can see they overwrite each other every time. The maximum length of the input line is normally indicated by the famous bleep, but this is not always true, since the maximum length is no more than 256 'tokenized' bytes. Here's an example of a tokenized line:

```
10 LET X=1: PRINT X

0A 00 13 0F 06 80 2D 0E 40 01 00 00 00 00 14 13 20 80 16


0A 00               Line 10
13                  Line offset (byte-count)
0F                  Statement offset
06                  The LET token
80                  Variable X (the 1st and only)
2D        =
0E                  Numeric constant
40 01 00 00 00 00   The number 1
14                  End of statement
13                  Statement offset
20                  The PRINT token
80                  Variable X
16                  End Of Line (EOL)
```

Following on the next couple of pages is the entire token list for Atari Basic and Turbo Basic.

APPENDIX A2:

HX	DC	COMMAND:	OPERATOR:
00	0	REM	
01	1	DATA	
02	2	INPUT	
03	3	COLOR	
04	4	LIST	
05	5	ENTER	
06	6	LET	
07	7	IF	
08	8	FOR	
09	9	NEXT	
0A	10	GOTO	
0B	11	GO TO	
0C	12	GOSUB	
0D	13	TRAP	
0E	14	BYE	[NUM CONSTANT]
0F	15	CONT	[STR "]
10	16	COM	[UNUSED]
11	17	CLOSE	"
12	18	CLR	,
13	19	DEG	$
14	20	DIM	: [STMT END]
15	21	END	;
16	22	NEW	[LINE END]
17	23	OPEN	GOTO
18	24	LOAD	GOSUB
19	25	SAVE	TO
1A	26	STATUS	STEP
1B	27	NOTE	THEN
1C	28	POINT	#
1D	29	XIO	[=
1E	30	ON	[]
1F	31	POKE	]=
20	32	PRINT	[
21	33	RAD	]
22	34	READ	=
23	35	RESTORE	SPACING
24	36	RETURN	*
25	37	RUN	+
26	38	STOP	-
27	39	POP	/
28	40	?	NOT
29	41	GET	OR
2A	42	PUT	AND
2B	43	GRAPHICS	(
2C	44	PLOT	)
2D	45	POSITION	= [ARITH ASSIGN]
2E	46	DOS	= [STRING ASSIGN]
2F	47	DRAWTO	[= [STRINGS]
30	48	SETCOLOR	[]
31	49	LOCATE	]=
32	50	SOUND	[
33	51	LPRINT	]

APPENDIX A2:

34	52	CSAVE
35	53	CLOAD
36	54	IMPLIED LET
37	55	SYNTAX ERR.
38	56	DPOKE
39	57	MOVE
3A	58	-MOVE
3B	59	*F
3C	60	REPEAT
3D	61	UNTIL
3E	62	WHILE
3F	63	WEND
40	64	ELSE
41	65	ENDIF
42	66	BPUT
43	67	BGET
44	68	FILLTO
45	69	DO
46	70	LOOP
47	71	EXIT
48	72	DIR
49	73	LOCK
4A	74	UNLOCK
4B	75	RENAME
4C	76	DELETE
4D	77	PAUSE
4E	78	TIME$=
4F	79	PROC
50	80	EXEC
51	81	ENDPROC
52	82	FCOLOR
53	83	*L
54	84	--
55	85	RENUM
56	86	DEL
57	87	DUMP
58	88	TRACE
59	89	TEXT
5A	90	BLOAD
5B	91	BRUN
5C	92	GO#
5D	93	#
5E	94	*B
5F	95	PAINT
60	96	CLS
61	97	DSOUND
62	98	CIRCLE
63	99	%PUT
64	100	%GET

=	
+	[UNARY]
-	
(	[LEFT STRING PAREN]
(	[" ARRAY "]
(	[DIM ARRAY LEFT PAREN]
(	[FUN LEFT PAREN]
(	[DIM STR LEFT PAREN]
,	[ARRAY]

HX	DC	FUNCTION:
3D	61	STR$
3E	62	CHR$
3F	63	USR
40	64	ASC
41	65	VAL
42	66	LEN
43	67	ADR
44	68	ATN
45	69	COS
46	70	PEEK
47	71	SIN
48	72	RND
49	73	FRE
4A	74	EXP
4B	75	LOG
4C	76	CLOG
4D	77	SQR
4E	78	SGN
4F	79	ABS
50	80	INT
51	81	PADDLE
52	82	STICK
53	83	PTRIG
54	84	STRIG
55	85	DPEEK
56	86	&
57	87	!
58	88	INSTR
59	89	INKEY$
5A	90	EXOR
5B	91	HEX$
5C	92	DEC
5D	93	DIV
5E	94	FRAC
5F	95	TIME$
60	96	TIME
61	97	MOD
62	98	EXEC
63	99	RND
64	100	RAND
65	101	TRUNC
66	102	%0
67	103	%1
68	104	%2
69	105	%3
6A	106	GO#
6B	107	UINSTR
6C	108	ERR
6D	109	ERL

APPENDIX A3:

BASIC ALTERATIONS

Well, this appendice is the last but one for completion of the book. It's not going to be a very descriptive one because I don't feel like it (I don't have the IQ anyway!). Sorry, folks. Perhaps if I make a second book, then this will be a subject for further investigation, but until then... Here's a small coverage in the field of altering the Basic language.

Altering Basic is really useful. Of course, you'll need to checkout location 54017; D301 in order to turn the ROM language into a RAM one. From there you can accomplish many tasks. Indeed, if you don't want to change anything, but rather you want to add functions to it, then this is also feasible. There are several methods to go about this, but perhaps the easiest and quickest is by use of the OS handler system. See HATABS, locations 794 - 828; $31A - $33C. For additional reference then seek out issues 37, 41, 43, 53, 57 and 64 of Page-6 magazine, or as it has recently become to be known; NEW ATARI USER.

Altering Basic has been a topic of concern for quite a long time, where one computer owner might have had previous experience on a different computer, and is now missing the use of some special command that he/she used a lot before. The magazine issues mentioned above contain good example programs that add various commands into the Basic language, indeed, as you learn more about the Atari XL/XE you will find that it is quite capable of adding virtually any command to your Basic language. Of course, in time you may move onto machine-language or perhaps move over to a newly developed one, such like the QUICK language. Anyway, getting back to the main subject... Perhaps, the very easiest and quickest method to achieve extra Basic commands is by use of a Basic sub-routine, where the sub-routine is the command! Another method is to use machine-code routines that can be called via the USR command. This method is used very often. Another method is to catch the keyboard input before it goes to the Basic interpreter, but that method is not detailed here coz I'm not brainy enough, maybe some other time. Anyway, the only other method is the one mentioned earlier. It is also perhaps the most popular one. By writing a new handler, we can perform new Basic processes with any of these standard Basic commands: OPEN, CLOSE, GET, PUT, STATUS, XIO, ENTER, INPUT, LIST, LOAD, NOTE, POINT, PRINT, RUN, and SAVE. In some cases, you can even use: CLOAD, CSAVE and LPRINT.

Again, see locations 794 - 828 for information on this. It's not that difficult once you get the hang of it. Of course, if you do write any commands then send them on, as I know I'd be interested. Good luck.

APPENDIX A4:

PROGRAM IMPROVEMENT.

Improving your Basic programs is really a task you should only perform on a copy of your finished Basic program to enhance its speed and also to reduce its memory requirements. There are many ways of going about this, here are 2 lists of best affectiveness, the 1st for speed and the 2nd for memory reduction.

Varying methods of program speed improvements:

1. If you've been editing/adding to the Basic program, then it will be worthwhile re-coding it.

2. Try to simplify the programs calculations, perhaps even convert them to boolean ones where possible. This includes IF/THEN statements. Enormous time can be saved.

3. Place your most frequently used GOSUB routines & FOR/NEXT loops in lower line numbers, since Basic searches your program beginning at line 0.

4. For frequently called routines nested in loops, try to put the routines in the main program since Basic wastes time adding/removing entries from the run time stack.

5. Make the most often changing loop from a nested set the deepest, this way the run time stack will be altered the fewest amount of times.

6. Simplify floating point calculations within a loop. If a result is found by multiplying a constant by a counter, then time can be saved by changing the operation to an add of a constant.

7. Setup multiple loops on the same line, this way Basic won't have to get the next line to continue the loop.

8. Approx. 30% of processing time can be saved by disabling the screen during operations not requiring the screen.

9. If screen display is needed, then substitute a faster mode (see CYCLE-STEALING appendice) or shorten the DL.

10. Use machine-code. Time is saved by using M/C to perform the loops in a Basic program, via the USR command.

APPENDIX A4:

Varying methods of saving memory in your Basic programs:

1. Again, re-code the listing. Speed and RAM are gained.

2. Remove your REMarks, they occupy essential space.

3. Replace constants used more than twice with variables. Doing this saves 6 bytes every time the variable is used.

4. Load variables with the READ statement from DATA rather than directly, since this saves 6 bytes each time.

5. Again, avoid direct values. Use variables of varied values to achieve other values, adding them etc.. This also applies to line-numbers used in subroutines.

6. Try to minimize the amount of variables your using. Each variable takes 8 bytes in the VVT plus bytes in VNT.

7. Clean up the variable value and name tables by LISTing the program to disk, typing NEW and re-ENTERing it. You should do this coz old variables ain't deleted from the table.

8. Keep variable names as short as can be, 1 char. is 1 byte.

9. Replace common text with strings that hold this text.

10. Initialize strings in direct assignment, it requires less space than the READ method.

11. Condense multiple lines on single lines where possible. 3 bytes are saved each time you do this to 2 lines.

12. Replace once used routines with in-line code, the GOSUB and RETURN waste unnecessary bytes when not needed.

13. Replace numeric arrays with strings if the data values do not exceed 255, since these values can be stored as a single character. For each character, 5 bytes are saved.

14. Replace SETCOLOR statements with POKE commands, this saves 8 bytes each time.

15. Replace POSITION statements with control characters within PRINTS (?'s). 14 bytes are saved each time on average.

16. Modify the string/array pointer to load predefined data, changing STARP this way saves string/array memory.

17. Delete code in program control, see the IOCBs in the MAP.

APPENDIX A5:

TURBO BASIC.
In addition to the normal Basic language, Turbo Basic supports many modifications and new keywords. I've listed them here in this appendice.

BLOAD   BLOAD "D:CHARLIE"
Loads file named CHARLIE without running it. Same as DOS option L with /N appended to filename.

BPUT   BPUT #C,A,L
Block output on channel C. A is the start address and L is the length. Same as FOR Q=A TO A+L:PUT #C,PEEK(Q):NEXT Q.

BRUN   BRUN "D:CHAPLIN"
Same as BLOAD except that the file CHAPLIN is loaded and run.

CIRCLE   CIRCLE X,Y,H,V
Draws a circle, whose center is X,Y. H and V are horizontal and vertical radius. V is optional, not being present H becomes the radious.

CLOSE   CLOSE
A nice modification which when used as shown turns all IOCB channels off.

CLS   CLS #P
Clears the screen. The #P channel is optional, normal mode-0 screen is default.

DEC   T=DEC(N$)
T returns the decimal equivalent of the hexadecimal number in N$.

DEL   DEL S,G
A long desired addition to Basic editing is this, where lines from S to G are deleted.

DELETE   DELETE "D2:OSCAR"
1 of many DOS functions from Basic. This deletes the file OSCAR on drive 2. The normal Basic equivalent is an XIO command.

DIM   DIM X(Z)
Same as normal DIM, although Turbo Basic now clears arrays and strings. The LEN command still returns the correct status of 0 when strings are of no length.

APPENDIX A5:

DIR DIR "D1:GOLDMAN"
Display the disk directory, the parameter string is not necessary. The default is "D1:*.*".

DIV H=C DIV E
H returns the integer quotient for C/E.

DO DO
The initial part of a DO/LOOP structure. Structured programming was created to eradicate the reference of line-numbers within a Basic program. It clarifies a listing considerably. This loop is what is known as a dead-loop, it has no end, although you can EXIT from the loop.

DPEEK DPEEK(Q)
This is an excellant feature where you can perform DL=PEEK(560)+256*PEEK(561) and Q=PEEK(DL) directly with DPEEK(560).

DPOKE DPOKE M,V
This is the opposite of DPEEK. Try DPOKE M,58368. The value DPOKEd is converted to LSB and MSB and put into locations M and M+1.

DSOUND DSOUND N,F,D,V
Another excellant feature that brings more power of the POKEY chip to the Basic user. The POKEY chip offers an ability to pair 2 channels together to achieve a much higher range of frequencies. The channels that can be paired are 0 with 1 and/or 2 with 3.

DUMP DUMP "D1:CHUMP"
A very useful editing command that DUMPs all the variables of a program to the screen (as default) or to the file CHUMP on drive 1.

ELSE IF A THEN W ELSE Q
A splended inclusion that allows the ability to nest multiple conditions in 1 IF/THEN statement. This ability also offers the ability to prevent control-flow going to the next line. In addition, you can restructure your IF/THEN loop like so:

```
IF condition
  reaction
ENDIF
```

ENDIF ENDIF
As shown above, it is used when changing the structure of your IF/THEN loop. This is used if you want more actions than would fit onto a normal program line.

ENDPROC ENDPROC
Last part of the PROC/ENDPROC loop. This is basically a GOSUB/RETURN routine or procedure. See EXEC.

APPENDIX A5:

ERL I=ERL
Better than DPEEKing locations 186 and 187 for the line number where the program stopped due to BREAK or an error, you can checkout this Basic variable. I returns the l'ine number.

ERR S=ERR
S returns the error code.

EXEC EXEC O
This is the GOSUB equivalent for a PROC/ENDPROC structure.

EXIT EXIT
This is the only way out of a DO/LOOP structure. The continue line is immediately after the location of the LOOP statement.

EXOR I=T EXOR N
I returns the EXclusive-OR result of T and N.

FCOLOR FCOLOR A
As COLOR assigns the selected colour register for PLOT and DRAWTO, FCOLOR selects the colour register (A) for the FILLTO command.

FILLTO FILLTO X,Y
This is the XIO 18 fill command.

FRAC S=FRAC(X)
S returns the FRACtional part of X.

GET GET L
In addition to normal Basic, this method of use now checks the keyboard for a single keypress. L returns the ASCII value of the last key pressed.

%GET %GET #C,S
S returns the number accessed from the device open on channel C. This is a special value, put to the device with the %PUT command. The number is actually written in its true 6-byte FP format and not as a character.

GO# GO# AWAY
Similar to GOTO, but addresses the line-name AWAY addressed with #, NOT the line-numbers themselves.

GO TO GO TO LONDON
Same as GOTO, this format eases programmers upgraded from a Spectrum (hint hint).

HEX$ T$=HEX$(B)
T$ returns the hexadecimal equivalent of the decimal number B.

INKEY$ INKEY$
This returns the present key pressed when executed. This is the same as DPEEK(121)+PEEK(764).

INPUT INPUT "JACK FLASH";Z,P
No difference to the older INPUT, although you can now output text to the screen as shown. You also have this INPUT #16 offering with Turbo Basic.

INSTR M=INSTR(G$,A$,H)
M returns the starting position of A$ within G$. H is optional, though, it allows you to begin the search at byte H within G$. Different text case (capital, non-capital etc.) is treated completely different.

LIST LIST U,
In addition to the normal process of LIST, this format now allows you to list a program from U onwards.

LOCK LOCK "D:GEM"
DOS option F. Lock a file from Basic.

LOOP LOOP
2nd part of the DO/LOOP structure.

MOD O=P MOD K
O returns the integer remainder of P/K.

MOVE MOVE S,D,B
An excellant command for Basic users. MOVE will copy B bytes of memory beginning at S, and place them beginning at D. This is especially useful for PMGs.

-MOVE -MOVE S,E,B
This is exactly the same as MOVE, although the copying of the memory is performed backwards. Occasionally important.

ON ON Q EXEC / ON Q GO#
This now gives these 2 variations given.

PAINT PAINT X,Y
A complete fill of an object, where co-ordinates X and Y are within.

PAUSE PAUSE F
Using this command, pauses program control for F jiffies/frames. Multiply F by 50 to achieve the PAUSE time in seconds.

POP POP
Ofcourse, this now handles stack entries for all of the new structured programming commands.

APPENDIX A5:

PROC PROC HARRY
The initial part of a PROC/ENDPROC procedure. It defines the beginning of the routine/procedure HARRY.

PUT PUT R
This now acts exactly like ? CHR$(R); where R is the characer going to the screen.

%PUT %PUT #F,E
The opposite of %GET. See this function for further details.

RAND S=RAND(Y)
S returns a RANDom integer between 0 and Y.

RENAME RENAME "D3:CHUG,BIONIC"
DOS option E from Basic. RENAMEs file CHUG on drive 3 to BIONIC.

RENUM RENUM P,O,J
RENUMber all program lines from line P to O, in increments of J. GOTOs and TRAPs are handled, though, variable line-number references are not.

REPEAT REPEAT
1st part of the REPEAT/UNTIL structure. This just marks the beginning.

RESTORE RESTORE #TINA
You can now restore to a label name (#TINA) as well as line numbers.

RND Z=RND 0
Same as normal RND, but you can now ommit the brackets surrounding the number.

SOUND SOUND
A nice feature is this method of turning all of the sound channels off at once.

TEXT TEXT X,Y,M$
A greatly desired function which plots the text in M$ onto the screen, beginning at co-ordinates X and Y.

TIME ? TIME
Returns the present time in format HHMMSS. You can also set the time with TIME=HHMMSS.

TIME$? TIME$
The same as TIME, except for a string variable.

TRACE TRACE - / +
A very explicit command which allows you to debug your Basic programs. TRACE mode, when engaged, displays the current line number being executed.

APPENDIX A5:

TRAP   TRAP #VAT
You can now TRAP control to label names.

TRUNC   W=TRUNC(F)
W returns the integer part of F, the fraction is TRUNCated.

UINSTR   Z=UINSTR(P$,G$,H$)
Same as INSTR, although the different case of text is irrelevant. Inverse, non-caps etc. is now treated the same.

UNLOCK   UNLOCK "D4:FELIX"
UNLOCKs the file FELIX on drive 4.

UNTIL   UNTIL R
2nd part of the REPEAT/UNTIL structure. Program control will repeat until condition R is met.

WEND   WEND
2nd part of the WHILE/WEND structure. WEND marks the end.

WHILE   WHILE Y
1st part of the WHILE/WEND structure. This is similar to REPEAT/UNTIL except that execution of the WHILE/WEND structure doesn't process even once before meeting the condition.

--   --
A special form of REM. It chucks 30 dashes across a program line.

*B   *B- / *B+
This command allows the BREAK key to be TRAPped when enabled with *B+.

*F   *F- / *F+
This command corrects a bug in normal Atari Basic, where loops such as FOR J=2 TO 1:NEXT J would initially execute once, even though the condition is already ended. *F- also allows you to leave it in just incase.

*L   *L- / *L+
The line indent command.

#TWIGGY
This is the line-label pointed to by the GOTO#, TRAP# etc..

$ FOR I=$0600 TO $0900
Here's a nice feature that allows you to use hexadecimal numbers in Turbo Basic as you would use decimal ones.

&   V=A & B
V returns the result of A AND B.

!   T=U ! F
T returns the result of U OR F

=%0-#3 Z=%0 etc.
These 4 constants simply denote the numbers 0-3, respectively. The only difference is that using these in your program is that X=1 requires 10 bytes, while X=%1 only needs 4. It's good practice to assign values to variables if the values are used more than once since a great amount of memory can be 'lost' in large programs.

Well, indeed they are the expansions to normal Basic which basically make up Turbo Basic. There are still a few simple facts that you should know also, so I'll run through what I know.

Programs can now be typed in lower case as well as all the other cases normally acceptable except for the * commands and GO TO. The language itself occupy's less memory than the original Basic too. There are 9 new error codes from 22 - 30. As you may have already discovered, errors now provide a reasonably clear explanation by supplying a word which describes the problem. Error number 15 is also updated to account for a deleted REPEAT statement. Variable, procedure and label names may now contain the underscore character (SHIFT and MINUS). You can also print a quote within a PRINT statement by using double-quotes together, like: PRINT "GREAT""EH!?". If you wanted to autoload a program on entry to Basic, then originally an AUTORUN.SYS file had to exist on the disk, but since Turbo Basic now uses this, TB searches for a file named AUTORUN.BAS for your autoload program. The new IF/THEN structure can be used like so:

```
10 FOR J=1 TO 10
20   IF J]5
30     PRINT "HI"
40     ELSE
50     PRINT "LO"
60   ENDIF
70 NEXT J
```

The compiler which comes with TB is better yet, increasing program speed twice over plus! In addition to that, it has to be said that it is an excellant compiler. There are only a few keywords that are not compilable, they are: *L, TRACE, NEW, DUMP, RENUM and DEL.

Anyway, with this truly amazing package, I'm sure 1st class quality software can be created by the average Basic programmer easily. I look forward to TURBO creations.

APPENDIX A6:

HANDY TRICKS.
As a means of quick reference, I've included a whole list of handy little tricks that you might not know that you can do. Have fun reading them!

CONTROL-1 is possibly the most used. It's a pause/unpause toggle for any print being listed on the screen, inside or outside of almost every program. Basic or Machine-code. This can also be simulated in programs with values 255 and 0 POKEd into 767 for screen pause and unpause, respectively.

CONTROL-2 is buzzer sound.

CONTROL-3 causes Error-136. Some Basic programs disable the break key, but they can still be broken into by pressing this key when the program is awaiting an input. To prevent this, the input must be TRAPped.

SHIFT-TAB can be used to set a tab anywhere across a text line. Useful when editing Basic/assembly programs.

CONTROL-TAB will clear the tab set with the shift-tab.

BREAK-KEY can be disabled by POKE 16,64 and POKE 53774,64.

LISTING Basic programs after being broken into can be prevented by adding a POKE 202,1 within the program itself. This way, if the program was to be broken into, it would be automatically erased.

RESET-KEY can be TRAPped with POKE 2,52, POKE 3,185 and POKE 9,2. When Reset is pressed an error will occur, thus, the Basic program can TRAP reset to any line. All the pokes and the TRAP must be setup each time reset is pressed. It can also be forced to coldstart the computer with a value of 1 POKEd to location 580. Poke with zero to revert to normal.

WARMSTART can be done with X=USR(58484). Otherwise known as pressing reset.

COLDSTART can be done with X=USR(58487). Otherwise known as turning the computer off and on.

BYE in Basic can also be achieved with X=USR(58481).

LEFT MARGIN can be changed by location 82. A value 0 is useful when typing in program listings, whereby all the screen columns are accessable, giving an extra 6 bytes to each program line.

APPENDIX A6:

RIGHT MARGIN is changed at location 83. Similar to Left margin.

TEXT-WINDOW can be used in Graphics 0 with a POKE 703,0. A value of 24 will disable it. The DOS menu actually uses this technique.

INPUT can be obtained with the Basic INPUT statement of course, but to get rid of the dreaded question mark, use INPUT #16;X$. This does not work in Turbo Basic unfortunately.

SPEED up the initializing of your Basic programs 30% by turning the screen off with POKE 559,0. Turn it back on with a POKE value of 34.

INPUT/OUTPUT through the cassette or disk-drive can be made silent with a POKE 65,0. Poke with non-zero to turn it back on.

DISKS can have data written to both sides by notching an identical hole on the opposite side of the disk. Believe it or not, I have written letters to several people who never knew about this.

LOAD machine-code files from Basic with OPEN #1,4,0,"D:FILENAME.EXT" and X=USR(5576). You can also use XIO 41,#1,0,0,"D:FILENAME.EXT".

MUSIC can be played from the cassette-unit and through the TV speaker with a POKE 54018,52. A value of 60 will turn it off. My music system tends to wake everyone up, so this is a good resort when programming in the early hours of the morning.

SCREEN display width can be altered to narrow, standard or wide with values 33, 34 and 35 POKEd to location 559, respectively.

LISTED-FILES from Basic are saved to disk exactly as you see them on-screen. You can load them into a word-processor and include direct-mode instructions (without line-numbers) in-between the lines of code, and when you ENTER the listing back in Basic, the direct-mode lines will execute as the file is loading!

CAPS-LOCK can be turned on or off with values 64 and 0 put at location 702. Control-lock can be forced with a value of 128.

APPENDIX A6:

TEXT can be opaque, inversed and turned upside down with various values poked to location 755. It can also be forced in inverse mode with POKE 694,128 and reversed with a value of 0.

CHARACTER-SET can be chosen at location 756. Value 224 is standard. Poking with 204 gives international characters under the control-key presses. Non-capitals are also obtainable on graphics 1 with a poke value of 226.

ESCAPE CHARACTERS such as the arrows, can either be acted upon or displayed on the screen with values 0 and any non-zero value POKEd into location 766.

CURSOR can be be turned invisible with a non-zero value poked to 752, and returned to normal with 0.

KEYBOARD keys can be detected by peeking location 764. A value of 255 means no key has been pressed, other values are particular keys. These values tend to be a total mix-up, though, on XL's they can be converted to ascii equivalents by taking the PEEKed value of address PEEK(121)+256*PEEK(122)+PEEK(764).

SCREEN vertical adjustment can be performed by changing the value in location 560 between 9 and 31. An explosion effect can be achieved in a game by poking random values between this range successively.

PRINT all output that normally goes to the screen to the printer with POKE 838,166 and POKE 839,238. Return to normal with POKE 838,163 and POKE 839,246. On the XL, the 4 values are 202, 254, 175 and 242 in the above order, respectively.

FINE-SCROLL can be enabled at location 622 with a value of 255. Disabled with 0. Try enabling, calling Graphics 0 and listing a long program.

KEYBOARD can be disabled with a POKE 621,255. It can be enabled with a value of 0.

INITIAL key delay is at location 729. 0 for no repeat, 1 for fast and 255 for very slow.

KEY REPEAT RATE is at 730. Similar to 729 except for all repeats after the initial keypress.

KEY CLICK is at location 731. 0 means sound on and 1 is off.

APPENDIX A6:

HELP-KEY can be found at memory-location 732. A value of 17 means help is pressed, 81 means shift and help whilst 145 means control and help. I have actually had a value of 209 in this register.

CONSOLE-KEYS can be found at location 53279. A value of 3 means option is pressed, 5 means select and 6 means start. Multiple combinations can be detected also.

RANDOM numbers between 0 and 255 can easily be obtained by peeking location 53770. Numbers between 0 and 65535 can also be obtained with PEEK(53770)+256*PEEK(53786).

MEMORY can be cleared at the speed of machine-code, from Basic by using locations 88, 89 and 106 in conjuction with the screen clear function. Just set 88 and 89 to the LSB and MSB start address, and set 106 to the MSB end address. Then, when a Basic clear function is issued, all this memory will be zeroed. This is especially useful for clearing PMG's or strings.

BASIC can be switched off with POKE 1016,1. Pressing reset will boot DOS.

DOS can be written to a new blank disk without DUP with OPEN #1,8,0,"D:DOS.SYS" and CLOSE #1 in Basic.

DELETE a DOS file from Basic with XIO 33,#1,0,0,"D:FILENAME.EXT".

DOS ACCESS can be disabled from Basic by changing locations 10 and 11. Try POKEing 10 with 203, 11 with 0 and 203 with 96.

LOCK your disk files from Basic with XIO 35,#1,0,0,"D:FILENAME.EXT".

UNLOCK your disk files from Basic with XIO 36,#1,0,0,"D:FILENAME.EXT".

VERIFY can be turned off, when using DOS by POKEing 1913 with 80. A value of 87 turns it back on. Note that all DOS alterations will only remain permanent when a new DOS has been written to a blank disk.

RENAME your DOS files from Basic with XIO 32,#1,0,0,"D:OLDNAME.EXT,NEWNAME.EXT".

WILDCARD ASTERISK (*) can be altered by putting the new wildcard ascii code at location 3783.

APPENDIX A6:

FILENAME CHARACTER RANGE can be altered to accept punctuation, numbers and non-caps with POKE 3818,33 and POKE 3822,123.

DUP can be called up with X=USR(6518) if it has previously been called from Basic. Note that this is very fast, but is not always reliable.

AUTORUN FILES can be prevented from loading when a DOS disk is booted, by successively pressing break when you hear pips through the TV speaker. If READY does not appear then press reset.

FORMAT your DOS disks from Basic with XIO 254,#1,0,0,"D:" for medium density and XIO 253,#1,0,0,"d:" for single.

REVISION DATE of your Atari is in day, month and year order, and is at locations 49154-49156.

Well, thats all of them folks! Of course, if you know of any other handy little tricks then please send them down for all to know. In the event of scaled responses about this book, I may decide to make additional leaves or appendices. If this does happen, then you should be able to find out about these corrections, additions etc. through TWAUG.

APPENDIX B

APPENDIX B1:

SOUND AND MUSIC.

Although, not an explanatory appendix, here are some various references and useful programs for creating your own music. Of course, sound is produced on the Atari with the SOUND statement, or with POKEs to location 53760 - 53768; $D200 - $D208. In Turbo Basic you also get the DSOUND statement which allows you to create sound in a much higher range of frequencies (over 8 octaves). If you are writing music in your own machine-code programs, then after loading the program in, you must POKE 53775 with 3 and POKE 53768 with an initial setting value to initialize POKEY correctly. If you don't do this then you won't get any sound at all.

The parameters of the SOUND and DSOUND statements are as follows:

SOUND CHANNEL,PITCH,DISTORTION,VOLUME

DSOUND PAIR,PITCH,DISTORTION,VOLUME

The normal SOUND statement gives you a choice of 4 channels, 256 frequencies (PITCH), 16 distortions and 16 volumes. The DSOUND statement is the same for PITCH and VOLUME, but the PITCH range now gives 65536 frequencies. The PAIR parameter means which CHANNEL pair you wish to use. 0 refers to channels 0 and 1, while 1 refers to channels 2 and 3. This is how the frequency range is increased, by pairing 2 channels.

You can hear the difference with the following TURBO program:

```
10 FOR I=0 TO 65435
20 DSOUND 0,I,10,8
30 DSOUND 1,I+100,10,8
40 NEXT I
```

Standard Atari Basic only offers this frequency range:

```
10 FOR I=0 TO 245
20 SOUND 0,I,10,8
30 SOUND 1,I+10,10,8
40 NEXT I
```

If you got tired of waiting for the TURBO frequency range to end, then you can stop the sound with any of END, SOUND, DSOUND or just pressing RESET.

APPENDIX B1:

If you wanted to create the DSOUND equivalent from machine-code, then you would set the necessary bit at location 53768; $D208. That bit would be bit-4 and/or bit-3. Note, that you'll also have to set bit-6 and/or bit-5 also, depending on which channels you were pairing. Another point to note, is that when putting your volumes and distortions in the appropriate registers, you should zero the volume output in the lower channel of the 2. So if you paired channels 1 and 2, then the volume level in channel-1 should be 0.

There was a nice selection of different sound affects in the SOUND chapter of YOUR ATARI COMPUTER by LON POOLE, very useful. There were such sounds like:

```
10 REM HI-LO SIREN
20 FOR J=0 TO 9
30 SOUND 0,47,10,8
40 FOR L=0 TO 99:NEXT L
50 SOUND 0,64,10,8
60 FOR L=0 TO 99:NEXT L
70 NEXT J
```

```
10 REM BIRDS
20 FOR J=0 TO 9
30 FOR K=3 TO 10
40 SOUND 0,K,10,8
50 NEXT K
60 NEXT J
```

```
10 REM TAKE-OFF
20 FOR L=1 TO 5
30 FOR J=0 TO 45
40 SOUND 0,J,8,J/3
50 NEXT J
60 FOR J=45 TO 0 STEP -1
70 SOUND 0,J,8,J/6+6
80 NEXT J
90 NEXT L
```

```
10 REM EXPLOSION
20 FOR J=-10 TO 10
30 SOUND 0,200,4,10-ABS(J)
40 SOUND 1,255,4,10-ABS(J)
50 SOUND 2,225,4,10-ABS(J)
60 SOUND 3,150,4,10-ABS(J)
70 NEXT J
```

Anyhow, apart from playing particular sounds, how about creating music pieces? To create music, then you'll need to create a suitable routine, of course the following example would achieve this:

```
10 FOR N=1 TO 11
20 READ F
30 FOR V=8 TO 0 STEP -1
40 SOUND 0,F,10,V
50 NEXT V
60 NEXT N
70 DATA 251,193,162,128,108,91,72,63,47,40,31
```

It does play music, but the program is very limited. It doesn't allow for more than 1 channel, sustaining delays etc.. If you have TURBO BASIC, then try the program on the next page:

<u>APPENDIX B1:</u>

```
100 DIM F(4),S(4),V(4)
104 DO
108   READ A
112   IF A=-1 THEN EXIT
116   G=INT(A/10)
120   A=A-(G*10)
136   FOR Z=1 TO A
140     READ F
144     S=FRAC(F)*10
148     IF S=0 THEN S=1
152     F=INT(F)
156     F(Z)=F:S(Z)=S:V(Z)=8
160   NEXT Z
164   FOR P=G TO 0 STEP -0.5
168     FOR Z=1 TO 4
172       V(Z)=V(Z)-S(Z)
176       IF V(Z)[0 THEN V(Z)=0
180       SOUND Z-1,F(Z),10,V(Z)
184     NEXT Z
188   NEXT P
192 LOOP
196 END
200 ------------------------------
300 DATA 31,144,31,144,31,144
304 DATA 63,193,162,128.05
308 DATA 31,144,31,144,31,144
312 DATA 63,162,128,108.05
316 DATA 31,144,31,144,31,144
320 DATA 33,162,128,108
324 DATA 31,162
328 DATA 33,126,96,81
332 DATA 31,162
336 DATA 63,128,108,91
340 DATA -1
```

```
300 DATA 23,128,144,114
302 DATA 01,144,01,136,01,128
304 DATA 23,128,114,102
306 DATA 01,121,01,114,01,108
308 DATA 23,102,91,85
310 DATA 01,102,01,96,01,91
340 DATA -1
```

Of course, if you want to RUN this program in normal Atari Basic, then you'll have to convert the various LOOP structure commands to GOTOs etc.. Line 144 would change to S=(F-INT(F))*10. But it is not recommended because the playing time is considerably slower. You can of course speed the program up by compacting it; more instructions on 1 line etc., or even compiling it!

This routine is quite short, but believe me it can be used for quite complex tunes. The arrays are used so that the volume of each channel can decay at different rates. The global speed of the music is controlled by variable G, which can be changed in your data. See overleaf for a breakdown of variables and DATA meanings:

APPENDIX B1:

VAR FUNCTION

G	Global speed of tune
A	Amount of channels to play simultaneously
F	Current Frequency being READ
S	Current Sustain value being READ This value is the rate of volume decrement
F(Z)	Frequency for each channel
S(Z)	Sustain rate for each channel
V(Z)	Volume for each channel This value decreases from 8 at a rate of S(Z) until it reaches 0.

The meaning of the DATA:

The 1st number in my DATA is a value of 31. This means that the Global speed of music played is at a rate of 3, and the amount of frequencies to be played at once is 1, thus, 1 frequency is READ (144). After this frequency is played the program looks for another 1st number, if that number is -1 then the program ends, otherwise the whole process is repeated. Of course, should the 1st number be 33, then we know the global music speed, but note that 3 frequencies will be played at the same time, thus, 3 numbers/frequencies will be READ from the DATA lines. OK now, so we can have up to 4 channels play at the same time, and we can change the global speed of the music, but what else is there?

The frequencies in their present form decrement in volume at a rate of 1. But, we can make each channel decrement at different rates if we like. If we add a fractional value to the frequency, then this number multiplied by 10 is the volume decrement rate for the channel playing that frequency. For example; if the DATA frequency value READ was 144.26, then the frequency would be 144 as we know, but the volume decrement rate would change from 1 to 2.6, get it!?

Now, the program also includes some other particular qualities as well as them mentioned. If the volume decay rate is very slow, and the next set of frequencies are READ before the last frequencies have died out, then the old volumes will continue to decay, so long as the old frequencies channel numbers are greater than the new frequencies channels used. For example; if the last played frequencies occupied all the channels and none of them have finished playing, then the only frequencies that won't stop decaying in volume are those which are replaced by the following frequencies. If only 1 frequency is read, then only channel-0 is used, channel-1 is for a 2nd frequency etc.. Also, if you want no-sound output (a quite delay) then add .8 to the frequency you want in silence. Simple as that!

Things might sound complex at first, but you should get the hang of it.

APPENDIX B1:

You can also improve the routine to allow for different distortions by reading a distortion for every frequency, or perhaps better still, you can break up the Z loop in lines 168 - 184 and put everything in single statements'if you follow me, this way you can use each channel for a set distortion. If you wanted to retain several of pure tone, and 1 of distortion 12 perhaps, then the best way to do it would be to use channel-0 for distortion 12, and the rest can be used as pure tone depending on how many channels you wanted playing at the same time.

Anyway, here's a table of equivalent piano notes for the Atari frequencies:

LOW	C	251
FREQ.	C# Db	230
	D	217
	D# Eb	204
	E	193
	F	182
	F# Gb	173
	G	162
	G# Ab	153
	A	144
	A# Bb	136
	B	128
MIDDLE	C	126
	C# Db	114
	D	108
	D# Eb	102
	E	96
	F	91
	F# Gb	85
	G	81
	G# Ab	76
	A	72
	A# Bb	60
	B	64
	C	63
	C# Db	57
	D	53
	D# Eb	50
	E	47
	F	45
	F# Gb	42
	G	40
	G# Ab	37
	A	35
	A# Bb	33
HIGH	B	31
FREQ.	C	29

That's about it really for this musical appendix, of course, you can expand on the main program included, if you do then please send me a copy.

APPENDIX B2:

VOLUME-ONLY SOUND.

Perhaps, a less known feature in the Atari would be Bit-4 of the AUDC# registers at locations 53761 - 53767. The bit which allows total control over sound-wave generation.
But why on earth would the wave-form of a sound need to be changed? And if I knew why, then how could it be done?

The reason why the wave-form of sound is changed is quite simple. The sound generated by the POKEY sound-chip is in the form of a square wave, but the sound of a piano for example is triangular; oh yes, you can simulate it with a hit, sustain, release type manner of sound, but it still doesn't sound quite like a piano. Other instruments also have different wave-forms, your own voice generates sound in intricately mixing sine waves! OK, so we know the reason for changing the wave-form, but how do we do it?

Take the Basic line: 10 POKE 53279,0:GOTO 10

This is making the TV speaker POP back and forth, causing the sound-vibrations in the air (wave-form). Try adding: 5 REM, to the program. This slows Basic down just a fraction more and affects the noise being made.
The sound itself is quite broken up using Basic, you really need to use machine-language for proper affects. There is a program that does just this at location 53279.

To create different shaped waves, you affectively need to change the position of the speaker (the volume at the AUDC# register) at different times, the actual note is just the global frequency of the wave-form. Here is the piano-wave:

```
volume
   0¦         -
   1¦        - -
   2¦       -   -
   3¦      -     -                    The Piano-wave;
   4¦     -       -                   you should realise
that
   5¦    -         -                  each delay of time
remains
   6¦   -           -                 the same for every
   7*  -             -                step of changing
volumes.
   8¦                 -             -
   9¦                  -           -
  10¦                   -         -
  11¦                    -       -
  12¦                     -     -
  13¦                      -   -
  14¦                       - -
  15¦________________________-___________

                                    time
```

APPENDIX B2:

Here's an assembly program to play a triangle wave;

```
100 ;
110          *=$CB
120 ;
130 ;
140 TEMPO    .BYTE 1
150 MSB      .BYTE 0
160 ;
170          *=$5000
180 ;
190          LDA #$3
200          STA  $D20F      ;SKCTL
210          LDA #$0
220          STA  $D208      ;AUDCTL
230          STA  $D40E      ;KILL VBIs
240          STA  $D20E      ;KILL IRQs
250          STA  $D400      ;KILL DMA
260 ;
270 REP      LDX #$0
280 PHASE    LDA DEL         ;,X OPTIONAL
290          STA MSB         ;VOL.DELAY
300          LDA VOLUME,X
310          ORA #$10        ;SET VOLBIT
320 DO       LDY TEMPO       ;FREQUENCY
330          STA  $D201      ;AUDC#
340 W        DEY             ;TIME-STEP
350          BPL W
360 ;
370          DEC MSB         ;FREQ HI-BIT
380          BPL DO          ;CONTROL
390 ;
400          INX             ;NEXT VOLUME
410          CPX #$1E        ;(29 IN TOT.)
420          BNE PHASE       ;LOOP IF LESS
430 ;
440          JMP REP         ;HARD REPEAT
450 ;
460 ;
470 VOLUME .BYTE 7,6,5,4,3,2,1,0,1,2,3
480          .BYTE 4,5,6,7,8,9,10,11,12
490          .BYTE 13,14,15,14,13,12,11
500          .BYTE 10,9,8
510 ;
520 DEL      .BYTE 1
```

If you wish to try different values in TEMPO by using the keyboard, then delete line 240 and change line 320 to:

```
320          LDY  $2FC
```

You may not think that it sounds like a piano at the moment, thats because the note doesn't decay. Here follows a program that you can add/alter to the previous one that will give you a decaying piano-wave, just make sure that you use the particular line numbers given:

```
152 REPS    .BYTE 20
154 REPCPY .BYTE 20
156 CTR     .BYTE 5
158 CTRCPY .BYTE 5

210 RERUN   LDA #$0
240         STA  $D20E     ;KILL IRQs

300 VT      LDA VOLUME,X
410 XD      CPX #$16       ;(22 IN TOT.)

440         DEC REPS       ;DECAY
441         BNE REP        ;DELAY
442         LDA REPCPY
443         STA REPS
444 ;
445         CLC
446         LDA VT+1       ;SELECT
447         ADC XD+1       ;NEXT
448         STA VT+1       ;WAVE
449         LDA VT+2
450         ADC #$0
451         STA VT+2
452 ;
453         DEC CTR        ;WAVES DONE?
454         BPL REP        ;NO
455         LDA CTRCPY     ;YES
456         STA CTR
457         LDA DEL&255    ;RESTORE
458         STA VT+1       ;ORIG
459         LDA DEL/256    ;WAVE
460         STA VT+2       ;ADDR
461 ;
462         LDA  $10       ;RESTORE
463         STA  $D20E     ;IRQs
464;
465 L       LDA  $2FC      ;WAIT
466         CMP #$FF       ;FOR
467         BEQ L          ;KEY
468         STA TEMPO      ;NEW-NOTE
469         JMP RERUN
470 VOLUME .BYTE 7,8,9,10,11,12,13
471         .BYTE 14,13,12,11,10,9
472         .BYTE 8,7,6,5,4,3,4,5,6
473 ;
475         .BYTE 7,8,9,10,10,11,12
476         .BYTE 13,12,11,10,10,9
477         .BYTE 8,7,6,5,5,4,5,5,6
478 ;
480         .BYTE 7,8,9,9,10,10,11
481         .BYTE 12,11,10,10,9,9,8
482         .BYTE 7,6,6,5,5,5,6,6
483 ;
485         .BYTE 7,8,8,9,9,10,10
486         .BYTE 10,9,9,8,8,7,6,6
487         .BYTE 6,6,6,6,6
```

```
488 ;
490         .BYTE 7,7,8,8,9,9,10,10
491         .BYTE 10,9,9,8,8,7,7,7
492         .BYTE 7,7,7,7,7,7
493 ;
495         .BYTE 7,7,7,8,8,9,9,9,9
496         .BYTE 8,8,8,7,7,7,7,7,7
497         .BYTE 7,7,7,7
498 ;
```

The way in which the piano-wave decays, is by playing successive waves whose top and bottom volume-peaks gradually flatten to one centralized volume. Try graphing my decay waves, each wave is 22 bytes in length. If you have graphed my decay-waves, then you'll notice that 8-bits (16 volumes) tends to be very limited, ie. the lower volume piano-waves are losing triangularity and becoming more like sine-waves.

POKEY - out of tune?
If a person told you that the music on your Atari was "out of tune", would you believe it? This piece of nonsense!? You'd probably have to work it out, wouldn't you, but how do you go about that?
Well, in COMPUTE!s 2nd book of Atari, there is a good article describing this subject, by Fred Coffey. Here's a brief overlook of it;

Considering the fact, that on the musical scale, the "A" note above middle "C" is 440Hz, we should be able to find out if our Atari really is in tune or not. Referring back to the Atari Basic manual, it says that to achieve this note, then you should use the number 72 in the pitch control of SOUND C,P,D,V. So how does POKEY derive 440Hz from the pitch value 72? Plug it through the following formula, and you'll find out:

PITCH=63921/(2*(P+1))

Did you come up with 437.8Hz? The Atari IS out of tune! OK, I admit, my Atari is out of tune, but is there anything that we can do about it?
Yes there is; 1 method of achieving this is to use the program listed earlier to pop a wave-form this many times per second. But, I like to take the easy way out wherever possible, so moving quickly onto method 2:

At AUDCTL, location 53768 we have bits that control 16-bit precision, and a 1.79MHz clock. It's these bits that we need to set, not all of them, only the ones necessary; bits-6 and 4 will do the trick, decimal 64+16=80, so POKE AUDCTL with 80. Perform a POKE 53763,(16*10)+8 to set distortion pure and volume at 8 also.
Now, you should understand that AUDF0 at 53760 and AUDF1 at 53762 are now changed from 2 seperate pitch channels, to one pitch being returned as AUDF2*256+AUDF0. If you stick 440Hz through the following formulas as PITCH, then you should be returned with the values that we need to POKE at AUDF0 and AUDF2:

APPENDIX B2:

P2=INT((178979/(2*PITCH)-7)/256
P1=INT(1789790/(2*PITCH)-7-256*P2*.5)

thus,

POKE AUDF0,P1 and POKE AUDF2,P2

OK, but is this 440Hz? Notice the use of the INTeger function in the P1 and P2 formulas. Are we smack on target, or are we off? If so, by how far?
Let us have a look. Substitute the P1 and P2 values in the following formula:

PITCH=1789790/(2*(256*P2+P1+7))

What is the pitch returned? Did you get 439.97Hz? Well, what can you say... only .03Hz off target! An improvement of 2.1Hz!! That can't be too bad.

SAMPLEing - how is it done?
Have you ever wanted to play SamPLe (.SPL) files in your own demos or programs? If you have then here's an assembly listing of the .SPL play routine:

```
100 ;
110 ;    SPL play routine
120 ;    brought to you by TOMO
130 ;    June '93
140 ;
150 ;
160      *=$2134
170 ;
180      LDA #$C0       ;dat-endpag+1
190      STA  $CD
200      LDA #$40       ;dat-startpag
210      STA  $CC
220      LDA #$0
230      STA  $CB
240      STA  $D20E     ;kill-IRQs
250      STA  $D40E     ;kill-NMIs
260 ;
270      LDA  $216      ;store
280      STA  $CE       ;VIMIRQ
290      LDA  $217
300      STA  $CF
310 ;
320 ;
330      LDA #$IRQ&255 ;set
340      STA  $216      ;addr
350      LDA #$IRQ/256 ;of new
360      STA  $217      ;VIMIRQ
370 ;
380      LDA #$3        ;2-tone
390      STA  $D20F     ;mode-off
400      LDA #$0
410      STA  $D200     ;initPOKEY
```

```
420      STA  $D208    ;CLK-rate
430      STA  $D202
440      LDA #$3       ;IRQ
450      STA  $D200    ;rate
460      LDA #$1
470      STA  $D20E    ;enable-IRQs
480      LDA #$A0
490      STA  $D201    ;rep-timer
500 WT   LDA  $CD
510      CMP  $CC      ;music
520      BNE WT        ;played?
530 ;
540      LDA #$0
550      STA  $D20E    ;kill-IRQs
560      LDA #$3       ;2-tone
570      STA  $D20F    ;mode-off
580      LDA  $CE
590      STA  $216     ;restore
600      LDA  $CF      ;orig
610      STA  $217     ;VIMIRQ
620      LDA  $10      ;
630      STA  $D20E    ;orig VIMIRQ
640      LDA #$40      ;&
650      STA  $D40E    ;NMIs
660      BRK           ;prog-end
670                    ;RTS or
680                    ;whatever
690 ;
700 IRQ  PHA
710      LDA #$0       ;used to
720      STA  $D20E    ;keep
730      LDA #$1       ;IRQ
740      STA  $D20E    ;going
750      STA  $D209    ;self-cause
760 X    BNE V1
770 V1   LDY #$0
780      LDA ($CB),Y   ;sample
790      LSR A
800      LSR A         ;take hi
810      LSR A         ;volume
820      LSR A
830      ORA #$10      ;set-VOLBIT
840      STA  $D203    ;AUDC1
850      LDA #$14      ;other-
860      STA X+1       ;volume
870      PLA
880      RTI
890 ;
900      LDA ($CB),Y   ;sample
910      ORA #$10      ;set-VOLBIT
920      STA  $D203    ;AUDC1
930      INC  $CB      ;next-byte
940      BNE P
950      INC  $CC      ;next-page
960 P    LDA #$0       ;reset for
970      STA X+1       ;hi-volume
980      PLA
990      RTI
```

APPENDIX B2:

The program in its present form is only 162 bytes long, so it will go just about anywhere. It uses $CB and $CC for the start address of the sample, $CD for its end address+1, which points to page-192 ($C0). $CE and $CF just retain the address of VIMIRQ so that it can be restored later. When the Interrupt is processing, program execution is held at WT in line 500. There are 2 parts to the interrupt, the 1st one (lines 700 - 880) plays the volume which is stored in the higher half of the byte, where as the 2nd part plays the volume stored in the lower half of the byte. This is how the SPL volumes are stored in memory as a means of condenseness.

With a few modifications, it's also possible to play more than one .SPL file at the same time. For example, make the following alterations/additions:

```
180     LDA #$80        ;dat-endpag+1
232     STA  $D0
234     LDA #$80        ;sample-2
236     STA  $D1        ;address

841     LDA ($D0),Y     ;sample-2
842     LSR A
843     LSR A           ;take hi vol
844     LSR A           ;of sample-2
845     LSR A
846     ORA #$10        ;set-VOLBIT
847     STA  $D205      ;AUDC3

922     LDA ($D0),Y     ;sample-2
924     ORA #$10        ;set-VOLBIT
926     STA  $D205      ;AUDC3

932     INC  $D0        ;next-byte
952     INC  $D1        ;next-page
```

The modified program will now play 2 .SPL files, however, the 1st sample must occupy memory $4000 - $7FFF and the 2nd must occupy $8000 - $BFFF. Notice, in this example, each sample has the same amount of memory reserved for it ($4000 bytes). Sample-2s address is stored in $D0 and $D1. Playing 2 samples at the same time takes a little bit more time to process, so you may find you'll have to alter the value $A0 on line 480 to a lower one. If the system doesn't achieve the time you require, it will cease, so the only other way of getting around this is to turn the screen off by loading location 559; $22F with 0. You may be able to get away with just turning half the screen off, but if you do try this, it might be best to avoid the use of a DLI. Instead use direct mode to read VCOUNT at 54283. If you do use a DLI, then keep it as short as possible: SEI, LDA #$0, STA $22F, CLI and RTI. You might even have time in the actual IRQ. Have fun!

APPENDIX B3:

POKEY IN STEREO?

To what extent will the XL/XE go to prove that it is the best 8-bit computer in existence? People said at one time that the main power of the Atari was its graphics (nice one Antic and GTIA). The Amstrad 464 thought it was the best computer for word-processing and printing, until Atari proved them wrong (nice one Antic and POKEY), and Commodore thought it was the best music 8-bit, until a program called Softsynth came to the Atari and made several demo disks, one of which is called World of Wonders. And now, the user of the Amiga 16-bit computer thinks he's in the clear with his sound chip, "Paula" or something like that! But have I got news for them or what! You too, like the Amiga user, can have 4 channel stereo sound (nice one POKEY2). In fact, the Amiga only has 2 channels per speaker. This modification gives the Atari 4 channels per speaker!

Anyhow, if you want to make the modification, then it's at your own risk, you also void any warranty you might have on your computer, but who needs warranties, Atari 8-bits don't go bang! (do they!??).

The parts you'll need are:

- Pokey chip (C012294)
- 74LS14/74HCT14 Inverter IC
- 1000 Ohm resistor, 1/4 Watt metal film 2-5% tolerance
- Two RCA style stereo jacks
- Two 100nF 16V bypass capacitors
- Two 12" strips of shielded audio cable
- One double-polar select switch (DPDT)
- Optionally, two 50K single turn trimmer pots

total cost approx. £8.50

All you need to fit everything together is a soldering iron, solder, a steady hand and a little bit of experience. The instructions have been made as clear as possible and double-checked for errors, so as long as you follow the instructions carefully then, hopefully, nothing should go wrong. One note, though, be careful not to hold the soldering iron on the soldering joint too long. You might also take an additional care, by wearing a static wrist band if you have one, but its not that important.

APPENDIX B3:

FITTING:

1. THE INVERTER:

a) Bend up all pins except 7 and 14
b) Cut off the narrow part of all the pins that were bent up
c) Install the inverter over the top of the existing 74LS14 inverter on the mother-board
d) Solder pins 7 and 14 of this new inverter to the same pins of the original inverter beneath
e) Run a small wire from pin-1 of the new inverter to pin-13 of the CPU. The CPU is part number C014806 on the XL/XEs
f) Unsolder and remove the 3K pullup resistor which is connected between pin-31 and Vcc of the original Pokey chip
g) Run a small wire from pin-2 of the new inverter to pin-3 of the same inverter, and then from there to pin-31 of the original Pokey. Note, that you can use the pad where you removed the 3K pullup resistor but be sure to get the correct one!

2. The 2nd POKEY:

a) Bend up all the pins on the 2nd Pokey which are marked with a minus-sign from the diagram shown on the next page. This includes: 8, 9, 10, 11, 12 13, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25 26, 27, 28 and 29
b) Cut off the narrow parts of all the pins that are bent up on Pokey2
c) Tin every pin which was not bent up on Pokey2, this includes pins: 1, 2, 3, 4, 5, 6, 7, 17, 30, 31, 32 33, 34, 35, 36, 37, 38, 39 and 40. These pins are marked with the hash (#) sign
d) Now, bend up pins 10, 31 and 37 which are indicated with a dollar sign ($), but do NOT cut these pins short!
e) Place the new Pokey on top of the original Pokey in piggy-back style
f) Solder the unbent pins of the new Pokey to the original Pokey.

...more continuing

APPENDIX B3:

The POKEY pinouts:

```
              ____________________
             |         ()         |
    Vss  #   |01                40|  #  D2
     D3  #   |02                39|  #  D1
     D4  #   |03                38|  #  D0
     D5  #   |04             $  37|  #  Audio OUT
     D6  #   |05                36|  #  A0
     D7  #   |06                35|  #  A1
     02  #   |07                34|  #  A2
   Pot6  -   |08                33|  #  A3
   Pot7  -   |09                32|  #  R/W
   Pot4  -   |10 $           $  31|  #   CS1
   Pot5  -   |11                30|  #  /CS0
   Pot2  -   |12                29|  -  /IRQ
   Pot3  -   |13                28|  -  Data OUT
   Pot0  -   |14                27|  -  A Clock
   Pot1  -   |15                26|  -  B Clock
   KS 2  -   |16                25|  -  KS 1
    Vcc  #   |17                24|  -  Data IN
 Keyb.5  -   |18                23|  -  Keyb.0
 Keyb.4  -   |19                22|  -  Keyb.1
 Keyb.3  -   |20                21|  -  Keyb.2
             |____________________|
```

g) Solder the 1000 Ohm resistor from pin-37 of the new Pokey to Vcc. The most convenient place to pick-up Vcc is where the 3K pull-up resistor was removed earlier
h) Solder a wire from pin-31 of the new Pokey to pin-4 of the new inverter
i) Mount the two RCA jacks on the rear of the case preferably in an area near the Pokeys
j) Solder a bypass capacitor to each of the centre conductors of the RCA jacks
* k) With the trim-pot knob facing you, pin-1 should be to the left side. Solder a wire from this pin on each trimmer, to a ground trace on the motherboard
* l) Connect the free end of the bypass capacitor to the centre pin of the trimmer (one capacitor to each trimmer)
* m) Connect the shields of the audio cables to the provided solder lugs on each RCA connector, and the centre conductor of the free terminal of each trimmer
n) Connect the centre conductor of the free end of the audio cable which is connected to the left RCA jack/trimmer/cap to pin-37 of the original Pokey
o) Connect the centre conductor of the free end of the audio cable which is connected to the right RCA jack/trimmer/cap to pin-37 of the new Pokey

p) The shield of the audio cable on the Pokey end should be cut and taped, or heat shrinked so that it does not touch anything
q) Run a 18-20 AWG wire from the ground lug of the RCA jacks to the wide ground area on the motherboard. This normally makes contact with the shield-box that covers the motherboard

3. FINISHING OFF:

You will now be able to connect the 2 RCA cables to an AUXiliary input to a tape, level input of a stereo or a boom box. You might find it better to centre the trimmers in their travel, adjusting them as needed to get best clarity. Glueing the trimmers to the back of the shell is a good point to note, to stop them from moving around inside.

Steps k, l and m which are marked with an asterisk (*) are not a necessity. The Pokey outputs can work fine without the trimmers connected. Just connect the bypass capacitors on each RCA jack to the appropriate audio cable centre conductors. You can also fit a switch into the setup, which will allow you to select between the normal stereo-mono output and the new stereo-stereo output. Just type in the following program to see a diagram of the switch circuit:

```
100 GRAPHICS 24
102 POKE 709,0:POKE 710,252
104 DIM A$(40)
106 DL=PEEK(560)+256*PEEK(561)
108 DM=PEEK(DL+4)+256*PEEK(DL+5)
110 COLOR 1
112 SET=PEEK(756)*256
114 GOTO 140
116 FOR J=1 TO LEN(A$)
118 C=ASC(A$(J,J))
120 NC=C
122 IF SGN(C-96)=-1 THEN NC=C-32
124 IF SGN(C-32)=-1 THEN NC=C+64
126 CH=SET+NC*8
128 FOR I=0 TO 7
130 AREA=DM+J*D+I*40+X+Y*40
132 POKE AREA,PEEK(CH+I)
134 NEXT I
136 NEXT J
138 RETURN
140 D=320
142 FOR Q=1 TO 6
144 READ A$,X,Y:GOSUB 116:NEXT Q
146 DATA OLD,2,40,NEW,12,40
148 DATA POKEY,3,32,POKEY,13,32
150 DATA O O O O,15,96,O   O,18,104
152 D=1
154 FOR Q=1 TO 15
156 READ A$,X,Y:GOSUB 116:NEXT Q
```

```
158 DATA 37,4,40,37,14,40
160 DATA 100nF,23,8,100nF,23,56
162 DATA AUDIO,30,12,AUDIO,30,42
164 DATA LEFT,30,21,RIGHT,30,51
166 DATA DOUBLE-POLAR,26,96
168 DATA SWITCH,29,104,LEDs,9,128
170 DATA GROUND,0,140,-Vcc,1,149
172 DATA R=220 Ohms,26,152
174 DATA +5 VOLTS,28,136
176 FOR Q=1 TO 26
178 Z=NOT (Q-3)
180 READ P1,P2,P3,P4
182 GOSUB 188
184 NEXT Q
186 GOTO 230
188 FOR W=0 TO Z
190 PLOT P1+W,P2+W
192 DRAWTO P1+P3+W,P2+W
194 DRAWTO P1+P3+W,P2+P4+W
196 DRAWTO P1+W,P2+P4+W
198 DRAWTO P1+W,P2+W
200 NEXT W
202 RETURN
204 DATA 12,37,21,44,92,37,21,44
206 DATA 114,102,42,58,264,148,40,0
208 DATA 252,146,10,4,151,148,99,0
210 DATA 8,148,64,0,72,140,0,16
212 DATA 72,140,24,0,104,140,17,0
214 DATA 72,156,24,0,104,156,17,0
216 DATA 72,124,48,0,72,20,0,104
218 DATA 72,20,144,0,34,50,38,0
220 DATA 114,50,10,0,124,50,0,55
222 DATA 150,116,16,0,166,50,0,66
224 DATA 166,50,50,0,222,50,64,0
226 DATA 222,20,64,0,125,116,20,0
228 DATA 125,148,20,0,137,116,0,32
230 FOR Q=1 TO 4
232 READ A$,X,Y:GOSUB 116:NEXT Q
234 DATA K,11,137,K,11,153
236 DATA H,26,17,H,26,47
238 PLOT 103,137:DRAWTO 103,144
240 PLOT 103,153:DRAWTO 103,160
242 PLOT 223,20:PLOT 223,50
244 PLOT 156,130:DRAWTO 230,110
246 COLOR 0
248 FOR Q=0 TO 1
250 PLOT 219+Q,17:DRAWTO 219+Q,54
252 NEXT Q
254 FOR Q=1 TO 3
256 READ A$,X,Y:GOSUB 116:NEXT Q
258 DATA The Stereo-stereo / stereo-mono switch,0,167
260 DATA Use shielded audio cable for all,3,175
262 DATA connections between Pokey and audio!,1,183
264 GOTO 264
```

APPENDIX B3:

There is one problem with the stereo-stereo mode, and that is when you try to play music or samples that are not modified or specially made for the stereo upgrade, you will hear these sounds from the left speaker only. But, if you add the switch, pin-37 of the old Pokey will lead to both speakers, thus, not using the new Pokey.

...where in memory is the new Pokey?

The original Pokey registers from $D200 - $D20F remain unchanged. For a full explanation of how the AUDF#, AUDC#, AUDCTL and SKCTL registers, see the appropriate locations.

The new Pokey registers take the following locations:

Address:	Name:	R/W	Function:
53776/D210	AUDF5	W	Audio #5 frequency
53777/D211	AUDC5	W	" " control
53778/D212	AUDF6	W	" #6 frequency
53779/D213	AUDC6	W	" " control
53780/D214	AUDF7	W	" #7 frequency
53781/D215	AUDC7	W	" " control
53782/D216	AUDF8	W	" #8 frequency
53783/D217	AUDF9	W	" " control
53784/D218	AUDCTL2	W	AUDIO CONTROL
53791/D21F	SKCTL2	W	SERIAL PORT CONTROL

The SKCTL2 register controls various functions of the Pokey device, and only needs to be initialized to a value of 3 to assure the additional 4 channels are active and ready. You can also test to see if your new Pokey works by PEEKing the AUDF# and AUDC# registers. If they return constant 0, then all is ok. You can also test this through the keycode register at $D209, with that of $D219 and if $D219 is 0, the upgrade is installed. You may want to mask the IRQs during the test for safety. This program will do the test for you, if the screen turns black then it is ok, else something is wrong:

```
10 DATA 104,120,173,9,210,141,198,2
12 DATA 88,96
14 FOR I=0 TO 9
16 READ D:POKE 1536+I,D:NEXT I
18 X=USR(1536)
```

where to from here...?

APPENDIX B3:

All we need now is some software to operate the new Pokey. So, get cracking experts! However, for the time being try the following program:

```
100 POKE 53768,5:POKE 53784,80
110 POKE 53775,3:POKE 53791,3
120 POKE 53760,254:POKE 53761,168
130 POKE 53764,255:POKE 53765,168
140 POKE 53777,160:POKE 53779,168
150 POKE 19,0:POKE 20,0
160 POKE 53776,PEEK(20)
170 POKE 53778,PEEK(19):GOTO 160
```

This will only work properly in stereo-stereo mode, so make sure your switch is set correctly.

Well, what can you do with a feature that is presently incompatible with all existing software? What else, but to change the existing software to MAKE it compatible!
You can do this by searching for all the 'pokes' in the old pokey and replacing two channels with two of the new pokey.

But where there are answers, there are problems. Like poking in different ways:

```
STA $D200, STA $D201
STX $D200, STX $D201
STY $D200, STY $D201
```

or even like:

```
LDY #1
STA $D1FF,Y or STA ($CB),Y
```

Which is very irritating to find (though, good protection).
Another problem is that some programs link channels to use 16-bit sound or filtering. See location 53768.

If the program doesn't use filters or 16-bit sound, then you can substitute channels 1 and 2 from the original Pokey addresses, to channels 3 and 4 of the new Pokey addresses. This way, the program will play in stereo-stereo for upgraded machines, but unchanged for the unmodified dudes.
If it does use filters, then you've got a problem in compatibility if you change it. You can exchange the 2 channels (1 and 3, or 2 and 4) from the old Pokey, for the same channels on the new Pokey and initialise AUDCTL2 with the same value as AUDCTL, but the program will only work properly with the upgraded system.

If there are channels used for 16-bit resolution, then you can also exchange the channels over for the other Pokey, but you need to also change the bit you set in AUDCTL. If AUDCTL sets bit-4, then after changing them over, AUDCTL2 should set bit-3. Test for bits 5 and 6 also.

APPENDIX B3:

You should note, that the changes made with filters and 16-bit sound are not always compatible, because, since the program uses two channels for 1 sound, it usually uses 1 or both remaining channels for additional sounds, and it is these channels that are lost on the unmodified system when playing a modified tune in this way.

If you have a copy of World of Wonders, perhaps the best music demo on the Atari, then you can turn it into stereo by using a sector editor.
The program stores volumes into the AUDC# registers on sector 1008, bytes $0B, $21, $38 and $51. All you have to do is add $10 to which-ever bytes above, to make them use the 2nd Pokey.

Here's a program that converts Softsynth to stereo:

```
100 DATA 62,4,87,4,37,28
102 DATA 61,4,86,4,36,28
104 DATA 66,4,91,4,44,28
106 DATA 21,23,21
108 DATA D:PLAY1.SYN,D:PLAY2.SYN
110 DATA D:PLAYB.SYN
120 FOR I=0 TO 20
130 READ D:POKE 1536+I,D:NEXT I
140 DIM F$(20)
150 FOR NR=0 TO 2
160 READ F$
170 XIO 36,#1,0,0,F$
180 OPEN #1,12,0,F$
190 FOR I=0 TO 2
200 NOTE #1,S,B
210 S=S+PEEK(1536+NR*6+I*2+1)
220 B=PEEK(1536+NR*6+I*2)
230 POINT #1,S,B
240 Q=PEEK(1554+I)
250 PUT #1,Q
260 NEXT I
270 NEXT NR
```

You can also change the music of a program yourself if you like. Some programs are easy to alter, but others are harder, it all depends on what music editor they were created on. Draconus, Zybex, panther, BMX simulator etc. (from the BIG demo) are ZUPKGC files. These are fairly easy to alter, because they store music in the sound registers only 1 place in the program.

APPENDIX B3:

All you'll need to search for is some machine-code that looks like this:

```
A2 08          LDX #$8
BD ?? ??  LOOP LDA  $????,X
9D 00 D2       STA  $D200,X
CA             DEX
10 F7          BPL LOOP
```

Where $???? might be any address!. You can change this loop to:

```
20 XX XX       JSR NEWROUTE
EA             NOP
EA             NOP
EA             NOP
EA             NOP
EA             NOP
EA             NOP
EA             NOP
EA             NOP
```

and include somewhere else in the program, where there is a suitable place of unused memory:

```
A2 04          LDX #$4
BD ?? ??  LOOP LDA  $????,X
9D 00 D2       STA  $D200,X
BD ?? ??       LDA  $????+5,X
9D 15 D2       STA  $D215,X
CA             DEX
10 F1          BPL LOOP
AD ?? ??       LDA  $????+8
8D 08 D2       STA  $D208
60             RTS
```

Don't forget that you have to initialize the new Pokey with:

```
               LDA #$3
               STA  $D21F
```

Credits for the upgrade go to Chuck Steinman, thanks also to Frankensteins information (I took from his articles).

APPENDIX C

APPENDIX C1:

CHARACTER CODES:

Here's the Atascii and Internal character codes inside the Atari.

CHAR	ATASCII	INTERN	CHAR	ATASCII	INTERN
space	32	0	L	76	44
!	33	1	M	77	45
"	34	2	N	78	46
#	35	3	O	79	47
$	36	4	P	80	48
%	37	5	Q	81	49
&	38	6	R	82	50
'	39	7	S	83	51
(	40	8	T	84	52
)	41	9	U	85	53
*	42	10	V	86	54
+	43	11	W	87	55
,	44	12	X	88	56
-	45	13	Y	89	57
.	46	14	Z	90	58
/	47	15	[	91	59
0	48	16	\	92	60
1	49	17	]	93	61
2	50	18	^	94	62
3	51	19	_	95	63
4	52	20	CTRL*,	0	64
5	53	21	CTRL*A	1	65
6	54	22	CTRL*B	2	66
7	55	23	CTRL*C	3	67
8	56	24	CTRL*D	4	68
9	57	25	CTRL*E	5	69
:	58	26	CTRL*F	6	70
;	59	27	CTRL*G	7	71
<	60	28	CTRL*H	8	72
=	61	29	CTRL*I	9	73
>	62	30	CTRL*J	10	74
?	63	31	CTRL*K	11	75
@	64	32	CTRL*L	12	76
A	65	33	CTRL*M	13	77
B	66	34	CTRL*N	14	78
C	67	35	CTRL*O	15	79
D	68	36	CTRL*P	16	80
E	69	37	CTRL*Q	17	81
F	70	38	CTRL*R	18	82
G	71	39	CTRL*S	19	83
H	72	40	CTRL*T	20	84
I	73	41	CTRL*U	21	85
J	74	42	CTRL*V	22	86
K	75	43	CTRL*W	23	87

APPENDIX C1:

CHAR	ATASCII	INTERN	CHAR	ATASCII	INTERN
CTRL*X	24	88	l	108	108
CTRL*Y	25	89	m	109	109
CTRL*Z	26	90	n	110	110
ESC	27	91	o	111	111
UP	28	92	p	112	112
DOWN	29	93	q	113	113
LEFT	30	94	r	114	114
RIGHT	31	95	s	115	115
CTRL*.	96	96	t	116	116
a	97	97	u	117	117
b	98	98	v	118	118
c	99	99	w	119	119
d	100	100	x	120	120
e	101	101	y	121	121
f	102	102	z	122	122
g	103	103	CTRL*;	123	123
h	104	104	¦	124	124
i	105	105	CLEAR	125	125
j	106	106	DELETE	126	126
k	107	107	TAB	127	127

Also see locations 121 and 122 for a few special Atascii characters. To achieve the inverse version of all the listed Atascii characters, simply add 128 to the character code value. There are only 128 internal codes, and to achieve the inverse runoffs of these characters, then bit-7 is set, or a value of 128 is added by the hardware when the character is outputted to the display. In addition to the above codes, there are also:

ATASC	FUNCTION	ATASC	FUNCTION
155	End Of Line	156	Delete Line
157	Insert Line	158	CTRL*TAB
159	SHIFT*TAB	253	CTRL*2 Buzzer
254	Delete Char	255	Insert Char

As well as the above codes, there is a 3rd standard unique to the Atari 8-bit. This standard is often referred to as the keyboard "RAW" value. The codes are as follows:

APPENDIX C1:

CHR KEY:	RAW CODE:	CHR KEY:	RAW CODE:
A	63	0	50
B	21	1	31
C	18	2	30
D	58	3	26
E	42	4	24
F	56	5	29
G	61	6	27
H	57	7	51
I	13	8	53
J	1	9	48
K	5	<	54
L	0	>	55
M	37	-	14
N	35	=	15
O	8	+	6
P	10	*	7
Q	47	;	2
R	40	,	32
S	62	.	34
T	45	/	38
U	11	TAB	44
V	16	SPACE	33
W	46	DEL	52
X	22	RETURN	12
Y	43	CAPS	60
Z	23	INV	39
		ESC	28

Of the 57 keys on the main keyboard, 53 of them can be used in 1 of 4 combinations. You can press the key on its own, use it with shift, with control or alternatively, use it with both shift and control held simultaneously. The code returned for the standard keypress is listed above, however, if you use shift, then add 64. If you use control then add 128. If you use both shift and control, then add 192.

APPENDIX C2:

NUMBER SYSTEMS:

Converting between number systems such as Decimal, Binary or Hexadecimal isn't that difficult once you got the right formulas or charts. Here's some varying ways of doing so:

Firstly let me give you true Binary columns:

```
DEC:   128  64  32  16   8   4   2   1
BIT:     7   6   5   4   3   2   1   0
```

You should see that if I threw a Binary number into this, like so:

```
         1   0   1   1   0   1   0   1
```

We have 1*128 + 0*64 + 1*32 + 1*16 + 0*8 + 1*4 + 0*2 +1*1, or to make it clearer: 128+32+16+4+1 which = 181 in decimal. Easy EH!?

You could if you wanted label the columns;

```
DEC:   8 4 2 1 8 4 2 1
BIT    7 6 5 4 3 2 1 0
```

So if we inserted the Binary number:

```
       1 1 0 1 1 0 1 1
```

Then we get 8+4+1 and 8+2+1, all we need to do is multiply the 1st half of the Binary conversion 8+4+1 by 16, and add the 2nd half. Hence, 8+4+1 = 13 * 16 = 208 + 8+2+1 = 219! Try the previous formula to prove it.

Decimal is Base-10 as you know, because each units, hundreds column etc. is a multiple of ten, ie. 1*10 = 10, 10*10 = 100 etc..

Binary is Base-2 because each column (digit) is a multiple of 2, ie. 1*2 = 2, 2*2 = 4, 2*4 = 8, 2*8 = 16 etc..

Hexadecimal is Base-16. If you wanted to convert any system to or from hex., you must 1st know what its 10th, 11th etc. digits are.

```
DEC: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
HEX:  F  E  D  C  B  A 9 8 7 6 5 4 3 2 1 0
```

As you can see, the number 0E in Hex. is 14 in decimal. But what if we had the number 9E...?

APPENDIX C2:

No problem. Knowing that Hex. is Base-16, we should see that the 1st column is 1*16, the 2nd is 16*16, then 256*16, 4096*16 etc., so:

9E in Hex. = 9*16 + 14 = 158 Dec.

Here's a table for easy reference:

DIGIT:

4th Hex	Dec	3rd Hex	Dec	2nd Hex	Dec	1st Hex	Dec
1000	4096	100	256	10	16	01	1
2000	8192	200	512	20	32	02	2
3000	12228	300	768	30	48	03	3
4000	16384	400	1024	40	64	04	4
5000	20480	500	1280	50	80	05	5
6000	24576	600	1536	60	96	06	6
7000	28672	700	1792	70	112	07	7
8000	32768	800	2048	80	128	08	8
9000	36864	900	2304	90	144	09	9
A000	40960	A00	2560	A0	160	0A	10
B000	45056	B00	2816	B0	176	0B	11
C000	49152	C00	3072	C0	192	0C	12
D000	53248	D00	3328	D0	208	0D	13
E000	57344	E00	3584	E0	224	0E	14
F000	61440	F00	3840	F0	240	0F	15

Binary and Hexadecimal conversion is probably the easiest of the lot! Take the Binary number:

01011011

All you have to do is split it in half, the left 4-bits becomes the left Hex. digit, and the right 4-bits becomes the right Hex. digit, ie;

0101 = 4+1 = 5, and 1011 = 8+2+1 = B (11 in Dec.), so our Hex. equivalent = 5B. For the decimal equivalent, just multiply 5*16 and add 11 as described earlier, 5*16 = 80 + 11 = 91 Decimal. To get the Binary value from the Hex. code, just reverse the operation!

There are many other ways to convert the numbers, but I feel that the ways I've described are the easiest and quickest!

If you wanted to convert a Decimal number to Binary, then you can do it like so:

Take the Dec. number 239;

APPENDIX C2:

239-128 = 111, so we have a 128 bit
111-64 = 47, 64 comes out too
47-32 = 15, yeap, a 32 also
15-16 goes negative, so this bit is 0
15-8 = 7, yes an 8 is there
7-4 = 3, a 4 too
3-2 = 1, and a 2
1-1 = 0, even the 1, all done

This gives:

11101111

All the bits except 16, and in fact, if you take 239+16, you get 255; 11111111. What a coincidence!

You might agree, that the hardest conversion to make is from Decimal to Binary, but who says that you have to go in a direct way. For instance, to convert the Decimal number 189 to Binary, why not go via Hex. 1st! 189/16 = 11 (or rather B) + the remainder which is 189-(16*B) (B Hex. = 11 Dec.) which gives you 13 Dec. or D Hex. Thus, you can now change BD to Binary. You can convert the B and the D as seperate parts, going back to single decimal numbers, called BCD (explained in a moment), thus: B = 11 and D = 13, so:

11=8+2+1 and 13=8+4+1 which gives:

1011 and 1101, or rather 10111101

Instead of going from Decimal, to Hex. and back to the singled decimal values, you can label the Binary columns in Hexadecimal, as shown:

ie:

BIT:	7	6	5	4	3	2	1	0
DEC:	128	64	32	16	8	4	2	1
HEX:	80	40	20	10	8	4	2	1

This way, changing the decimal number 189 to Binary, via Hex. would go like so:

189d/16d = Bh and 189d-(16h*11d) (11d = Bh) = 13d or Dh, so:

B0h = 80h+20h+10h = 1011 and
0Dh = 08h+04h+01h = 1101, so:
BDh = 10111101b

where h = Hex., d = Dec. and b = Binary

APPENDIX C2:

Binary Coded Decimal (BCD) is similar to what we came across earlier, where we took a Decimal number to Hex. and from there, we converted the two Hex. digits to seperate Decimal values to work-out which Binary columns to set in each half of a full Binary number.

When a number is in BCD, what is meant is that when you take the Binary sum of the byte, you must split the Binary into two halves, and the two decimal values extracted from each half of the Binary sum is the actual decimal number, you DO NOT multiply the 1st digit extracted by 16!, ie:

a BCD number is shown as:

10010110

Split the Binary into 1001 and 0110, and this returns 8+1 and 4+2, giving 96. Now the real Decimal equivalent of this Binary number is 9*16+6, but we do not do this because the number is a BCD one. It is MEANT to be 96d! This can be very confusing when reading a memory location that is in BCD format, because Basic returns the Decimal equivalent of the Binary bits. You'll have to convert that Decimal number to the Binary bits so that you can extract what the number is meant to be, a BCD number!
Thats about it with number systems, all you need to remember to convert to any other Base, is that each column multiplies the previous column by the Base.

APPENDIX C3:

LSBs AND MSBs

Most often used as pointers to tables and vectors to routines, where LSB is the Least Significant Byte and MSB is the Most Significant Byte. Take the following example:

```
DL=PEEK(560)+256*PEEK(561)
```

You will have come across this quite often in this book. The variable DL finds the address of the Antics Display List instructions in memory. The LSB (low byte) is in location 560 and the MSB (high byte) is in location 561. As you should know, you cannot perform:

```
POKE 560,39968
```

Not in an 8-bit computer anyway! So, to represent this address, we simply have to divide it by 256 to find the high-byte, and take the remainder for the low byte, hence:

```
HI=INT(39968/256)
LO=39968-(HI*256)
```

Another 2 formulas you will see often in this book. The number 256 is used as the division because this is the maximum amount of values that 1 memory location in the computer can have.

In assembly language, to take the high and low bytes would look something like this:

```
LDX ADDRESS/256      ;high byte
LDY ADDRESS&255      ;low byte
```

The high byte just finds the integer of ADDRESS divided by 256. The low byte ANDs the address with the low bits, and only returns a value whose binary bits are set. See the LOGIC appendice for an explanation of the AND function.

APPENDIX C4:

BOUNDARIES:

When you setup a Display List (DL), Display Memory or Player/Missile Graphics (PMGs), you need to organize them suitably in memory. The instructions of a DL cannot run through a 1K boundary, for example:

```
Addr: Instr:
$53FC 2
$53FD 2
$53FE 2
$53FF 2
$5400 2
etc.
```

This will not work, since the DL instructions run straight through a 1K boundary (1K = 1024; $400 bytes). You'll have to change this to something like:

```
$53FC $2
$53FD $1       JMP-instruction to address:
$53FE $00      LSB;
$53FF $54      MSB;   $00+256*$54 = $5400
$5400 $2
etc.
```

Display Memory (DM) must be organized so that it does not go through a 4K (4K = 4096; $1000 bytes) boundary. For example, if the mode-line columns were in memory like so:

```
0123456789ABCDEF0123456789ABCDEF01234567
0123456789012345678901234567890123456789
          |               |
    addr.$5FF0       addr.$6000
```

If the 10th byte in the line was location $5FF0, then you would think that the 26th byte would be location $6000. But, the 26th byte will actually be address $5000, the 27th byte would be $5001 etc. in this case. To avoid this, you should organize the memory correctly, do this by shifting the previous LMS address over so that the last byte of the 4K boundary is the last byte of the line. Should the last byte of the 4K boundary be the last byte of the line, then the 1st byte in the next line will continue after the 4K boundary, and into the next one, hence, everything is fine. You can also use the LMS instruction to point to the next area of memory to display.

APPENDIX C4:

When the Atari sets up Graphics modes 8, 9, 10, 11, 14 and 15, it achieves boundary crossing by inclusion of a 2nd LMS instruction at the point of the DL where it needs it. LMS instructions are also necessary when you want more than 4K on the screen at one time. Graphics mode 8 on an 800XL, for example, begins its DM at 33104; $8150. Here, only 94 lines of 40 bytes per line can be accommodated before the 2nd LMS instruction needs to point to the next 4K boundary because 94*40=3760. If you add 3760 to 33104, you get 36864; $9000 (the next boundary). And because there is more than 4K to be displayed on the screen at the same time, an LMS needs to be present.

Player/Missile Graphics have boundary limitations also. But, with PMGs, depending on what resolution you are using, you have to POKE the start address (hi-byte only) into PMBASE. With Double-line resolution, PMBASE must begin on a 1K boundary (a multiple of 4 pages), but with Single-line resolution, PMBASE must begin on a 2K boundary (a multiple of 8 pages).

When you set PMBASE with the appropriate value, a table is configured as shown in the PMBASE appendice. If you do not give an acceptable start boundary address, then Antic will not calculate the table correctly, as simple as that!

A suitable address for Double-line resolution can be taken from this formula:

```
POKE PMBASE,ADDRESS*4
```

This ensures, that whatever value you give variable ADDRESS, it is multiplied by 4, 1*4=1024, 2*4=2048 etc., they are all 1K boundaries!

The Single-line resolution formula would be:

```
POKE PMBASE,ADDRESS*8
```

Any value given this time will ensure that a 2K boundary is found correctly.

APPENDIX C5:

BOOLEAN EXPRESSIONS:

Boolean programming is quite a powerful technique that can totally re-configure the standard Basic program. Take the following example:

```
A=(F>10)
```

This is exactly the same as:

```
A=0
IF F<10 THEN A=1
```

IF/THEN statements take a fair bit of time to process in Basic, so doing without them would be a bonus.
Another format of the expression is:

```
GOTO 100+4*(YES=1 OR YES=4)
```

Take the following lines that detect the joystick:

```
S=STICK(0)
IF S=07 THEN X=X+1
IF S=11 THEN X=X-1
```

You can change this to:

```
S=STICK(0)
X=X+(S=7)-(S=11)
```

Here's how it goes: If S=7 then X=X+1-0. If S=11 then X=X+0-1. If X<>7 or 11 then X=X+0-0.

You can even put the boundaries in the same formula, ie:

```
X=X+((S=7) AND (X<MAX))-((S=11) AND (X>MIN))
```

The full formulas and all 8 directions can be found with these 2 formulas:

```
X=X+(S=5 OR S=6 OR S=7)-(S=9 OR S=10 OR S=11)

Y=Y+(S=5 OR S=9 OR S=13)-(S=6 OR S=10 OR S=14)
```

I'm sure you can put your own boundaries into the formulas. You don't want me to do everything do you? OK, I'll give you a clue if you don't know. You've got to put both X and Y MINs/MAXs in both formulas...

APPENDIX C6:

LOGIC:
Anything to do with logic, that's what this appendice is about! Try the following formula exactly as it's shown, on your Atari:

```
? 2+3*2
```

What answer did you get? 10 or 8? This proves that the Atari computes all its LOGIC in a particular order. It doesn't necessarily work from left to right!
The actual order of precedence is as follows:

powers	powers are done first
divide	next is divisions
multiply	then multiples
minus	onto subtractions
plus	and lastly, additions

```
? 5+7*9/8-3
```

If you work from left to right in this formula, then you'll get 10.5, but this ain't how it works is it! The real answer is obtained by dividing 9 by 8, multiplying by 7, subtracting 3 and adding 5, which gives you 9.875.

Another feature that precedes all of these factors, is brackets. When bracketing particular segments of the formula, this is calculated first, for example:

```
? (5+7)*9/8-3
```

This gives:

```
  (12) * 9 / 8 - 3
   12  * 1.125 - 3
      13.5     - 3  = 10.5
```

Mathematical functions take the highest order of all formulas, and require brackets as an essential part of their syntax, ie:

```
? 7+COS(9-3)/5*2
```

This gives:

```
  7+INT(COS( 6 ) * 10 ) / 5  *  2
  7+INT( 9.945218954  ) / 5  *  2
  7+         9          / 5  *  2

        ...the rest is as before
```

Mathematical functions, and indeed all other functions can be used in many ways. For instance:

```
SIN(COS(5)*ATN(1))
```

This is a perfectly feasible syntax. If an error occurs, it is due to the values, divide by zero or out of range. You can even substitute standard functions into mathematic expressions, for example:

```
ASC("Z")*SIN(x)
```

This will multiply the sine of variable "x" by the ASCII code of the letter Z. Consider for argument purposes, that you have DIMensioned A$ and it contains the string "SUE19DOB290294". Here are some other expressions that are of perfectly evaluable syntax:

```
2*VAL(A$(4))           - Multiplies 19 by 2

ASC(CHR$(A$(1,1)))+1   - finds ASCII code of S and adds 1

PEEK(SGN(PEEK(88)))    - finds contents of location 1 if
                         location 88 contains a positive
                         number, location 0 otherwise.

CHR$(VAL(A$(9,10)))    - returns the character whose ASCII
                         code is 29.
```

The context of Basics functions is quite unlimited, so long as they abide by a syntax law. For instance; CHR$ expects a value in its argument which it treats as an ASCII code of the character it returns. ASCII is the opposite of CHR$, so ASCII expects the argument to address a character, either via the use of the CHR$ command, or the use of inverted commas.

On a similar line to functions, is the logical operators; AND, OR and NOT. These can be substituted in the above examples as well, so long as they use the correct syntax, ofcourse.

These operators can be used in 2 different ways, depending solely upon whether you are using Basic or machine-language. The machine-language way is described later on.

The 1st 2 operators, AND and OR take the format:

argument1 operator argument2

The result of the operation is dependent upon the 2 arguments as in the following truth table:

Input:		Output:	
arg1	arg2	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

So, with the AND operator, if both arguments are positive then the result is 1. Otherwise the result is 0. With OR, the result is 0 only when both arguments are negative.

The NOT operator is just an inversion of the argument. If the argument is positive, then the result is 0, if the argument is negative, then the result is 1.

In machine-language, there isn't a NOT instruction, but there is EOR. AND, OR (ORA) and EOR actually affect the binary bits of a number.

The AND instruction is widely used to turn particular bits on or off, for instance:

```
AND #$F0
```

This instruction will turn off all the low 4-bits in the byte, leaving the high 4-bits unchanged. This is a handy technique for ensuring that any colour going to the screen is at its darkest shade. If used along with:

```
ORA #$08
```

it will set bit-3 of the byte, and leave all other bits unaffected, thus, all colour output would be at luminance 8.

```
EOR #$80
```

This is a widely used technique which will simply inverse a byte.

The EOR truth table is:

Input:		Output:
arg1	arg2	EOR
0	0	0
0	1	1
1	0	1
1	1	0

You'll only get a 1, when both input arguments are alternate.

Another use of the EOR instruction, is to alternate between a blank screen and an image being put there. Very handy for showing an image, blanking it out, moving it and re-placing it. For example:

```
LDA SCREENBYTE
EOR DATABYTE
STA SCREENBYTE
```

APPENDIX C6:

The 1st time through the loop would select just the bits from the databyte, but the 2nd time through the loop will return a blank byte.

You can also simulate the Basic NOT command with:

EOR #$F

The logical operators can also go much further than these simple 3 described. You can nest an AND with a NOT to achieve what is known as a NAND. An OR and a NOT achieve a NOR. In fact, you can create your own special truth tables to achieve whatever program you want using just Basic Formulas!

APPENDICE C7:

ERROR CODES.

This appendice contains very many of the error codes you're likely to come across within the Atari personal computers with a little description alongside each one. This list begins with the Basic language error codes:

BASIC ERRORS:

ERR. CODE dec	hex	ERR. NAME
2	2	OUT OF MEMORY There is not enough RAM available for the process the Atari is trying to carry out, or there are too many nested FOR/NEXT loops or subroutines.
3	3	VALUE ERROR The numeric value is either too great, too small or of the wrong sign (negative when it should be positive).
4	4	TOO MANY VARIABLES A standard Basic program is limited to 128 different variable names (256 in TURBO Basic). Variables previously used, but presently deleted still affect variable counts, so to overcome this problem, LIST your Basic program to disk, coldstart the computer and re-ENTER the program.
5	5	STRING LENGTH ERROR The element or cell being addressed is past the end of the strings or arrays DIMension.

6	6	OUT OF DATA The most recent READ statement was trying to obtain an element of data past the end of all DATA elements. You should use RESTORE to point to the DATA line that you wish to READ.
7	7	NUMERIC/LINE ERROR The numeric value is negative, or greater than 32767 in a situation where it is not allowed, such like a line number.
8	8	INPUT STATEMENT ERROR An attempt to input a string value into a numeric variable was made
9	9	ARRAY/STRING DIMENSION ERROR The string in use is unDIMensioned, or an already existing string has tried to be re-DIMensioned.
10	A	ARGUMENT STACK OVERFLOW An expression is too large, or there is too much nesting of GOSUBs or FOR/NEXT loops.
11	B	FLOATING POINT OVERFLOW/UNDERFLOW ERROR A number is greater than the magnitude 9.99999999 * 10E-97 (97 digits after the decimal point).
12	C	LINE NOT FOUND A GOSUB, GOTO or IF-THEN statement tried to reference a non-existent line number.
13	D	NEXT WITHOUT FOR A NEXT statement with no existing FOR has been encountered. Perhaps a POP statement has taken its address off the stack.
14	E	LINE TOO LONG The line entered is greater than 3 logical lines (120 bytes). The end of a program line is denoted by a BEEP sound.
15	F	GOSUB OR FOR LINE DELETED A RETURN or NEXT statement can no longer find its relation, GOSUB or FOR.
16	10	RETURN WITHOUT GOSUB There is no existing GOSUB for the recently encountered RETURN statement to react to.
17	11	GARBAGE ERROR A previously executable line is no longer of any sense. Perhaps due to POKEing in the wrong area of memory, or a machine-code routine crashing the Basic program.

18	12	INVALID STRING CHARACTER A non-numeric string was trying to be converted to a numeric value using the VAL function.
19	13	LOAD PROGRAM TOO LONG Not enough RAM for the program trying to load.
20	14	DEVICE NUMBER ERROR A device number less than 0 or greater than 7 was used.
21	15	LOAD FILE ERROR The command being used to load a file is not the companion to which it was saved with. LIST goes with ENTER, CSAVE goes with CLOAD and SAVE goes with LOAD.
128	80	BREAK KEY ABORT The BREAK key was pressed during an I/O operation.
129	81	IOCB CHANNEL ALREADY OPEN You are trying to OPEN a channel that is already OPEN.
130	82	NONEXISTENT DEVICE Your program is trying to use a non-existent device.
131	83	IOCB OUTPUT ONLY ERROR An attempt to read from a file which is only OPENed for write was done.
132	84	INVALID COMMAND An illegal command has been used in an I/O operation such as XIO.
133	85	CHANNEL NOT OPEN An I/O operation tried to use a channel which has not been OPENed.
134	86	BAD IOCB CHANNEL NUMBER A channel outside the range 0 - 7 was referenced.
135	87	IOCB INPUT ONLY ERROR An Attempt to write to a file which is only OPENed for read was done.
136	88	END OF FILE ERROR Either the EOF record has been reached, or the CTRL+"3" key was pressed.
137	89	TRUNCATED RECORD A data record greater than the INPUT command can accomodate has been read, thus, truncating the record. INPUT must find an EOL character at a maximum of 120 bytes apart.

APPENDIX C7:

138	8A	DEVICE TIMEOUT The specified device has not responded in a particular amount of time, given by location 774.
139	8B	DEVICE NAK The device cannot carry out the command asked of it.
140	8C	SERIAL BUS FRAME ERROR Serial bus data inconsistency. The device may be faulty.
141	8D	CURSOR OUT OF RANGE The cursor is trying to access a co-ordinate outside the range offered by the Graphics mode in use. See the ROWS and COLUMNS in the chart on page-16.
142	8E	SERIAL BUS DATA FRAME OVER-RUN Serial bus data inconsistency. The device may be faulty, or perhaps even the I/O lead itself.
143	8F	SERIAL BUS DATA FRAME CHECKSUM ERROR The data being transferred is corrupted.
144	90	DEVICE DONE ERROR The disk is either write-protected, or the disk directory is scrambled.
145	91	BAD SCREEN MODE HANDLER There is either a problem with the screen handler, or the disk drive detected a difference between what it wrote compared to what it was supposed to write.
146	92	FUNCTION NOT IMPLEMENTED An unallowable action was attempted, such-like: outputting to the keyboard, or inputting from the printer etc..
147	93	INSUFFICIENT RAM Not enough memory to perform the task the Atari has been appointed, such like changing from Graphics 0 to Graphics 8 with only a few bytes of memory spare.
150	96	SERIAL PORT ALREADY OPEN Each serial port can be OPEN to only 1 channel simultaneously.
151	97	CONCURRENT MODE ERROR A serial port must be OPENed for concurrent mode BEFORE enabling current mode I/O with the XIO 40 command.

152	98	CONCURRENT MODE BUFFER ERROR An inconsistent buffer length and address during the startup of concurrent I/O using the optional program-provided buffer feature.
153	99	CONCURRENT MODE ACTIVE An I/O on a serial port was attempted, while another serial port was OPEN and active in concurrent mode.
154	9A	CONCURRENT MODE INACTIVE The I/O attempted through the serial port requires the concurrent mode.
160	A0	DRIVE NUMBER ERROR A drive number outside the range 1 - 8 was used.
161	A1	TOO MANY OPEN FILES Normally, only 3 disk files can be OPEN at one time. See location 1801.
162	A2	DISK FULL The disk is full, to the last sector!
163	A3	UNRECOVERABLE SYSTEM ERROR During I/O, an unknown error occured which cannot be determined or recovered from.
164	A4	FILE NUMBER MISMATCH The sector POINTed to is not within the file OPENed, or the disk-file' sector-link bytes are scrambled (the last 3 bytes of every DOS sector).
165	A5	FILE NAME ERROR The file-name is illegal, see locations 3783, 3818 and 3822.
166	A6	POINT DATA LENGTH ERROR You are POINTing to a byte in a sector which doesn't exist. There are normally 128 bytes in a sector, but 256 in true double density.
167	A7	FILE LOCKED A locked file was accessed for alteration. You should unlock the file first.
168	A8	INVALID DEVICE COMMAND A non-existent XIO command was attempted, or is not defined for the device in use.
169	A9	DIRECTORY FULL The disk directory allows up to 64 files only. With SPARTADOS, you can create sub-directories which is an excellent feature brought down from grandfather programs such as MS-DOS on the IBM.

APPENDIX C7:

170	AA	FILE NOT FOUND The specified file-name is not on the disk directory.
171	AB	POINT INVALID Incorrect use of the POINT command; an attempt was made to use POINT with an incorrectly OPENed file.

STATUS ERRORS:

ERR. CODE dec	hex	ERR. DESCRIPTION:
1	1	Operation complete and OK
3	3	EOF approaching; next read gets the last data in the file.

These are the only differences in errors, all others including those given on page-83 are the same as the Basic error codes listed previous.

DOS 3 ERRORS:

Also among the errors is probably the worst list of all! Those of DOS 3, why on earth did Atari change everything (including the error codes!) when it was quite fine in the beginning!? Here are the alterations only returned by DOS 3. In my opinion, and possibly another few thousand others, you should convert all your DOS 3 files to DOS 2.5, or use an even better DOS again such as SUPERDOS V.5 or SPARTADOS.

ERR. CODE dec	hex	ERR. NAME
		Errors 2 - 10 are the same as DOS 2.X, except when using the DOS 3 menu functions; they are then used as follows:
2	2	NO COMMAND No file with an extender .CMD exists in drive-1.
3	3	INPUT REQUIRED You've given a blank character in the Rename function which is not allowed.
4	4	NO CARTRIDGE You tried executing the TO-CARTRIDGE function when one doesn't exist.
5	5	I/O ERROR Any I/O error, ie. printer is not on-line.

6	6	INVALID END ADDRESS The END address, given in the Save function is lower than the START address.
7	7	MEM.SAV LOAD ERROR The system is unable to restore the memory using the MEM.SAV file. The program that you had in memory is now lost, told you DOS 3 was a waste of time didn't I!
8	8	MEM.SAV SAVE ERROR Something has happened while the system was trying to write the MEM.SAV file. Try changing the disk that your writing to.
9	9	DRIVE INPUT ERROR Invalid device specification supplied.
10	A	FILENAME INPUT ERROR Invalid filename supplied.

Here's a few additional errors included with DOS 3, not on any other DOS:

174	AE	DUPLICATE FILENAME You are trying to Rename a file to a name of a file that already exists.
175	AF	BAD LOAD FILE The file you are trying to load is not a load-type file.
176	B0	INCOMPATIBLE FORMAT You are trying to perform a DOS 3 function with a DOS 2.0 disk. Your halfway there!
177	B1	DISK STRUCTURE DAMAGED DOS 3 does not recognize the files on the disk due to damage (well done).

The sooner you get rid of DOS 3, the better, because DOS 3 is not only incompatible with DOS 2, 2.5, SPARTADOS, SUPERDOS etc., but it saves in a format that can easily waste 'chunks' of memory, literally! Send off to Atari for your replacement.

APPENDIX C8:

TRIGONOMETRIC FORMULAS:

This list is, by far, not complete, but does provide some more commonly used trigonometric formulas. Some values of "x" invalidate some functions, such as COS(x)=0 then SEC(x) is not real. Make sure you check for these:

```
ARCCOS(x)=-ATN(x/SQR(-x*x+1))+1.5707633
  Returns the inverse cosine of x(ABS(x)<1)

ARCCOSH(x)=LOG(x+SQR(x*x-1))
  Returns the inverse hyperbolic cosine of x (x>=1)

ARCCOT(x)=-ATN(x)+1.5707633
  Returns the inverse cotangent of x

ARCCOTH(x)=LOG((x+1)/(x-1))/2
  Returns the inverse hyperbolic cotangent of x(ABS(x>1)

ARCCSC(x)=ATN(1/SQR(x*x-1))+(SGN(x)-1)*1.5707633
  Returns the inverse cosecant of x(ABS(x)>1)

ARCCSCH(x)=LOG((SGN(x)*SQR(x*x+1)+1)/x)
  Returns the inverse hyperbolic of x(x>0)

ARCSEC(x)=ATN((SQR(x*x-1)+(SGN(x)-1)*1.5707633
  Returns the inverse secant of x(ABS(x)>=1)

ARCSECH(x)=LOG((SQR(-x*x+1)+1)/x)
  Returns the inverse hyperbolic secant of x(0<x<=1)

ARCSIN(x)=ATN(x/SQR(-x*x+1))
  Returns the inverse sine of x(ABS(x)<1)

ARCSINH(x)=LOG(x+SQR(x*x+1))
  Returns the inverse hyperbolic sine of x

ARCTANH(x)=LOG((1+x)/(1-x))/2
  Returns the inverse hyperbolic tangent of x(ABS(x)<1)

COSH(x)=(EXP(x)+EXP(-x))/2
  Returns the hyperbolic cosine of x
```

COT(x)=COS(x)/SIN(x)
  Returns the cotangent of x(x<>0)

COTH(x)=EXP(-x)/(EXP(x)-EXP(-x))*2+1
  Returns the hyperbolic cotangent of x(x<>0)

CSC(x)=1/SIN(x)
  Returns the cosecant of x(x<>0)

CSCH(x)=2/(EXP(x)-EXP(-x))
  Returns the hyperbolic cosecant of x(x<>0)

LOGa(x)=LOG(x)/LOG(a)
  Returns the base a logarithm of x(a>0,x>0)

LOG10(x)=LOG(x)/2.30258509
  Returns the common (base ten) logarithm of x(x>0)

MODa(x)=INT((x/a-INT(x/a))*a+0.05)*SGN(x/a)
  Returns x modulous a: the remainder after division
  of x by a(a<>0)

SEC(x)=1/COS(x)
  Returns the secant of x(x<>pye/2)

SECH(x)=2/(EXP(x)+EXP(-x))
  Returns the hyperbolic secant of x

SINH(x)=(EXP(x)-EXP(-x))/2
  Returns the hyperbolic sine of x

TAN(x)=SIN(x)/COS(x)
  Returns the tangent of x(x[]0)

TANH(x)=-EXP(-x)/EXP(x)+EXP(-x))*2+1
  Returns the hyperbolic tangent of x

Note: pye instead of its symbol and the base elements "a" and "10" on the 2 LOG formulas, where they should be entered as base expressions.

APPENDIX C9:

DISPLAY MODES.

When you call a GRAPHICS mode from BASIC you can normally access it in 1 of 2 different ways, choosing either a whole graphics screen, or a graphics screen with a text window at the bottom. The 4 tables on the next sheet show you the exact memory configurations for both these combinations. Of course, you can always add 32 to your GRAPHICS mode value to access the mode without clearing the screen, but in addition to this it is also possible to obtain an invisible text window. You do this by calling the GRAPHICS mode you want, adding 16 so that you obtain a full graphics screen and then include a POKE 703,4 to enable the text window. This way, the 160 bytes that are normally unused in the full-screen mode would be taken by the text-window, but since you called the mode WITHOUT a text window, there is no Display List supplied to display the memory you type in, whether it be in-screen or below! This technique also works in GRAPHICS 0, but the text window then occupies the real bottom 4 lines of the rest of the screen.

It is also possible to achieve a visible text window in GTIA modes 9, 10 and 11. You do this exactly the same way as you would call a GTIA mode in machine-code by calling GRAPHICS 8, POKEing 87 with 9 and POKEing 623 with either 64, 128 or 192 depending on whether you wanted GTIA 9, 10 or 11, respectively. This way, the memory configuration would then take the same format as GRAPHICS 8 with a text window.

Oh dear, the text window is unreadable. What a shame... Well, thats just another problem to overcome isn't it! There are a few ways, one of which uses a short DLI on the very last scan-line of the graphics area (immediately above the text window). See the end of Appendice C11 for this program.

In addition to the 16 modes given, there is also a graphics mode usually referred to as GRAPHICS 0.5; ANTIC code 3. It isn't accessible with the standard BASIC statement so you need to create your own Display List (DL). Try the following program:

```
10 GRAPHICS 0
20 DL=PEEK(560)+256*PEEK(561)
30 POKE DL+3,64+3
40 FOR I=6 TO 23
50 POKE DL+I,3:NEXT I
60 POKE DL+24,16:POKE DL+25,65
70 POKE DL+26,PEEK(560)
80 POKE DL+27,PEEK(561)
```

APPENDIX C9:

This modes memory configuration is as follows:

```
|----------|----------|
|28bytes DL|28bytes DL|
|----------|----------|
|  4bytes  |  4bytes  |
|  unused  |  unused  |
|----------|----------|
|          |          |
| 760bytes | 760bytes |
| char.map | char.map |
|          |          |
|----------|----------|
| 40 bytes | 40 bytes |
|  unused  |  unused  |
|----------|----------|
| 160bytes | 160bytes |
|textwindow|  unused  |
|----------|----------|
```

Note, that to obtain the text window you must POKE 703 with 4. The only snag is that the text window is invisible (off screen), but that's no problem! If you want the text window on-screen then add the following lines to the previous program:

```
22 DM=PEEK(DL+4)+256*PEEK(DL+5)
24 DM=DM+(5*40)
26 HI=DM/256:LO=DM-HI*256
28 POKE DL+4,LO:POKE DL+5,HI
```

This does, however, change the memory configuration to:

```
|----------|----------|
|28bytes DL|28bytes DL|
|----------|----------|
|  4bytes  |  4bytes  |
|  unused  |  unused  |
|----------|----------|
| 200bytes |          |
|  unused  | 200bytes |
|----------|  unused  |
|          |          |
| 600bytes |          |
| char.map |----------|
|          |          |
|----------| 760bytes |
| 160bytes | char.map |
|textwindow|          |
|----------|----------|
```

APPENDIX C9:

You may find the tables a little peculiar at first, but they are correct. This mode only allows 19 lines to be on-screen at once since the mode-byte is now 10 scan-lines deep.

You may wonder why the standard text screen has a mode-byte of this configuration. Well, the main reason this mode is used is so that you can achieve 'true descenders' in text, where any non-capital text can droop below the base level of capital text. Also, in addition to this the international character-set with the phonetic symbols can be fully exploited. There is a program in appendice G4 which redefines the character-set to achieve full power of this mode.

Another use for this mode would be for enlarging characters twice over by substituting this Antic code in the example program at the top of page-162 in the map. Anyway, if all thats been offered doesn't satisfy you, then you can always create your own. See locations 560 and 561, also the BOUNDARIES appendice. Here's a split-screen variant:

```
10 GRAPHICS 15+16
12 DL=PEEK(560)+256*PEEK(561)
14 RT=PEEK(106)*256
20 W2=RT-80:W1=W2-80:U2=W1-776
22 S2=U2-3160:S1=S2-3160
24 U1=DL+178
26 FOR I=0 TO 7
28 READ D
30 POKE DL+84+I,D:POKE DL+170+I,D
32 NEXT I
34 DATA 66,0,0,0,2,78,0,0
36 POKE DL+175,65
38 FOR I=0 TO 77
40 POKE DL+6+I,14:POKE DL+92+I,14
42 NEXT I
50 H=INT(S1/256):L=S1-H*256
52 POKE DL+4,L:POKE DL+5,H
54 POKE 88,L:POKE 89,H
56 H=INT(S2/256):L=S2-H*256
58 POKE DL+90,L:POKE DL+91,H
60 H=INT(W1/256):L=W1-H*256
62 POKE DL+85,L:POKE DL+86,H
64 H=INT(W2/256):L=W2-H*256
66 POKE DL+171,L:POKE DL+172,H
68 H=INT(DL/256):L=DL-H*256
70 POKE DL+176,L:POKE DL+177,H
72 POKE 703,4
80 STOP
```

The program is quite large for a DL change, the reason being that it gives you all of the memory pointers that you could need. The DL takes 178 bytes, the 4 mode-0 lines still act as the text window, but in 2 halves.

APPENDIX C9:

There are 2 unused areas of memory beginning at U1 and U2, the sizes of which are 704 bytes and 776 bytes, respectively. To draw in the top half of the screen, the vertical co-ordinates are 0 to 78.
The 2nd screen is co-ordinates 79 to 157. Should you want to load information from disk into the 2 areas, then area 1 begins at S1 and area 2 begins at S2. The memory configuration is as follows:

```
|-----------|
|178bytesDL |
|-----------|
| 704bytes  |
|  unused   |
|-----------|
|           |
|6320 bytes |
|  bitmap   |
|           |
|-----------|
| 776bytes  |
|  unused   |
|-----------|
| 160bytes  |
|textwindow |
|-----------|
```

You should also note that every table in this appendice has been calculated from the DL address to RAMTOP. Where RAMTOP is always the next byte above the text window memory.

If your unsure about the split-screen DL program pre-leafed, then consult the DISPLAY LISTS appendice and locations 560 and 561.

COLOURS PER MODE.
Lastly, to complete this appendice here is a table showing you how many colours are allowed standard in each mode.

MODE:	COLOURS:	REGISTERS:
0	2 1/2	709 - 710 and 712
1	5	708 - 712
2	5	Same as mode-1
3	4	708 - 710 and 712
4	2	708 and 712
5	4	Same as mode-3
6	2	" " " 4
7	4	" " " 3
8	2 1/2	" " " 0
9	1	712
10	9	704 - 712
11	16	712
12	5	Same as mode-1
13	5	" " " 1
14	2	" " " 4
15	4	Same as mode-3

It's all fairly straightforward; modes 0 and 8 can have a border colour and a background colour, but the foreground plotting colour will be a luminance of the background colour. Should you overlay PMGs, then the foreground colour under the PMG overlap will become a luminance of 'the PMG colour. Modes 1, 2, 12 and 13 have 5 colours. Modes 1 and 2 are what I call byte-handicapped, meaning that the whole of the byte can only be 1 of 4 colours. You print text in these colours by choosing capitals, non-capitals or inverse combinations. The 5th colour is background. Modes 12 and 13 are a bit different and are not byte-handicapped. They can have up to 5 colours spread throughout 1 character. For more information on these modes see the GRAPHICS 12/13 appendice. Modes 3, 5, 7 and 15 are 4 colour modes, each colour accessed by use of the COLOR statement. Modes 4, 6 and 14 are 2 colour modes. COLOR 1 being the plotting colour and COLOR 0 being the background colour.

The GTIA modes 9, 10 and 11 are different again. Mode 9 can have up to 16 shades by use of the COLOR statement. The shades are the luminance of the background colour in 712. Mode 10 can achieve 9 colours, 704 being the background colour. Mode 11 is the opposite to mode 9, where 16 colours can be accessed from the background shade given in 712.
Although these are all the standard colour configurations, it is also possible to excel on this again. DLIs offer additional colours on the screen, and in fact, you can also achieve 256 colours in the GTIA modes on screen at once. The photos on the ATARI Corporation demonstration disk proves this. But, there is also another technique to gain extra colour which is similar to artifacting in mode 8, which you can perform on mode 15. One way of doing this is by converting your GRAPHICS 15 displays into GRAPHICS 12 by the use of a program called LOGOS CREATOR by THORGAL. This will allow you to have 1 extra colour in your pictures. The ATARI Artist cartridge shows another way by plotting a pixel of 1 colour exactly opposite a pixel of another colour, whilst alternating them vertically. Whilst this technique does work it doesn't look that good. A better way would be as in the program below:

```
10 GRAPHICS 15+16
12 POKE 708,52:POKE 709,132
14 POKE 710,212
16 FOR X=10 TO 90 STEP 10
18 READ C1,C2
20 FOR Q=0 TO 19 STEP 2
22 COLOR C1
24 PLOT X,50+Q:DRAWTO X+8,50+Q
26 COLOR C2
28 PLOT X,50+Q+1:DRAWTO X+8,50+Q+1
30 NEXT Q
32 NEXT X
34 GOTO 34
36 DATA 1,1,2,2,3,3
38 DATA 1,2,2,3,1,3
40 DATA 2,1,3,2,3,1
```

It's possible to have 10+ colours in mode 15, think about it!

0	0+16	1	1+16	2	2+16	3	3+16
32 bytes DL	32 bytes DL	34 bytes DL	32 bytes DL	24 bytes DL	20 bytes DL	34 bytes DL	32 bytes DL
960 bytes char.map	960 bytes char.map	400 bytes char.map	480 bytes char.map	200 bytes char.map	240 bytes char.map	200 bytes bitmap	240 bytes bitmap
		80 bytes unused		40 bytes unused		40 bytes unused	
		160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused

4	4+16	5	5+16	6	6+16	7	7+16
54 bytes DL	56 bytes DL	54 bytes DL	56 bytes DL	94 bytes DL	104 bytesDL	94 bytes DL	104 bytesDL
						96 bytes unused	96 bytes unused
400 bytes bitmap	480 bytes bitmap	800 bytes bitmap	960 bytes bitmap	1600 bytes bitmap	1920 bytes bitmap	3200 bytes bitmap	3840 bytes bitmap
80 bytes unused		160 bytes unused		320 bytes unused		640 bytes unused	
160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused

8	8+16	9	9+16	10	10+16	11	11+16
176bytesDL	202bytesDL	202bytesDL	202bytesDL	202bytesDL	202bytesDL	202bytesDL	202bytesDL
80 bytes unused	80 bytes unused	80 bytes unused	80 bytes unused	80 bytes unused	80 bytes unused	80 bytes unused	80 bytes unused
6400 bytes bitmap	7680 bytes bitmap	7680 bytes bitmap	7680 bytes bitmap	7680 bytes bitmap	7680 bytes bitmap	7680 bytes bitmap	7680 bytes bitmap
1280 bytes unused							
160 bytes text window	160 bytes unused	160 bytes unused	160 bytes unused	160 bytes unused	160 bytes unused	160 bytes unused	160 bytes unused

12	12+16	13	13+16	14	14+16	15	15+16
34 bytes DL	32 bytes DL	24 bytes DL	20 bytes DL	174 bytes DL	200 bytesDL	176 bytes DL	202 bytesDL
				96 bytes unused	96 bytes unused	80 bytes unused	80 bytes unused
800 bytes char.map	960 bytes char.map	400 bytes char.map	480 bytes char.map	3200 bytes bitmap	3840 bytes bitmap	6400 bytes bitmap	7680 bytes bitmap
160 bytes unused		80 bytes unused		640 bytes unused		1280 bytes unused	
160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused	160 bytes text window	160 bytes unused

APPENDIX C10:

PLAYER/MISSILE GRAPHICS:

Here's an easy reference table for all PMG locations, and a map of PMBASE organisation:

(W)	53248	- 53251	HPOSP0	- P3
	53252	- 53255	HPOSM0	- M3
	53256	- 53259	SIZEP0	- P3
	53260		SIZEM0	- M3
	53261	- 53264	GRAPHP0	- P3
	53265		GRAPHM0	- M3
	53266 (704)	- 53269 (707)	COLPM0	- PM3
	53275 (623)		PRIOR (GPRIOR)	
	53276		VDELAY	
	53277		GRACTL	
	53278		HITCLR	
	54272 (559)		DMACTL (SDMCTL)	
	54279		PMBASE	
(R)	53248	- 53251	M0PF	- M3PF
	53252	- 53255	P0PF	- P3PF
	53256	- 53259	M0PL	- M3PL
	53260	- 53263	P0PL	- P3PL

```
Double-line     0 ****PMBASE**** 0     Single-line
resolution        *            *       resolution
bytes             *   unused   *       bytes
offset:           *            *       offset:
             +384 *------------* +768
                  *            *
                  *  missiles  *
                  *   #0 - 3   *
                  *            *
             +512 *------------* +1024
                  *            *
                  *  player#0  *
                  *            *
             +640 *------------* +1280
                  *            *
                  *  player#1  *
                  *            *
             +768 *------------* +1536
                  *            *
                  *  player#2  *
                  *            *
             +896 *------------* +1792
                  *            *
                  *  player#3  *
                  *            *
        1K  +1024 ************** +2048   2K
```

APPENDIX C10:

GRAPHICS IN YOUR OWN PROGRAMS.

Should you be including PMGs in your own programs, then you should perform the following steps. If you do not wish to use PMBASE for full blown shapes, then you should ignore steps 2 and 4:

1. CALL YOUR PLAYFIELD:
   A simple graphic call will suffice

2. ENABLE P/M DMA AND RESOLUTION:
   see DMACTL

3. DETERMINE GRAPHIC SHAPE:
   see PMBASE for full-blown shapes, otherwise see GRAPHs

4. ENABLE DMA TO SCREEN:
   see GRACTL

5. DETERMINE P/M COLOURS AND SIZES:
   see COLs and SIZEs

6. DETERMINE HORIZONTAL POSITIONS:
   see HPOSs

VERTICAL MOVEMENT:
Vertical player missile movement is usually only of an acceptable speed if you use machine-code, however, here's the Basic answer:

```
100 REM TOMO' Vertical PMG movement
110 REM using strings
120 REM April '92
130 REM
140 DIM PM$(256),I$(24)
150 ADDR=(9*4096)/256
160 POKE 140,0:POKE 141,ADDR+2
170 POKE 559,42:POKE 53248,100
180 POKE 704,253:POKE 53256,1
190 FOR C=1 TO 24
200 READ D:I$(C,C)=CHR$(D):NEXT C
210 DATA 0,255,48,48,56,0,0,255,129
220 DATA 193,255,0,0,255,161,177
230 DATA 129,0,0,255,129,193,255,0
240 POKE 54279,ADDR:POKE 53277,2
250 FOR J=1 TO 120
260 PM$(J,J+24)=I$:NEXT I
270 FOR J=120 TO 1 STEP -1
280 PM$(J,J+24)=I$:NEXT I
290 GOTO 250
```

There you have it. Enjoy yourself!

APPENDIX C11:

DISPLAY LIST INTERRUPTS.
Well now, let me see. If you look down at locations 512 and 513, you may notice that I hadn't intended to give full DLI details in this book. In fact, I was only going to give you some solutions to overcoming problems using DLIs. But, I've changed my mind, and have included my tutorial on DLIs. I'll still be including what I originally was going to put here as well, so expect a lot of reading!

Just as a means of reference, the DLI is an NMI interrupt processed by the ANTIC chip. They are user created, and their purpose is to gain the full potential of any feature of the hardware. The hardware being whatever the computer can do! As an example, you could achieve 128 colours in Graphics 15, create a screen of scrolly stars out of a Player/Missile Graphic, even turn 4 PMGs into 8, the list goes on...

But, as you would imagine. To gain such power at your fingertips, you'll need to sacrifice a lot of grey-matter to understanding them. You would normally need to understand Machine-code, but having cut a few corners, the average Basic programmer should be able to make their own DLIs after reading this tutorial/appendice (fingers crossed).

We'll kick-off with the Televisions Raster scan. Take a look at figure-1;

```
a F----------------------- b
c ------------------------ d
e ------------------------ f
g ------------------------ h
i ------------------------ j
k ------------------------ l
m ------------------------ n
o ------------------------ p
q ------------------------ r
```

Consider the "F" is the pulse of an electron-beam. This "F" (Fred from now on) travels from a to b, in reaching b it switches off, but, continues travelling to c where it then switches back on. Fred continues this type of journey until it reaches the bottom right-hand corner of the tube, alias r. Upon reaching r, Fred turns off and returns directly to a to continue the journey indefinitely. Each horizontal journey, from a to b, c to d etc. is called a scan-line. Each journey from b to c, d to e etc. is referred to as a horizontal blank.
The journey from r to a is called the vertical blank; for an explanation of this time-period, see the relating appendice. In real,ity, the European TV (PAL) has 312 scan-lines, all of which are called a frame, and there are 50 frames drawn every second, hence, the mains power of 50Hz.

APPENDIX C11:

The DLI is a Machine-code interrupt routine that executes during a horizontal blank. But, although there are 311 horizontal blanks, you don't actually get to use all of these. It depends on which Graphics mode you are using. If you were using Graphics 0, then you could only achieve 24 on-screen DLIs since there are only 24 mode lines which can set the DLI to occur! Another limitation is the amount of time each DLI has to execute. Under normal circumstances, you would only be allowed 34 Machine-cycles of time, which is approximately 10 machine-code instructions. The time does vary, depending on the width of the playfield, controlled with location 559.

Try this, program-1:

```
10 GRAPHICS 0
20 DL=PEEK(560)+256*PEEK(561)
30 J=0
40 READ D:IF D+1 THEN POKE 1536+J,D:J=J+1:GOTO 40
50 DATA 72,169,182,142,24,208,104,64,-1
60 POKE 512,0:POKE 513,6
70 POKE DL+16,PEEK(DL+16)+128
80 POKE 54286,128+64
```

The listing was programmed with 6 steps in mind:

1. Select the Graphics mode
2. Find the address of the Display List (DL)
3. Poke the Machine code interrupt routine (DLI) into a safe
   area of memory. In this case, page-6 (1536; $600)
4. Tell Antic where to find the DLI
5. Tell Antic where you wish the DLI to be executed
6. Add the magic powder; make it work!

OK then, to progress further, then you should be able to understand steps 1, 2 and 3 (Lines 10, 20 and 30 - 50). If you don't, then you can get more details from appendices LSBs and MSBs, MODES and locations 1536 - 1791, 560 - 561 and 512 - 513.

4. For step 4, Antic needs to know where in memory the DLI resides, so to achieve this memory locations 512 and 513 are used as an LSB/MSB vector address;

DLIADDR = PEEK(512)+256*PEEK(561)

Hence, 0+256*6 = 1536, the memory to where we POKE our machine-code routine.

5. Step 5, we need to tell Antic where on the Graphics DL we want the DLI interrupt to execute. We do this by setting bit-7 (decimal 128) to the relevant mode-line. Thus, we just add 128 to the mode-line in line 70.

6. The final step is to add the magic powder. Antic doesn't normally run a DLI, so we have to do this ourselves. Do this by setting bit-7 (decimal 128) at hardware location 54286; $D40E. You should also note, however, that this location is also used to enable a vertical blank interrupt. The VBI uses bit-6 (decimal 64) and you should leave this interrupt enabled for normal Atari working. If you do disable this interrupt, then all of the actions in appendice D1 will be de-activated.

3 other areas to explain are:

a. Hardware and shadow registers
b. Machine-language
c. DLI needs and limitations etc.

a. Hardware and Shadow registers.

A shadow register is a memory location whose contents are transferred to its hardware register during the vertical blank. As an example, the Graphics 0 background colour 'blue' is conrolled with location 710. But, should you POKE location 710 with a different colour, then the background is only changed to this new colour when the contents of location 710 are copied into its hardware location 53272. This is done during the Atari' deferred Vertical Blank Interrupt, and it serves 2 purposes;
The first is to achieve a precise timing in colour change, thus, you see no flicker on the screen. The 2nd is a little more depthy. Since some hardware registers are used for 1 purpose when POKEing them, and a completely different purpose when PEEKing them, there would be no way of finding out the value contained in them for the purpose that you POKE them, hence, the reason for keeping shadow registers!

b. Machine-language.

If you're not familiar with 6502 machine-language, then you might have thought it to be hard to learn, perhaps. In fact, in some ways it is easier to learn than Basic, but I am not taking away any achievement that you will feel when you do understand machine language. The main challenge in grasping the lowest level language of the system, is that of Binary. Once you achieve this, then the rest isn't so hard. Indeed, if I told you just a few details about machine-code, then you would be able to use most of its instruction codes right now! So, why not!??

As your reference, pull out the machine-language appendice D4. If you take a look at the 1st table, you will see all the assembly instructions ADC, AND, ASL etc.. The internal machine codes for each instruction are alongside, under their particular mode. These modes; IMM, ABS etc. are also explained in the appendice.

APPENDIX C11:

The 3 numbers stand for 3 things. The 1st is the machine-code, the 2nd is the amount of time in machine-cycles the instruction takes to process (remember there are 34 cycles per horizontal blank?), the 3rd is the amount of bytes the instruction takes. I think now, you should be able to convert the machine-code from line-50 of the earlier listing to assembly, and maybe even english!

```
 72            48            PHA Implied
169 182        A9 B6         LDA Immediate
141  24 208    8D 18 D0      STA Absolute
104            68            PLA Implied
 64            40            RTI Implied
 -1                          n/a ;data termination code
```

Did you get this? If you didn't, I wanna know why?? Anyway, the assembly instructions are abbreviated english, for example; LDA is LoaD the Accumulator, STA is STore the Accumulator, RTI is ReTurn from Interrupt. In fact, every LD is load, ST is store. PHA and PLA are perhaps more awkward functions to learn, but they mean PusH the Accumulator to the stack and PulL the Accumulator from the stack. The stack being an area of memory that remembers values pushed on top, or pulled from the top. In the event of a PHA, the contents of the Accumulator is pushed on to the top of the stack and in the event of a PLA, the value on the top of the stack is pulled off, just like a stack of cards, where only 1 at a time (always the top one) can be added or removed.

The Accumulator by the way is an internal register, there are 3 in total. The Accumulator, the X-register and the Y-register. They are much like Basic variables except that they can only hold a number between 0 and 255. In machine-code, we only deal with numbers. There is no such thing as 'string-arrays' or 'string-registers' because a character "D" for example is itself treated by its numeric code, whether it be ascii, raw or internal.

Anyhow, getting back to the mainstream. These converted codes could be seen as this:

```
 72           Stack=A
169 182       A=V
141  24 208   POKE M,A
104           A=Stack
 64           End Interrupt routine
```

So, there we have it. The machine-code interrupt routine actually translates loading register A with a number (182) and storing that number at memory location M.

M is derived from using 24 as the LSB and 208 as the MSB, thus, 24+256*208=53272. The background colour Hardware register in Graphics 0.

Right then, you will have noticed there are a few things I jumped past. What are the Stack=A, A=Stack and RTI used for?
When a DLI is executed, the system jumps to your machine-code routine and in doing so, the address to where it came from is placed in the X-register and the Accumulator as LSB and MSB form. But, since your routine may use these registers, you must remember their contents so that they can be replaced before using the RTI instruction to end your routine and ReTurn to normal system control.

Another point you may have noticed, is that the machine-code routine stores the colour in the hardware register itself, and that the top half of the screen remains blue. We store the value directly to the hardware register because this is the actual register that changes the colour, not 710. 710 is simply used by the system as a shadow of the hardware register. Also, the reason for the top half of the screen remaining blue, is because although our DLI changes the colour half way down the screen, the colour is updated from the shadow register during the Vertical Blank Interrupt. You could turn this off if you wanted, ofcourse.

Some additional information you may like to know is for the other assembly instructions. Here's a quick review of some of them:

ADC is Add with carry. AND performs logical AND on the Accumulator with a given byte. All the B?? instructions are Branches. A branch is similar to a format of GOTO like;

1 GOTO (line-1)+byte IF case is true

Where;

```
       169 103        LDA Immediate
        16   1        BPL Relative
no       0            BRK Implied
yes      0            BRK Implied
```

Load the Accumulator with value 103, and Branch if this result is positive, which it is. The destination of the jump is address 'no' plus 1. Branch instructions offer much more than what I've given here, but you really need to get a 6502 machine-language book for a full explanation.

DEC DECrements the given memory location. DEX and DEY decrement the X and Y internal registers. All the T?? instructions are Transfer instructions. For instance, TAX would transfer the contents of A into X.

Going back to the program, if you noticed a glitch in the changing of the colour, then you can find an explanation and solution later in this appendice. For the moment, here's some more DLIs for your fancy.

APPENDIX C11:

Just replace line-50 with whichever of the 3 DLIs you want to see in action, below;

```
50 DATA 72,173,20,0,141,23,208,169,52,141,24,208,104,64,-1

50 DATA 72,173,10,210,141,0,210,169,168,141,1,210,104,64,-1

50 DATA 72,169,33,141,0,212,169,64,141,1,212,104,64,-1
```

MULTIPLE DLIs.
So far, you have only been able to execute 1 DLI per frame, but, it is also possible to have many more DLIs running on the same frame. You could run the 1 DLI more than once by setting the DLI bit on more mode-lines than just the 1. You could also execute many DLIs on alternate frames from the 1 mode-line if you like. But, you can also execute several DLIs all on the same frame. The way in which you do these last 2 techniques is by altering the address contained in the DLI vector 512 and 513 to point to the next DLI from within the previous DLI. Here are 2 demonstration programs:

Program-2; the same DLI more than once:

```
10 GRAPHICS 1
20 DL=PEEK(560)+256*PEEK(561)
30 J=0
40 READ D:IF D+1 THEN POKE 1536+J,D:J=J+1:GOTO 40
50 DATA 72,173,11,212,141,22,208,104,64,-1
60 POKE 512,0:POKE 513,6
70 FOR I=6 TO 24
80 POKE DL+I,PEEK(DL+I)+128
90 ? #6;"THE ONE DLI REPEATED"
92 NEXT I
94 POKE 54286,192
```

Program-3; more than 1 DLI, here's 3:

```
10 GRAPHICS 0
20 DL=PEEK(560)+256*PEEK(561)
30 J=0
40 READ D:IF D+1 THEN POKE 1536+J,D:J=J+1:GOTO 40
50 DATA 72,169,132,141,24,208,169,13,141,0,2,104,64
52 DATA 72,169,52,141,24,208,169,26,141,0,2,104,64
54 DATA 72,169,182,141,24,208,169,0,141,0,2,104,64,-1
60 POKE 512,0:POKE 513,6
70 POKE DL+10,PEEK(DL+10)+128
72 POKE DL+16,PEEK(DL+16)+128
74 POKE DL+22,PEEK(DL+22)+128
80 POKE 54286,192
```

c. DLI NEEDS AND LIMITATIONS.

I did mention earlier, that when a DLI is executed the Accumulator and the X-register are used to hold the return address when exiting your interrupt routine.

Should you need to remember and restore both the Accumulator and the X-register, then you can use the following code:

a. 72	Stack=A	This is the
138	TXA	Remembering
72	Stack=A	routine.
b. 104	A=Stack	This is the
170	TAX	Restoring
104	A=Stack	routine.
64	RTI	

Your routine would begin with part a, then do whatever you want via your own code and then end with part b.

Also mentioned earlier, you are limited to a specific amount of time per DLI which is normally 34 cycles. But, should you be using only 1 DLI, or the DLIs you're using are several mode-lines apart then you CAN increase the size of them considerably!

FIXING A GLITCH.
If you recall program-1, you'll remember that a glitch was visible due to the DLI changing the background colour in plain view. To overcome this, you should ensure that the colour registers are changed 'off-screen', and there are 2 ways of achieving this:

The 1st method is to store the value into the horizontal synchronization register immediately prior to storing the value into the colour register itself. To do this, include the 3 codes; 141 10 212 after loading the Accumulator with the colour value.
This method is very effective, but it does have 1 drawback. To achieve its precise timing, it turns the CPU off and re-powers it exactly 7 cycles before the beginning of the next scan-line, which wastes crucial DLI time that could otherwise be used for something else! The 2nd method overcomes this problem, and you do this by either wasting a little bit of time with the use of the NOP instruction (No-OPeration) or you can process other functions of your DLI while the electron-beam is in a visible-zone (on-screen), whilst performing colour changing after these functions and off-screen. Of course, the other functions must, themselves not effect colour, DL or DM in the present scan-line on the screen.

OVERCOMING DLI PROBLEMS.

In general, the DLI is very cleverly thought of and its interraction with the rest of the system IRQs and VBIs is flawless, but a problem does arise when using the keyboard with activated DLIs.

APPENDIX C11:

Many sources say this is because a STA WSYNC (STA $D40A) occurs during the key-click routine. I wouldn't argue about it, but I think that it is actually because the OS goes into a tight loop in this routine (including a few others) so that the sound given by pressing a key sounds like it does. See the OS source-listing at $F989. In fact, there are 4 tight-loops using the scan-counter ($D40B) in the OS as well as 2 STA WSYNC's. The 2 STA WSYNC's occur in the VBLANK parameter setting and the fine-scroll DLI processed by the deferred VBI at addresses $C279 and $FCCE, which don't affect DLI timing in the slightest!

The 4 tight-loops using the scan-counter are at addresses $F057, $F810, $F822 and $F989. The 1st one is used in the OPEN completion routine (for Graphics), and since OPENing graphics screens is achieved before creating DLIs, this does not affect the smooth running of DLIs. The 2nd, 3rd and 4th tight-loops occur during the screen scroll routines and the key-click routine, and it is these tight-loops that do effect the smooth running of DLIs.

There are a few ways you can overcome these problems. The best and most obvious would be to disable the keyboard, but what if you wanted to take input from the keyboard? Well, why not disable the keyboard prior to the execution of your DLIs, but enable it below your DLIs! There are many ways you can achieve this, you could even disable the keyboard in the Vertical Blank, but enable it in your last DLI. Problem solved. But, what about the screen scroll routines tight-loops?

This is more awkward. You could always turn your ROM OS into a RAM OS and re-write the routines somehow, but, this I'm sure you'll agree could be fairly difficult. There is a way, however, by including a very small routine before your 1st DLI;

```
SYNC LDA  $D40B
     CMP #$Scan-line
     BNE SYNC
     STA  $D40A
```

I first seen this routine in Paul Lays 'Smoother DLIs', Page-6 issue 23. It sure does the trick, but you'll have to work-out the correct scan-line for the interrupt to execute on, and since it turns the CPU off for 1 scan-line, you might have to execute it a little earlier than normal.
So, does this mean that all the problems are solved? Unfortunatly not; because of the nature of the DLI, this now brings about another problem. Also documented on in the article previously described, but appears to be a little inaccurate. Since the DLI now escapes the time of the horizontal blank and becomes a tight-loop itself, any immediately needed register changes may be delayed by conflicting IRQs whose priority is highest.

A solution to this was apparently included in this article as well, but I find that it is unnecessary to go to such lengths when all is needed is to set the Interrupt flag immediately at the beginning of your DLI with SEI, and ending your DLI with CLI.

I/O GLITCHES.

In addition to the above problems, now solved. There is the case of DLIs occasionally active whilst an IRQ loads or saves data from/to a peripheral device such as the disk-drive.

It is actually possible to have DLIs fully operational without having to reconfigure the entire I/O sub-system, and the way in which to achieve this is to sustain them from within an immediate VBI alike the OS VBI achieves its fine-scrolling DLI. You could also use an IRQ to activate your DLIs if you so desired.

Anyway, if you find any information in this appendice inaccurate or would like to discuss the interraction of detailed timing considerations with the IRQs, DLIs and VBIs, then please get in touch with me, because I would like to hear from you. This subject is a very tricky one.

Finally, to end this appendice I leave you with the program to obtain a readable text window in the GTIA modes:

```
10 GRAPHICS 8
12 POKE 87,9
14 POKE 623,64
20 DL=PEEK(560)+256*PEEK(561)
30 FOR I=0 TO 10
34 READ D:POKE 1536+I,D:NEXT I
36 DATA 72,169,0,141,10,212,141,27,208,104,64
40 POKE 512,0:POKE 513,6
50 POKE DL+166,PEEK(DL+166)+128
60 POKE 54286,192
70 FOR I=0 TO 79
72 COLOR I
74 PLOT I,I:DRAWTO I,159
76 PLOT 0,I*2:DRAWTO I,I*2
80 PLOT I,I:DRAWTO 79,I
90 NEXT I
```

APPENDIX C12:

BOOT:

What happends when you boot a disk in the drive? How many sectors load in, and where do they go?
The information you're after is contained in the very 1st 6 bytes of sector 1 on a disk. Take a look at the following table:

BYTE:	DESCRIPTION:
0	Null; unused...
1	Amount of sectors to load
2,3	Load address
4,5	Initiation address

When you turn the computer on with an assumed perfect disk-setup with a disk in the drive etc., the computer calls sector 1 and places it in memory beginning at location 1024. From here, the computer then transfers this 128 bytes to the start address given by bytes 2 and 3. After doing this, it then loads all the sectors, placing them 1 after the other following on from where the 1st sector was transferred to.
Once all the sectors are loaded, the Atari JMPs to the load-address+6, which is the byte immediately following the initiation address bytes. This is the beginning of the users machine-language program.

The initiation address is where the Atari will JMP to when it encounters the next RTS instruction, unless of course you do a JSR, which itself places an address on top of the initiation address. The address below the initiation address is that of Basic, if enabled. Otherwise, it points to the Self-Test entry.

The maximum amount of sectors that you can load in one go is 256, do this by placing 00 in the sectors to load byte. FF is 255. If you want to load more sectors, then you should use your own sector loading routine, which might look something similar to:

```
MORE JSR  $E453     ;get sector
     INC  $30A      ;point to
     BNE XP         ;next sector
     INC  $30B      ;to load
XP   CLC
     LDA  $304      ;next
     ADC #$80       ;128-byte
     STA  $304      ;load area
     LDA  $305      ;for
     ADC #$0        ;next
     STA  $305      ;sector
     DEC COUNT      ;sectors-to-go
     BNE MORE
```

APPENDIX C12:

CASSETTE BOOT:

The cassette boot is identical to the disk boot, except that each sector is now called a record and it is comprised of 132 bytes. The extra bytes are fully explained in CASBUF at 1021.

DOS SECTOR BYTES OVERHEAD:

Sectors on a normal formatted disk offer 128 bytes for use, but on a DOS disk you only have 125. The last 3 bytes on the disk are used as follows:

```
BYTE: USE:

125    Hi-6 bits: file number 0 - 63;
       Lo-2 bits: next sector number in file;
                  (hi-2 bits of byte 126)

126    Next sector in the file link

127    Number of bytes used in the sector; 0 - 125.
```

You should notice that the lowest 2-bits of byte 125 are considered by DOS to be 2 additional bits that append to the high bits of byte 126. This gives a higher range, ie:

```
      ---BYTE 125:--- ---BYTE 126:---
BITS: 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
      ---file#--- ----next-sector----
```

Bit-7 on byte 126 is decimal 128, bit-0 of byte 125 represents decimal 256, and bit-1 represents 512, this means there are 1024 combinations which is the amount of sectors that DOS can access on a 1050 density disk. The next sector in a file link only applies to DOS sectors excluding the directory, VTOC sector and the initial 3 boot sectors on a disk. When this sector-link is 0, there are no more sectors in the file.

See locations 736,737 for an explanation of a binary-file.

APPENDIX C13:

GRAPHICS 12/13.

The character modes in the XL/XE are 0, 0.5, 1, 2, 12 and 13. For a description of altering mode-0 character-sets, then see location 756. Mode 1 and 2 character-sets are just the same as mode-0, see the MODES appendix also. Mode 0.5 is something special, see the MODES appendix and a relevant program in appendix G4. Also see VSCROL location $D405 for a way of mixing different 1/2s of text or graphics modes.

Modes 12 and 13 are unlike all the other text modes. The defining of the shape is done the same, but the result is somewhat different. Try typing the two character redefinition programmes on page-68, but change the Graphics mode in line 20 to 12 or 13.

Take the following redefinition grid:

```
DEC: 128  64  32  16   8   4   2   1  Coloour
BIT    7   6   5   4   3   2   1   0  register

     |-------|-------|-------|-------|
     | 0 : 0 |   :   |   :   |   :   |   712 Background
     |-------|-------|-------|-------|
     |   :   |   :   |   :   |   :   |
     |-------|-------|-------|-------|
     | 0 : 1 |   :   |   :   |   :   |   708
     |-------|-------|-------|-------|
     |   :   |   :   |   :   |   :   |
     |-------|-------|-------|-------|
     | 1 : 0 |   :   |   :   |   :   |   709
     |-------|-------|-------|-------|
     |   :   |   :   |   :   |   :   |
     |-------|-------|-------|-------|
     | 1 : 1 |   :   |   :   |   :   |   710 or 711
     |-------|-------|-------|-------|
     |   :   |   :   |   :   |   :   |
     |-------|-------|-------|-------|
```

You plot in an 8 by 8 grid, but unlike the normal Graphics 0 definition, 2 bits are now paired to represent 1 pixel of selective colour. The colour is taken from the register depending on the bit-pair as shown. Should the bit-pair be 1:1, then the colour will be taken from register 710 for normal characters, and register 711 when the character is inversed.

Try changing line 160 of the program to:

```
160 DATA 0,0,0,85,0,170,0,255,0
```

As you can see, there are 4 colours across the screen. If you want the 5th colour, then try printing 'inverse' spaces. To change the colours simply change locations 708 - 712.

APPENDIX C13:

Graphics 13 is exactly the same as Graphics 12, except that every line of characters are twice as high, alike Graphics 2 compared to Graphics 1. These modes, especially mode-12 are extremely powerful modes in the Atari since you can create colourful detailed displays and save enormous memory and processing time. Graphics 12 is used very much in the production of commercial software because of these factors. Take for example perhaps 1 of the all time best arcade games on the 8-bit Atari, Boulderdash. This game uses 4 Graphics 12 characters in a square to build all of its game objects such as diamonds, boulders etc.. The 'whole' cave-screen takes approx 1300 bytes of memory, where as if the game used Graphics 15, the memory usage would be something like 10300 bytes. Of course, the larger the cave, the more memory you save! Basically, the entire memory usage of the Graphics 12 method is 1024 bytes character-set memory and screen memory (usually 960 bytes), a total of 1984 bytes. The Graphics 15 method takes just screen memory, but that is 7680 bytes. There are so many advantages to using Graphics 12, another very important one is the processing time to perform common tasks like printing, memory copying etc.. Quite often a program might find it just too slow to perform these tasks in Graphics 15, but on average, using Graphics 12 is not 8 times faster, but actually about 72 times faster. My calculation comes from the amount of cycles it takes for the equivalent machine-code instructions to copy the equivalent amount of display memory between the 2 modes.

Besides these advantages, there is 1 feature that poses a problem. That problem is the display of text, where it appears virtually unreadable. This happens because the character-set used is designed for Graphics 0 only. There are a few ways around this, firstly you could use the easy method:

```
1 GRAPHICS 12+16
2 POKE 87,0
3 POKE 708,0:POKE 709,8:POKE 710,8
4 LIST
5 STOP
```

This isn't that bad considering. It's a very simple technique and does the job for most applications, but if you want proper clarity then you'll have to redesign the character-set. Use the earlier program and make the following additions/alterations:

```
100 READ CH:IF CH=-1 THEN 180
160 DATA 33,0,252,204,204,252,204,204,0
162 DATA 34,0,252,204,240,204,204,252,0
164 DATA 35,0,252,192,192,192,192,252,0
180 ? #6;"ABC  ABC"
182 REM NOTE* THE 2ND ABC IS INVERSED
```

APPENDIX C13:

I've just redesigned the 1st 3 characters of the character-set (internal codes 33 - 35), and in doing so I've set both bits of the bit-pair for each pixel displayed so that registers 710 and 711 are used. This allows you to have your text in any of 2 colours depending on whether you inverse the text or not.

If you want to be able to display text in all of the 5 standard available colours then that's a little trickier. Obviously, without using DLI's, you would have to create 3 sets of character definitions within the 1 character-set. The lower-case characters can be used for 1 definition and the graphics characters can become the other definition, whilst the standard capital character range can be the standard and inverse colours. Having a 5th colour present at the same time as these other 4 does become tricky, though. It depends on what characters, numbers or symbols you want or don't want in your character-set. If you don't want the numbers and the symbols above them then you can redefine 20 letters under these ranges. Try including this data to the previous program:

```
166 DATA 65,0,84,68,68,84,68,68,0
168 DATA 214,0,168,136,136,168,136,136,0
```

You can now print a capital "A" to a Graphics 12 screen in any of 4 colours by using a capital "A" normal and inversed, "control" and "A", and a non-capital "a". Why not convert the entire character-set, most font editor programs allow for Graphics 12 or Antic 4 as it's otherwise known. You can get hold of a suitable program from the public domain. TWAUG's public domain has fairly recently included an apparently excellent font editor. Why don't you give them a call. Happy font making.

As a last point for your curiosity. There exists 1 or 2 programs that will convert your Graphics 15 displays into Graphics 12 ones. One such program is called LOGO's MAKER (by THORGAL I believe). The result will require several DLI's to change the character-set at several points on the screen, but it brings about increased advantage. It don't just give you 1 extra colour, in fact, it actually gives an extra 4 colours. See the last paragraph of the MODES appendix. The technique is known as Bleeding colours.

APPENDIX C14:

DISPLAY LISTS.

Building your own display lists is a brilliant feature with the Atari, instead of having a boring mode of all the same mode-lines, you can create a special mode for title-screens or games of all types. In fact, all of the standard modes were specially designed the way they are to give the user a varied choice for the displaying of text and graphics.

The charts at the end of this appendix shows you the list of display codes (the Display Lists DLs) for all of the standard modes with and without text window. For a full description of the codes and how to use them refer to locations 560 and 561 in the map. Note that the addresses for Display Memory (DM) are given for all XL/XEs except the 600XL. No worries, however, since all of the modes DM requirements have been calculated immediately below RAMTOP. Thus, because RAMTOP is 160 in all Atari' except the 600XL, to find the correct addresses in this one all you have to do is relate the values to your RAMTOP alike they do to a RAMTOP value of 160. For example, the MSB of 156 in the immediate DM address of mode 0 is 4 pages lower than a RAMTOP of 160. So, to find the address on the 600XL just take 4 pages from its RAMTOP (location 106).

Before you can create your own DLs, there are a few things you should understand. In looking at the original DL tables, you may wonder why there are unused bytes all over the place. Well, this is very important to know. Unfortunatly, there are 2 problems that you should always make sure you have correct. They are both described in the BOUNDARIES appendix. Basically, your Display List (DL) cannot run straight through a 1K boundary (a multiple of 1024 bytes) and the Display Memory (DM) is a little more tricky. If the amount of memory being displayed to the screen exceeds 4K (4096 bytes), then a second LMS instruction (described at 560 and 561) must occur exactly on the byte that pertrudes the next 4K boundary from the prior one! Again, it is only possible to put an LMS instruction on a particular mode-line (and not an individual byte), so you should realise that should your DM be exceeding 4K, then it must be pre-calculated so that this pertruding byte is at the beginning of the 1st mode-line in the 2nd 4K boundary, thus, you then put your LMS instruction on this mode-line (this byte!). Hence, the last byte of the previous 4K boundary is the last byte of its last mode-line!

What I described there (?) is the only real difficulty in creating Display Lists. There are a few other points that you should abide by, but they are not a necessity. Besides, with a, little practice of DM calculating you'll find the creating of Display Lists a lot of fun!

APPENDIX C14:

CREATING YOUR OWN DLs.

In order to create your own DLs, you will need to reference the information given at locations 88, 89, 560, 561 and the BOUNDARIES appendix. Just before we do create our own unique DL, let me just clarify the problem and solution to DM that exceeds 4K using the following addresses as reference:

```
10 GRAPHICS 15+16
20 DL=PEEK(560)+256*PEEK(561)
30 DM=PEEK(DL+4)+256*PEEK(DL+5)
40 RT=PEEK(106)*256
```

From the table at locations 88 and 89, we can see that graphics mode 15 takes more than 4K (actually RT-DL bytes), so this is ideal to find out how the system calculates its modes DM.

The system tries to shuffle all its DM around the middle of the 2 boundaries as possible, so since we know that more than 4K is used, lets see how the highest 4K is used. The top 160 bytes are always reserved for the text-window, whether it be on or off. The remainder of this top 4K = 4096-160 = 3936. There are no other high reservations, so the next thing to do is to see how many mode-15 lines we can get out of this memory. 3936/40 = 98.4 what a pity. A non integer! We can have 98, but there is a .4 that must be left aside (remember the 2nd LMS problem). OK, 98 it is. If you multiply .4 by 40 then you'll get the amount of bytes we cannot use for mode-15 in this higher 4K boundary, which is 16! What a coincidence, it states this on the DL tables!

The system places these 16 unused bytes at the end of the 98 lines so that it can address the beginning of the 98 lines DM exactly on this 2nd 4K boundary. Anyway, since graphics 15 offers 192 mode-lines, 192-98 leaves 94 to be put in the lower 4K boundary (the hi-part of the screen). 4096-(94*40) = 336 bytes, the amount of memory left (from 8K) for the actual Display List (DL). Thus, you should now see that the bottom 98 lines DM of mode-15 begins at RT-4096, while the top 94 lines DM begins at RT-4096-(94*40) (DM)!

The DL occupies the last bytes of the 1st page in the 1st 4K boundary (in 8K modes), thus, 256 (1 page) - 202 = 54. The 54th byte into the 1st 4K boundary is the beginning of the DL for mode 15+16. It isn't so important to calculate the DL, so long as you keep it in it's original area all should be fine because there are also a few pointers in the memory that are pre-calculated, such like the top of program memory at 741 and 742 where it points to the DL-1.

Well, with that tricky explanation out of the way, why don't we create a unique DL of our own. Now, what shall we do? Speak up, I can't hear you. Hmmm, I guess they haven't invented the conversable book!

Enough of this. Since we already have a split screen game DL (in some other appendix?), why don't we create a title-screen DL. It won't exceed 4K, but that split-screen DL does and I'm sure that you can understand how I went about creating that one with the information there and here.

So then, how does a few mode 2 lines, a couple of mode 1 lines, a small graphics 8 area and a finish of mode 0 lines sound? If you reference locations 88, 89, 560 and 561 you'll get the following information:

Basic Graphics Mode	Amount of Mode Lines	Antic Code	Scan Lines	Display Memory Bytes
2	4	7	16*4= 64	20*4= 80
1	2	6	8*2= 16	20*2= 40
8	80	15	1*80= 80	40*80= 3200
0	4	2	8*4= 32	40*4= 160
			---	----
			192	3480

You'll notice that the total number of scan-lines the DL occupys is 192. This isn't essential, but should be abided by for various flickery reasons (in some cases). Another point of note, is that the total memory usage is less than 4K (3480 + DL). Now, this can become tricky, since you have to protect this much memory. A Graphics call will protect the memory you need, and the normal procedure would be to call Graphics 8+16, the reason being that it is the highest memory usage mode which is used in our DL, but this will protect 8K! If you want to protect only that memory necessary, then you can use a better technique. If your DM usage is less than 4K then call Graphics 14, but if it uses more then call Graphics 15.

The DL in its present form isn't quite complete. Firstly, you should include 3 8-BSL instructions at the beginning of the DL to centre it on the screen. It should have an LMS instruction on the very 1st mode-line instruction (code 64) and you should follow that with the start address of DM. Lastly, you must include a JVP instruction at the end of the DL followed by 2 bytes which address the beginning of the DL. This is to instruct Antic that the DL is finished. Take the program on the next leaf:

APPENDIX C14:

```
10 GRAPHICS 14
12 DL=PEEK(560)+256*PEEK(561)
20 FOR I=0 TO 10
22 READ D:POKE DL+I,D:NEXT I
24 DATA 112,112,112,71,0,144,7,7,7
26 DATA 6,6
30 FOR I=11 TO 90:POKE DL+I,15:NEXT I
32 POKE DL+91,66
34 POKE DL+92,PEEK(660)
36 POKE DL+93,PEEK(661)
38 FOR I=94 TO 96:POKE DL+I,2:NEXT I
44 POKE DL+97,65
46 POKE DL+98,PEEK(560)
48 POKE DL+99,PEEK(561)
50 RT=PEEK(106):DM=(RT-16)*256
52 H=INT(DM/256):L=DM-H*256
54 POKE 88,L:POKE DL+4,L
56 POKE 89,H:POKE DL+5,H
```

There you have it. Screen memory begins at DM, the Display List begins at DL. I've also included an LMS instruction on the 1st of the bottom 4 Graphics 0 lines so that they use the text-window memory. If you wish to place text in the Graphics 1 or 2 lines, then use IOCB #6 (? #6), but ensure you POKE 87 with the mode your printing or drawing to. The Graphics 8 area can be accessed with a POKE 87,8 and the rows are from 3 to 82.

Here's the memory configuration for this Display List:

```
|----------|
|100bytesDL|
|          |
| 74 bytes |
|  unused  |
|----------|  4K BOUNDARY
| 80 bytes |
|  mode-2  |
|----------|
| 40 bytes |
|  mode-1  |
|----------|
|3200 bytes|
|  mode-8  |
|----------|
| 776bytes |
|  unused  |
|----------|
|          |
| 160bytes |
|textwindow|
|----------|  RAMTOP
```

APPENDIX C14:

0		
112	*3	
66	64	156
2	*23	
65	32	156

1		
112	*3	
70	128	157
6	*19	
66	96	159
2	*3	
65	94	157

2		
112	*3	
71	112	158
7	*9	
66	96	159
2	*3	
65	88	158

3		
112	*3	
72	112	158
8	*19	
66	96	159
2	*3	
65	88	158

4		
112	*3	
73	128	157
9	*39	
66	96	159
2	*3	
65	74	157

5		
112	*3	
74	160	155
10	*39	
66	96	159
2	*3	
65	106	155

6		
112	*3	
75	224	151
11	*79	
66	96	159
2	*3	
65	130	151

7		
112	*3	
77	96	144
13	*79	
66	96	159
2	*3	
65	96	159

8		
112	*3	
79	80	129
15	*93	
79	0	144
15	*65	
66	96	159
2	*3	
65	80	128

9		
112	*3	
79	80	129
15	*93	
79	0	144
15	*97	
65	54	128

10		
112	*3	
79	80	129
15	*93	
79	0	144
15	*97	
65	54	128

11		
112	*3	
79	80	129
15	*93	
79	0	144
15	*97	
65	54	128

12		
112	*3	
68	160	155
4	*19	
66	96	159
2	*3	
65	126	155

13		
112	*3	
69	128	157
5	*9	
66	96	159
2	*3	
65	104	157

14		
112	*3	
76	96	144
12	*160	
66	96	159
2	*3	
65	82	143

15		
112	*3	
78	80	129
14	*93	
78	0	144
14	*65	
66	96	159
2	*3	
65	80	128

APPENDIX C14:

0+16			1+16			2+16			3+16		
112	*3		112	*3		112	*3		112	*3	
66	64	156	70	128	157	71	112	158	72	112	158
2	*23		6	*23		7	*11		8	*23	
65	32	156	65	96	157	65	96	157	65	80	158

4+16			5+16			6+16			7+16		
112	*3		112	*3		112	*3		112	*3	
73	128	157	74	160	155	75	224	151	77	96	144
9	*47		10	*47		11	*95		13	*95	
65	72	157	65	104	155	65	120	151	65	152	143

8+16			9+16			10+16			11+16		
112	*3		112	*3		112	*3		112	*3	
79	80	129	79	80	129	79	80	129	79	80	129
15	*93		15	*93		15	*93		15	*93	
79	0	144	79	0	144	79	0	144	79	0	144
15	*97		15	*97		15	*97		15	*97	
65	54	128	65	54	128	65	54	128	65	54	128

12+16			13+16			14+16			15+16		
112	*3		112	*3		112	*3		112	*3	
68	160	155	69	128	157	76	96	144	78	80	129
4	*23		5	*11		12	*191		14	*93	
65	128	155	65	108	157	65	56	143	78	0	144
									14	*97	
									65	54	128

APPENDIX D

APPENDIX D1:

VBLANK processes

The VBlank routines were formerly documented in the OS source listing, pages 34 - 37. In the XL/XE's they are processed at 49378; $C0E2.

Stage-1 VBLANK:
Performed every VBI:

1. Increment the realtime clock at 18 - 20; $12 - $14
2. Process the attract mode variables at 77; $4D
3. Decrement system timer-1 at 536; $218 and if 0, then JSR through 550; $226

Stage-2 VBLANK:
Performed every VBI which does not interrupt critical sections, see CRITIC at 66; $42.

1. Update the Hardware registers from the shadow registers as follows:

Shadow:	Hardware:	Update reason:
SDLISTL/H	DLISTL/H	Display List end
SDMCTL	DMACTL	
CHBAS	CHBASE	
CHACT	CHACTL	
GPRIOR	PRIOR	
COLOUR0-4	COLPF0-4,BAK	Attract mode
PCOL0-3	COLPM0-3	
LPCNV/H	PENV/H	Light pen
STICK0-1	PORTA	Joysticks
PTRIG0-3	PORTA	Paddle triggers
PADDL0-3	POT0-3	Paddles
STRIG0-1	TRIG0-1	Joystick triggers
n/a	CONSOL	Console speaker off

2. System timers 2 - 5 are decremented, and if 0, the corresponding flag/JSR is performed. See 538 - 545.
3. A character is read from POKEY keyboard register at 53769 and read into CH at 764 if the auto-repeat is active.
4. The keyboard debounce counter is decremented by 1 if it is not 0 and if no key is being pressed.
5. Keyboard auto-repeat logic is processed.
6. Exit the VBLANK routine through 58466; $E462.

APPENDIX D2:

ATARI TIMING VALUES:

Ian Chadwick missed out on 1 important point when he wrote Mapping, and that was the timing values in its Appendix-3. Only the NTSC values were given! Here are the timing values for both NTSC and PAL.

PARTICULAR:	PAL:	NTSC:
Clock freq.	2.217MHz	1.79MHz
1 Machine-cycle	0.562usec	0.558usec
1 TV-frame	1/50th sec	1/60th sec
Scan-lines	312/frame	262/frame
Colour-clocks	228/scan-line	228/scan-line
" "	2/machine-cycle	2/machine-cycle
Machine-cycles	35568/frame	29868/frame
" "	114/scan-line	114/scan-line

The VBLANK differs between PAL and NTSC also. On NTSC Atari', the VBLANK is 7980 machine-cycles, but on the PAL Atari', it is 13680 machine-cycles. The time is further reduced depending on what graphic mode you are in and whether you use PMG' with PMBASE. It's all to do with cycle-stealing with DMA. See the CYCLE-COUNTING appendix.

Horizontal blank times:	
Wide playfield	18 machine-cycles
Normal playfield	34 " "
Narrow playfield	50 " "

Here are my calculations:
1 (second) / 50 (frames) = 0.02 = 20ms (time per PAL frame), 0.02 / 312 (scan-lines) = 64.103us per line, and 64.103us / 114 (cycles/line) = 0.562us (1 cycles time). The frame cycles is: 312 (scan-lines) * 114 (cycles/frame) = 35568. On a standard Atari DL, there are 192 scan-lines, or 192 * 114 = 21888 cycles per DL. The remaining 120 scan lines are considered as the VBLANK time; 120 * 114 = 13680 cycles.

The PAL CPU is 19% faster than the NTSC, but the TV frame refresh rate is 12% slower in the PAL system. The VCOUNT register at 54283; $D40B keeps track of the present scan-line the TV electron-beam is processing divided by 2. PAL systems range from 0 - 156, but NTSC ranges between 0 - 131.

APPENDIX D3:

CYCLE STEALING.

Calling a Graphics mode is very easy to accomplish I'm sure you'll agree, but how about sustaining that mode? Unless you know, you'd probably say what the heck is he talking about! Well, you'll find in this appendix some fascinating details about retaining the display screen. Advanced users will probably find my look-up charts very handy.

When you call a Graphics mode, of course the screen appears, but it is actually 'there' only 50 times a second (plus transistor de-luminizing time!). The technical details of a frame you'll find in the TIMINGS appendix. Anyway, without going into the television specifics, the visual image is constructed within the Antic chip and sent to the GTIA chip for colouring etc.. This appendix breaks down the constructing process, the main reason being that occasionally, more advanced programmers need to know how much time is available per frame, and sometimes even how much time is available to DLI's in particular areas of the screen. The less knowledgeable users will find this information imaginatively illuminizing.

Although the Atari supplies you with 35568 cycles (29968 NTSC) of processing time per frame, this time is actually much shorter due to 'cycle stealing' performed by the Antic chip to create the television picture. The amount of cycles stolen depends on the Display mode and Player/Missile Graphics as follows:

For each byte of Display Memory fetched (DMA'd), 1 cycle is stolen. 1 cycle is also stolen for every byte in the DL, so if your DL is 32 bytes long then 32 cycles would be a DL steal. If a DL instruction is for memory-map graphics (not text) AND the memory-map mode line extends greater than 1 scan-line in height, then the data for each scan-line of the mode-line is only fetched on the top scan-line of the mode-line (!).

Memory refresh takes 9 cycles for EVERY scan-line in the frame, unless pre-empted by a high resolution graphics mode. This last sentence was mentioned in the Hardware manual, it's an indisputably crap explanation if you ask me because it is very confusing. Anyway, this is what I believe it's supposed to mean:

9 cycles are stolen by Antic to do memory refresh per scan-line EXCEPT on Hi-Resolution mode-lines, why I don't know is because it doesn't say!! I presume it's referring to horizontal resolution whereby there ain't enough cycles available on the scan-line to perform a clear refresh?

APPENDIX D3:

The book also says in another paragraph about 'lost cycles' in the Hi-Resolution modes, which is why I presume a horizontal deficiency in refresh time. Anyway, The Hardware manual goes on to say that memory refresh continues throughout the Vertical Blank. I only wish that the book was more thorough in its explanation because I find it very irritating!.

Missiles take 1 cycle every line in single-line res., and 1 every other line in double-line res.. The Hardware manual states that you cannot disable missile DMA whilst enabling player DMA, but according to DMACTL location 559; $22F, you can. Player DMA takes 4 cycles every 1 or 2 lines depending on resolution, as with missiles.

The Player/Missile and Display List (DL) instructions fetch DMA process occurs during the horizontal blank if they are required for the following scan-line. In memory-map modes, the graphics data is fetched as it is required across the 1st scan-line of the mode-line. Again, if the mode-line is greater than 1 scan-line in height, then the already fetched data is remembered by Antic and used accordingly. In character modes, the character codes for that mode-line are fetched in the 1st scan-line inclusive of the font data needed for that scan-line, while in all succeeding scan-lines, only the font data itself is fetched as required. The character codes are remembered.

In a standard Graphics 0 mode, the Hardware manual states that most of the cycles in the top scan-line of each mode-line are used up, so there is time for only 1 memory refresh cycle instead of the usual 9. What this means I don't know, since I can't SEE any difference. In the narrow width screen, you get 2 memory refresh cycles. Again, as explained earlier (another confusing bit for you), the memory refresh cycle is done fast enough to make up for 'lost cycles' in the high resolution modes (see my presumption, explained earlier). Once memory refresh starts on a hi-res scan-line, it re-occurs every 4 cycles unless pre-empted by DMA (I'm in out of my depth!). Actually, what is meant is that refresh takes place unless it conflicts with the time a byte is accessed from memory (DMA) (?).

All interrupts reach the CPU near the end of the horizontal blank, with standard or narrow screen widths, refresh DMA begins after the end of the horizontal blank. The time at which Antic performs cycle stealing is not static, it all depends on the graphics mode, the screen width and whether or not the horizontal fine-scrolling bit has been set on the mode line. Horizontal fine-scrolling is achieved by delaying the time at which DMA takes place for even numbered colour clocks, to scroll odd numbered colour clocks, Antic has an internal 1 colour clock delay. Overleaf is a diagram showing you the exact occurrence of cycle stealing:

APPENDIX D3:

Here's the explicit timing of cycle stealing:

```
| End of    |
| Previous  | Horizontal                  |
| Line      | Blank                       |
|           |                             |
|           | 20 Cycles/40 C/Clks         |
|           |                             |
|
|   |   |   |   |   |   |   |   |
| 5 | 4 | 1 | 1 | 4 | 2 | 8 | 1-9
|   |   |   |   |   |   |   |   |
|
|          |    |    |    |    |   Refresh cycles &
WSYNC      |    |    |    |    |   Char/Graphics
           |    |    |    |    |   data DMA
           |    |    |    |    |   (depends on
           |    |    |    |    |    graphic mode)
           |    |    |    |    |
           |    |    |    |    |-- Interrupt
           |    |    |    |
           |    |    |    |-- Address DMA
           |    |    |        (3 byte DL instruction)
           |    |    |
           |    |    |-- Player DMA
           |    |
           |    |--  DL instruction fetch DMA
           |
           |-- Missile DMA
```

Here's a quick-reference to calculating cycle loss per frame:

Stealing Purpose	Cycles Stolen
Missiles	1 cycle per line Single-line res. 1 cycle every other line for double-line res.
Players	4 cycles per line single line res. 4 cycles every other line for double-line res.
DL	1 cycle per byte
DM	1 cycle per byte, for text modes add 1 cycle for every character
Refresh 1st	9 cycles every scan-line, but only 1 in scan-line of mode-0 rows (narrow width=2)

APPENDIX D3:

The Hardware manual gave an example of cycles loss in Graphics mode 0, you'll find that example below, and following it are my calculations of cycle loss for every other mode.

Graphics 0 cycle loss example:

The DL is 32 bytes long, thus, 32 cycles are lost due to this. 960 cycles are lost to DMA the characters (40*24), and 8*960 cycles are taken to DMA the character data (each font row). Refresh DMA takes 9 cycles for 312 scan-lines (262 NTSC) except for the 1st scan-line on all 24 mode-lines, where only 1 refresh cycle takes place. Thus:

PROCESS			CYCLES LOSS
DL			32
Characters	40*24	=	960
Char.data	960*8	=	7680
Refresh	312*9-24*8	=	2616
TOTAL		=	11288

Thus, the total DMA is 11288 cycles lost per PAL frame in Graphics 0. For the NTSC frame-loss, then change Refresh to 262*9-24*8 which = 2166, giving the total time loss per NTSC frame of 10838 cycles. 36% loss of total frame time of 29868 cycles. The PAL frame loss is 32% from 35568 cycles.

Graphics Mode	Time Loss (cycles) PAL	NTSC	Graphics Mode	Time Loss (cycles) PAL	NTSC
0	11288	10838	0+16	11288	10838
0.5	11043	10593		11043	10593
1	7850	7400	1+16	7160	6710
2	6040	5590	2+16	4988	4538
3	4450	4000	3+16	3080	2630
4	4670	4220	4+16	3344	2894
5	5070	4620	5+16	3824	3374
6	5910	5460	6+16	4832	4382
7	7510	7060	7+16	6752	6302
8	10792	10342	8+16	10690	10240
9	10792	10342	9+16	10690	10240
10	10792	10342	10+16	10690	10240
11	10792	10342	11+16	10690	10240
12	11290	10840	12+16	11288	10838
13	7840	7390	13+16	7052	6602
14	8256	7806	14+16	6848	6398
15	10792	10342	15+16	10690	10240

With a bit of luck, all of the calculations are correct. GTIA modes 9,10 and 11 have been calculated with and without a text-window.

APPENDIX D3:

Graphics 0.5 (the 10 row mode 0) has been calculated for the amount of mode-lines that will fit into 192 scan-lines (19). The text-window in this mode is of the same Antic code. Obviously, the timing for all the modes is only as accurate as my interpretation of the given facts. If anyone believes any information is wrong, then please get in touch with me and we'll sort it out. While your pondering in bafflement as to how I achieved some of the calculations, then here is how I went about a couple of them:

Graphics 1 with text-window:

Display List (DL)		34
Characters	20*24	400
Character-data	400*8	3200
Refresh	312*9	2808

...not forgetting the text-window:

DL	already included	
Characters	40*4	160
Character-data	160*8	1280
Refresh	already included, except that we must subtract 4*8 because only 1 refresh cycle occurs on 1st scan-line of each row.	
TOTAL		7850

To find the NTSC value, then change Refresh to 262*9 or simply take 450 off the PAL value. All text-windows are the same, so you can work all text-window modes out by firstly calculating the standard screen and adding all refresh time, and then adding 1408. The 1408 is found with 160 (characters) + 160*8 (char.data) and subtracting 4*8 (refresh loss per 4 rows). Graphics 2 and 13 are similar to Graphics 0, 1 and 12 in that you only multiply the characters by 8 to find the character data, since there are only 8 different rows in all cases. Refresh does the rest.

Graphics 15 without text-window:

Display List (DL)		202
Characters		n/a
Character-data		n/a
Disp. Memory (DM)	40*192	7680
Refresh	312*9	2808
TOTAL		10690

Graphics modes such as 7 are alike Graphics 15. Modes 12 and 13 lose 8 refresh cycles on the 1st scan-line of each mode-line also, as like Graphics 0. Any problems, contact me.

APPENDIX D4:

MACHINE LANGUAGE.

This appendix is merely meant as a informational reference, it will not teach M/L, but if you're an experienced Basic programmer then you will gain some knowledge and insight into Machine-code if you use your head.

Given on the next few pages are the instruction code charts showing the bytes required, cycles processing time and flags affected. From the chart, you will notice the 13 different 6502 addressing modes. They are as follows:

01. Immediate:

The immediate addressing mode is the easiest one to understand by far, it is also the quickest of the 13 modes along with Implied and Accumulator. An instruction in this mode comprises of 2 bytes, the instruction byte itself and 1 byte which is treated as direct/immediate data. You can associate this mode with the Basic statement: LET R=V.

02. Absolute:

Comprising of 3 bytes, this mode uses 2 data bytes which are treated as an LSB/MSB address to a memory location. It would be used in the same manner as R=PEEK(M), where M is achieved by LSB+256*MSB.

03. Zero Page:

This is exactly the same as the Absolute addressing mode, except that you can only address memory locations 0 - 255 (Page 0), thus, the instruction is comprised of 2 bytes only (no MSB).

04. Accumulator:

This mode uses just 1 byte, the instruction code itself. It's used to perform a maths operation on the Accumulator in the 6502. For instance, ASL A will multiply the contents of the Accumulator by 2, to multiply by 10, then repeat this instruction 3 times and add the original value twice. If you can't understand this, then get reading some machine-code book because it's quite a heavy subject. It's to do with moving the bits of a byte once to the left and manipulating any bits going out the left hand side of the byte (the high side) via the Carry flag and into the high byte (MSB). Any 6502 book will do, try the local library.

05. Implied:

A single-byte mode again, this time to perform a particular task such as setting or clearing a particular flag, incrementing an index register or whatever.

06. Indexed Indirect:

Namely (Ind,X) or (ZP,X). This mode comprises just 2 bytes, the instruction code and 1 byte that we shall call M for argument sake. Take the statement: R=PEEK(PEEK(M+X)+256*PEEK(M+1+X)). This is the exact function of this addressing mode. The contents of location M+X is the LSB, the contents of location M+1+X is the MSB, and R then equals PEEK(LSB+256*MSB). Easy eh!?

07. Indirect Indexed:

Namely (Ind),Y or (ZP),Y. This mode is very similar to the previous mode and is also 2 bytes in size. The Basic equivalent would be: R=PEEK(PEEK(M)+256*PEEK(M+1)+Y). The LSB and MSB is found beginning at M, then the Y index register is added to this value.

08. Zero Page,X:

This is the same as R=PEEK(M+X). Where on a load operation, the (R)egister in question would be loaded with the contents of location (M+X). The mode is 2 bytes in size.

09. Zero Page,Y:

Exactly the same as the above mode except the index register is Y and not X.

10. Absolute,X:

This mode is the same as mode 08, except that the address is absolute, thus, the instruction is comprised of 3 bytes.

11. Absolute,Y:

Exactly the same as mode 10, except that it uses the Y index register.

12. Relative:

Comprising of 2 bytes, this mode uses the instruction-code and 1 special byte which depicts where to branch to from the origin of the instruction. This is a very clever instruction which will involve a good bit of reading in a 6502 book.

13. Indirect:

This mode is 3 bytes in size and is used by only 1 instruction; JMP (M). At location M and M+1 is the LSB and MSB of the real address in memory to actually JuMP to.

APPENDIX D4:

In addition to the modes and instructions, there is a processor status register which shows the current status of the 8 6502 flags; Negative, Overflow, Reserved, Break, Decimal, Interrupt, Zero and Carry.
The Negative flag is set positive when the most recent value processed goes negative, used in conjunction with BMI and BPL; Branch if result MInus (N=1) and Branch if result PLus (N=0). Overflow is set if bits overflow the size of the byte and carry being processed using ADC and SBC instructions. BReaK is set positive when the BRK instruction is executed. The Decimal flag is selectively set to tell the 6502 to handle bytes in Decimal mode, not Hexadecimal. This way you can add decimal numbers together and obtain decimal results instead of converting between the number systems Decimal and Hexadecimal. The numbers must, however, be represented as Binary Coded Decimal (BCD) numbers. See Number Systems appendix. The Interrupt flag is set positive to disable the IRQ' temporarily. This is especially useful when executing code that must occur at a particular time on the screen. Clear the flag with CLI to re-enable IRQ execution. The Zero flag is set positive when the result of an operation is 0, used in conjunction with BNE and BEQ; Branch if Not Equal to 0 (Z=0) and Branch if EQual to 0 (Z=1). The Carry flag is mainly used to show that a carry has occurred when adding/subtracting 2 numbers together (the result is greater than 255 or less than 0). The Carry flag can also be used to test if a number is less than, equal to or greater than another number by using the Compare instructions. Again, a 6502 book will give you all the information you need and they are usually quite easy to get hold of. The final flag is the reserved/unused one. In all sources this flag is said to be unused, but in fact, I have reason to believe that it can increase the speed of the main-clock pulse. If I remember how I cleared this bit before this book is finished, then I'll include the information, but all I can recall is that you've got to use 2-3 instructions, one of which is an illegal one!

Here's a description of the characters used in the flags column of the instruction charts:
Except for the BIT instruction, all other numbers in the flags column refer to the state of the flag. The numbers in the BIT flags refer to the status of the according bits of the byte being operated upon. 'Y' means the flag is affected and 'C' alongside SBC refers to note C. Lastly, be careful when acting on flags affected from illegal operations. Some of the flags return unusual status' for different operation-byte values... especially the DCP instruction! One case is where the Zero-flag is set when a result of #$AA is reached using DCP!

The 'Notes' are described under the illegal instructions.
Well folks, in this machine-language appendix you will notice some Assembler Mnemonics you will not have come across before, so here is some description on them:

APPENDIX D4:

1. AAC

First on the list is AAC; AND Accumulator with byte and if result is negative then set Carry.

2. AAX

AND the X-register with the Accumulator and store the result in memory.

3. ABX

AND Accumulator with byte, then AND this result with the X-register and then store the result in the Accumulator.

4. ARR

AND Accumulator with byte, then rotate 1-bit right in the Accumulator and check bits 5 and 6.
If both bits are 1 then set C and clear V,
" " " " 0 " clear both C and V,
" only bit 5 is 1 " " C and set V,
" " " 6 " " " set both C and V.

5. ASR

AND Accumulator with byte, then shift right 1-bit in the Accumulator.

6. ATX

AND Accumulator with byte, then transfer Accumulator to the X-register.

7. AXA

AND the Accumulator with the X-register, then AND this result with Q and store in memory. Note, that wherever Q is used, the value maybe 1, 3 or 7. I haven't fully discovered what makes the difference as yet, perhaps someone can do a survey of 6502'.

8. AXS

AND Accumulator with X-register, then subtract byte (without Carry) from this result and store in the X-register.

9. DCP

Decrement memory. The only difference between DCP and DEC is the way in which the flags are affected. Many of the illegal-instructions affect the flags in odd ways, so be careful if you do use these codes.

10. DOP

No operation (Double NOP).

11. ISC

Increment memory by 1, subtract memory from Accumulator (with Carry) and store result in Accumulator.

12. KIL

Freeze Program Counter (PC; program lockup). I can't seem to put a processing time for these instruction-codes on the illegal-codes table because I can't find a way to time them!

13. LAR

AND memory with the Stack-Pointer and transfer the result to the Accumulator, X-register and Stack-Pointer.

14. LAX

Load the Accumulator and X-register with memory.

15. NOP

No operation.

16. RLA

Rotate 1-bit left in memory, AND Accumulator with memory and store result in the Accumulator.

17. RRA

Rotate 1-bit right in memory, then add memory to the Accumulator with Carry).

18. SBC

Exactly the same as SBC #byte.

19. SLO

Shift 1-bit left in memory, OR Accumulator with memory and store result in the Accumulator.

20. SRE

Shift right 1-bit in memory, EOR Accumulator with memory and keep result in Accumulator.

21. SXA

AND X-register with Q and store result in memory. Where Q maybe 1, 3 or 7.

22. SYA

AND Y-register with Q and store result in memory.

23. TOP

This instruction will turn your Atari into an Amiga. (NA!) Not really, I'm just kiddin'. Actually, it is a Triple NOP and does nothing at all except waste time.

24. XAS

AND Accumulator with X-register and store the result in the Stack-Pointer, then AND Stack-Pointer with Q and store the result in memory.

Thats all folks! They are the 24 illegal-instructions of the 6502. Enjoy them!

For your convenience, I thought it a good idea to also include the full instruction list in numerical order, giving both the decimal and hexadecimal codes along with the addressing mode that each instruction uses.

I've always found that when I'm browsing through other peoples demos, games etc. to see how they perform a particular task, I find there are a few codes that I don't know off by hand. I then have to search through the instruction table to find it. This can be quite a pain sometimes, but with the numerical ordered list, it's much easier and quicker to find the byte and to see what mode it uses.

0	00	BRK	Implied	58	3A	nop	Implied
1	01	ORA	(Zpag,X)	59	3B	rla	Abs,Y
2	02	kil	Implied	60	3C	top	Abs,Y
3	03	slo	(Zpag),Y	61	3D	AND	Abs,X
4	04	dop	Zpag	62	3E	ROL	Abs,X
5	05	ORA	Zpag	63	3F	rla	Abs,X
6	06	ASL	Zpag	64	40	RTI	Implied
7	07	slo	Zpag	65	41	EOR	(Zpag,X)
8	08	PHP	Implied	66	42	kil	Implied
9	09	ORA	Imm	67	43	sre	(Zpag,X)
10	0A	ASL	Accum	68	44	dop	Zpag
11	0B	aac	Imm	69	45	EOR	Zpag
12	0C	top	Abs	70	46	LSR	Zpag
13	0D	ORA	Abs	71	47	sre	Zpag
14	0E	ASL	Abs	72	48	PHA	Implied
15	0F	slo	Abs	73	49	EOR	Imm
16	10	BPL	Relative	74	4A	LSR	Accum

17	11	ORA	(Zpag),Y	75	4B	asr	Imm
18	12	kil	Implied	76	4C	JMP	Abs
19	13	slo	(Zpag,X)	77	4D	EOR	Abs
20	14	dop	Zpag,X	78	4E	LSR	Abs
21	15	ORA	Zpag,X	79	4F	sre	Abs
22	16	ASL	Zpag,X	80	50	BVC	Relative
23	17	slo	Zpag,X	81	51	EOR	(Zpag),Y
24	18	CLC	Implied	82	52	kil	Implied
25	19	ORA	Abs,Y	83	53	sre	(Zpag),Y
26	1A	nop	Implied	84	54	dop	Zpag,X
27	1B	slo	Abs,Y	85	55	EOR	Zpag,X
28	1C	top	Abs,X	86	56	LSR	Zpag,X
29	1D	ORA	Abs,X	87	57	sre	Zpag,X
30	1E	ASL	Abs,X	88	58	CLI	Implied
31	1F	slo	Abs,X	89	59	EOR	Abs,Y
32	20	JSR	Abs	90	5A	nop	Implied
33	21	AND	(Zpag,X)	91	5B	sre	Abs,Y
34	22	kil	Implied	92	5C	top	Abs,X
35	23	rla	(Zpag,X)	93	5D	EOR	Abs,X
36	24	BIT	Zpag	94	5E	LSR	Abs,X
37	25	AND	Zpag	95	5F	sre	Abs,X
38	26	ROL	Zpag	96	60	RTS	Implied
39	27	rla	Zpag	97	61	ADC	(Zpag,X)
40	28	PLP	Implied	98	62	kil	Implied
41	29	AND	Imm	99	63	rra	(Zpag,X)
42	2A	ROL	Accum	100	64	dop	Zpag
43	2B	aac	Imm	101	65	ADC	Zpag
44	2C	BIT	Abs	102	66	ROR	Zpag
45	2D	AND	Abs	103	67	rra	Zpag
46	2E	ROL	Abs	104	68	PLA	Implied
47	2F	rla	Abs	105	69	ADC	Imm
48	30	BMI	Relative	106	6A	ROR	Accum
49	31	AND	(Zpag),Y	107	6B	arr	Imm
50	32	kil	Implied	108	6C	JMP	Indirect
51	33	rla	(Zpag),Y	109	6D	ADC	Abs
52	34	dop	Zpag,Y	110	6E	ROR	Abs
53	35	AND	Zpag,X	111	6F	rra	Abs
54	36	ROL	Zpag,X	112	70	BVS	Relative
55	37	rla	Zpag,X	113	71	ADC	(Zpag),Y
56	38	SEC	Implied	114	72	kil	Implied
57	39	AND	Abs,Y	115	73	rra	(Zpag),Y
116	74	dop	Zpag,X	174	AE	LDX	Abs
117	75	ADC	Zpag,X	175	AF	lax	Abs
118	76	ROR	Zpag,X	176	B0	BCS	Relative
119	77	rra	Zpag,X	177	B1	LDA	(Zpag),Y
120	78	SEI	Implied	178	B2	kil	Implied
121	79	ADC	Abs,Y	179	B3	lax	(Zpag),Y
122	7A	nop	Implied	180	B4	LDY	Zpag,X
123	7B	rra	Abs,Y	181	B5	LDA	Zpag,X
124	7C	top	Abs	182	B6	LDX	Zpag,X
125	7D	ADC	Abs,X	183	B7	lax	Zpag,X
126	7E	ROR	Abs,X	184	B8	CLV	Implied
127	7F	rra	Abs,X	185	B9	LDA	Abs,Y
128	80	dop	Imm	186	BA	TSX	Implied
129	81	STA	(Zpag,X)	187	BB	lar	Abs,Y

130	82	dop	Imm	188	BC	LDY	Abs,X
131	83	aax	(Zpag,X)	189	BD	LDA	Abs,X
132	84	STY	Zpag	190	BE	LDX	Abs,Y
133	85	STA	Zpag	191	BF	lax	Abs,Y
134	86	STX	Zpag	192	C0	CPY	Imm
135	87	aax	Zpag	193	C1	CMP	(Zpag,X)
136	88	DEY	Implied	194	C2	dop	Imm
137	89	dop	Imm	195	C3	dcp	(Zpag,X)
138	8A	TXA	Implied	196	C4	CPY	Zpag
139	8B	aax	Imm	197	C5	CMP	Zpag
140	8C	STY	Abs	198	C6	DEC	Zpag
141	8D	STA	Abs	199	C7	dcp	Zpag
142	8E	STX	Abs	200	C8	INY	Implied
143	8F	aax	Abs	201	C9	CMP	Imm
144	90	BCC	Relative	202	CA	DEX	Implied
145	91	STA	(Zpag),Y	203	CB	axs	Imm
146	92	kil	Implied	204	CC	CPY	Abs
147	93	dop	(Zpag,X)	205	CD	CMP	Abs
148	94	STY	Zpag,X	206	CE	DEC	Abs
149	95	STA	Zpag,X	207	CF	dcp	Abs
150	96	STX	Zpag,X	208	D0	BNE	Relative
151	97	aax	Zpag,Y	209	D1	CMP	(Zpag),Y
152	98	TYA	Implied	210	D2	kil	Implied
153	99	STA	Abs,Y	211	D3	dcp	(Zpag),Y
154	9A	TXS	Implied	212	D4	dop	Zpag,Y
155	9B	xas	Abs,Y	213	D5	CMP	Zpag,X
156	9C	sya	Abs,X	214	D6	DEC	Zpag,X
157	9D	STA	Abs,X	215	D7	dcp	Zpag,X
158	9E	sxa	Abs,Y	216	D8	CLD	Implied
159	9F	axa	Abs,Y	217	D9	CMP	Abs,Y
160	A0	LDY	Imm	218	DA	nop	Implied
161	A1	LDA	(Zpag,X)	219	DB	dcp	Abs,Y
162	A2	LDX	Imm	220	DC	top	Abs,Y
163	A3	lax	(Zpag,X)	221	DD	CMP	Abs,X
164	A4	LDY	Zpag	222	DE	DEC	Abs,X
165	A5	LDA	Zpag	223	DF	dcp	Abs,X
166	A6	LDX	Zpag	224	E0	CPX	Imm
167	A7	lax	Zpag	225	E1	SBC	(Zpag,X)
168	A8	TAY	Implied	226	E2	dop	Imm
169	A9	LDA	Imm	227	E3	isc	(Zpag,X)
170	AA	TAX	Implied	228	E4	CPX	Zpag
171	AB	atx	Implied	229	E5	SBC	Zpag
172	AC	LDY	Abs	230	E6	INC	Zpag
173	AD	LDA	Abs	231	E7	isc	Zpag
232	E8	INX	Implied	244	F4	dop	Zpag,Y
233	E9	SBC	Imm	245	F5	SBC	Zpag,X
234	EA	NOP	Implied	246	F6	INC	Zpag,X
235	EB	sbc	Imm	247	F7	isc	Zpag,X
236	EC	CPX	Abs	248	F8	SED	Implied
237	ED	SBC	Abs	249	F9	SBC	Abs,Y
238	EE	INC	Abs	250	FA	nop	Implied
239	EF	isc	Abs	251	FB	isc	Abs,Y
240	F0	BEQ	Relative	252	FC	top	Abs,Y
241	F1	SBC	(Zpag),Y	253	FD	SBC	Abs,X
242	F2	kil	Implied	254	FE	INC	Abs,X
243	F3	isc	(Zpag),Y	255	FF	isc	Abs,X

MNEM	FUNCTION	NOTE	IMM	ABS	ZPAGE	ACCUM	IMPLD	(ZP,X)	(ZP),Y	ZPAG,X	ABS,X	ABS,Y	RELATV	INDRCT	ZPAG,Y	FLAGS
			OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	NV-BDIZC
ADC	A=A+M+C	[4] [1]	69-2-2	6D-4-3	65-3-2			61-6-2	71-5-2	75-4-2	7D-4-3	79-4-3				YY—YY
AND	A=A&M	[1]	29-2-2	2D-4-3	25-3-2			21-6-2	31-5-2	35-4-2	3D-4-3	39-4-3				Y—Y-
ASL	C <76543210 <0			OE-6-3	06-5-2	OA-2-1				16-6-2	1E-7-3					Y—YY
BCC	BRANCH ON C=0	[2]											90-2-2			
BCS	BRANCH ON C=1	[2]											B0-2-2			
BEQ	BRANCH ON Z=1	[2]											F0-2-2			
BIT	A&M			2C-4-3	24-3-2											76—7-
BMI	BRANCH ON N=1	[2]											30-2-2			
BNE	BRANCH ON Z=0	[2]											D0-2-2			
BPL	BRANCH ON N=0	[2]											10-2-2			
BRK	BREAK						00-7-1									
BVC	BRANCH ON V=0	[2]											50-2-2			
BVS	BRANCH ON V=1	[2]											70-2-2			
CLC	C=0						18-2-1									—0
CLD	D=0						D8-2-1									—0—
CLI	I=0						58-2-1									—0–
CLV	V=0						B8-2-1									-0—
CMP	COMP.A WITH M	[1]	C9-2-2	CD-4-3	C5-3-2			C1-6-2	D1-5-2	D5-4-2	DD-4-3	D9-4-3				Y—YY
CPX	COMP.X WITH M		E0-2-2	EC-4-3	E4-3-2											Y—YY
CPY	COMP.Y WITH M		C0-2-2	CC-4-3	C4-3-2											Y—YY
DEC	M=M-1			CE-6-3	C6-5-2					D6-6-2	DE-7-3	59-4-3				Y—Y-
DEX	X=X-1						CA-2-1									Y—Y-
DEY	Y=Y-1						88-2-1									Y—Y-
EOR	A=A@M	[1]	49-2-2	4D-4-3	45-3-2			41-6-2	51-5-2	55-4-2	5D-4-3					Y—Y-
INC	M=M+1			EE-6-3	E6-5-2					F6-6-2	FE-7-3					Y—Y-
INX	X=X+1						E8-2-1									Y—Y-
INY	Y=Y+1						C8-2-1									Y—Y-
JMP	JUMP TO NEW LOC.			4C-3-3										6C-5-3		
JSR	JUMP TO SUB.ROUT			20-6-3												
LDA	A=M	[1]	A9-2-2	AD-4-3	A5-3-2			A1-6-2	B1-5-2	B5-4-2	BD-4-3	B9-4-3				Y—Y-
LDX	X=M	[1]	A2-2-2	AE-4-3	A6-3-2							BE-4-3			B6-4-2	Y—Y-
LDY	Y=M	[1]	A0-2-2	AC-4-3	A4-3-2					B4-4-2	BC-4-3					Y—Y-
LSR	0 >76543210>C			4E-6-3	46-5-2	4A-2-1				56-6-2	5E-7-3					0Y—Y
NOP	NO-OPERATION						EA-2-1									
ORA	A=A^M		O9-2-2	OD-4-3	05-3-2			O1-6-2	11-5-2	15-4-2	1D-4-3	19-4-3				Y—Y-
PHA	MS=A	S=S-1					48-3-1									
PHP	MS=P	S=S-1					08-3-1									
PLA	S=S+1	A=MS					68-4-1									Y—Y-
PLP	S=S+1	P=MS					28-4-1									RESTORED
ROL	< 76543210 <C <			2E-6-3	26-5-2	2A-2-1				36-6-2	3E-7-3					Y—YY
ROR	> C>76543210>			6E-6-3	66-5-2	6A-2-1				76-6-2	7E-7-3					Y—YY
RTI	RETURN FROM INT						40-6-1									RESTORED
RTS	RETURN FROM SUBR						60-6-1									
SBC	A=A-M-NOT C	[1]	E9-2-2	ED-4-3	E5-3-2			E1-6-2	F1-5-2	F5-4-2	FD-4-3	F9-4-3				YY—YC
SEC	C=1						38-2-1									—1
SED	D=1						F8-2-1									—1
SEI	I=1						78-2-1									—1
STA	M=A			8D-4-3	85-3-2			81-6-2	91-6-2	95-4-2	9D-5-3	99-5-3				
STX	M=X			8E-4-3	86-3-2										96-4-2	
STY	M=Y			8C-4-3	84-[illegible]-2					94-4-2						
TAX	X=A						AA-2-1									Y—Y-
TAY	Y=A						A8-2-1									Y—Y-
TSX	X=S						BA-2-1									Y—Y-
TXA	A=X						8A-2-1									Y—Y-
TXS	S=X						9A-2-1									
TYA	Y=A						98-2-1									Y—Y-

LEGEND	
A	ACCUMALATOR
X	X-REGISTER
Y	Y REGISTER
+	PLUS
-	MINUS
=	EQUALS
^	OR
&	AND
@	EOR
<	LEFT
>	RIGHT
76543210	BYTE
PC	PROG.COUNTER
S	STACK POINTER
MS	MEMORY/S.POINTER
M	MEMORY
#	CYCLES
B	BYTES
N	NEGATIVE
Z	ZERO
C	CARRY
I	INTERRUPT
D	DECIMAL
V	OVERFLOW

MNEM	FUNCTION	NOTE	IMM	ABS	ZPAGE	ACCUM	IMPLD	(ZP,X)	(ZP),Y	ZPAG,X	ABS,X	ABS,Y	RELATV	INDRCT	ZPAG,Y	FLAGS
			OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	OP-#-B	NV-BDIZC
AAC	A=A&M C=NOT N		OB-2-2													Y—YY
			2B-2-2													Y—YY
AAX	M=A&X			8F-4-3	87-3-2			83-6-2							97-4-2	Y—Y-
ABX	A=M&A&X		8B-2-2													Y—Y-
ARR	A=M&A then ROR A		6B-2-2													YY—YY
ASR	A=M&A then LSR A		4B-2-2													Y—YY
ATX	A=M&A X=A		AB-2-2													Y—Y-
AXA	A=X&A& [$] M=A											9F-5-3				
AXS	X=X&A X=X-M		CB-2-2													Y—YY
DCP	DEC M			CF-6-3	C7-5-2			C3-8-2	D3-8-2	D7-6-2	D7-7-3	DB-7-3				Y—YY
DOP	Double NOP		80-2-2		04-3-2			93-6-2		14-4-2					34-4-2	
			82-2-2		44-3-2					74-4-2					D4-4-2	
			89-2-2		64-3-2					54-4-2					F4-4-2	
			C2-2-2													
			E2-2-2													
ISC	M=M+1 A=A-M			EF-6-3	E7-5-2			E3-8-2	F3-8-2	F7-6-2	FF-7-3	FB-7-3				YY—YY
KIL	Stop PC; lockup						02-?-1									
							12-?-1									
							22-?-1									
							32-?-1									
							42-?-1									
							52-?-1									
							62-?-1									
							72-?-1									
							92-?-1									
							B2-?-1									
							D2-?-1									
							F2-?-1									
LAR	A=SP&M X=A SP=A											BB-4-3				Y—Y-
LAX	A=M X=M			AF-4-3	A7-3-2			A3-6-2	B3-5-2	B7-4-2		BF-4-3				Y—Y-
NOP							1A-2-1									
							3A-2-1									
							5A-2-1									
							7A-2-1									
							DA-2-1									
							FA-2-1									
RLA	ROL M A=M&A			2F-6-3	27-5-2			23-8-2	33-8-2	37-6-2	3F-7-3	3B-7-3				Y—YY
RRA	ROR M A=M+C			6F-6-3	67-5-2			63-8-2	73-8-2	77-6-2	7F-7-3	7B-7-3				YY—YY
SBC	Same as legal SBC		EB-2-2													YY—YY
SLO	ASL M A=M^A			OF-6-3	07-5-2			13-8-2	03-8-2	17-6-2	1F-7-3	1B-7-3				Y—YY
SRE	ASR M A=M@A			4F-6-3	47-5-2			43-8-2	53-8-2	57-6-2	5F-7-3	5B-7-3				Y—YY
SXA	M=X& [$]											9E-5-3				
SYA	M=Y& [$]										9C-5-3					
TOP	Triple NOP			0C-4-3							1C-5-3	3C-5-3				
				7C-4-3							5C-5-3	DC-5-3				
												FC-5-3				
XAS	SP=X&A M=SP& [$]											9B-5-3				

NOTES:

[1] Add 1 to "N" if page boundary is crossed

[2] Add 1 to "N" if branch occurs to same page, or 2 for a different page.

[3] If in Decimal Mode z flag is invalid, check Accumulator for Zero result.

[c] Carry not = below

[$] This value appears to change on different machines, on mine it has been 1 and 3, on another it has been 7.

APPENDIX D4:

Finally, I must give credit to MegaMagazine, where a fair chunk of the information in this appendix originated from. My appreciation to WosFilm and someone called Freddy (Hello).

Added note:

This note has been added to this page very late to give you some more information concerning the possibility of switching to a faster CPU clock by clearing the Reserved/unused bit of the Status Processor register (Bit-5).

It seems that I can't remember what instructions I used to clear this bit, although I believe that 1 of the 3 instructions was ABX. There is, however, a relatively easy way of finding out. If you can get hold of a program called "The 6502 Simulator", and simulate instructions from memory in the area of $BC40 (without any other programs loaded), you will find that the program seems to crash (the screen loses its lower half). Is this a bug in the program, or has it something to do with the main CPU clock changing speed??

When I get the time, I will be hunting this program down and I'll sort out fact from fiction. But in the meantime, on with this book...

APPENDIX D5:

VERTICAL BLANK INTERRUPTS.

Yeap! A 2nd appendix concerning the Vertical Blank. The other (D1) simply describes the existing OS VBI processes for both Immediate and Deferred VBIs, but this one should teach you how to create your own VBIs.

Similar to the DLI, you will need to understand the nature of the TV raster-scan, so if your unsure about this, the information you need is in the DLI appendix. As described in that appendix, the vertical blank is an interval of time which occurs when the TVs electron beam returns to the top left of the screen from the bottom right. On PAL systems this time period is a maximum of 13680 cycles, which approximates to 3420 machine-code instructions. This time is also reduced according to the Graphics mode, PMGs and a few other points (see the CYCLE-STEALING appendix).

To create your own VBI, you should firstly put your machine-code routine into a protected area of memory such as page-6 (1536 ;$600). Once this is done you can now decide whether your routine should be an immediate VBI or a deferred VBI. If you POKE the address of your VBI into the immediate vector at 546 - 547; $222 - $223, then your interrupt routine should normally end with a JMP to address $E45F. But, if your interrupt routine is intended to be deferred, then its address should be POKEd into the deferred vector at 548 - 549; $224 - $225, and your routine should end with a JMP to address $E462. By following these rules, then should your VBI be immediate, it would be the very 1st VBI executed, followed by the systems immediate VBI and then its deferred one. Should your VBI be deferred, then it will be the very last VBI executed. Note, that if CRITIC (location 66) is not clear, then the systems deferred VBI will not execute and neither will yours.

On the other hand, you do not have to follow these rules. You can completely disable the original systems VBIs and only activate your own by ending your immediate VBI with an indirect JMP ($224) through the deferred vector, or directly to $E462.

Having set your routine in protected memory which exits with the correct JMP address and setting the correct vector, you should enable your VBIs by setting bit-6 (decimal 64) at location 54286; $D40E. You should also know that when you POKE the address of your routine to the immediate or deferred vector, you must ensure that both LSB and MSB bytes are loaded before the VBI re-executes between the time you load them. You can do this by disabling the VBI while you do this in Basic. There is also another method which is described in the SYNCHRONIZED REGISTER LOADING appendix.

APPENDIX D6:

SYNCHRONIZED REGISTER LOADING.

Due to occasional mistiming with a users program and the setting of the VBI vectors, it is possible to accidentally set the LSB before the execution of a VBI, and the MSB 'too late' (after the interrupt exceedes the priority of the main program). In the case of this happening, the system would try to execute a VBI whose vector is only 'half' set, thus, jumping to a wrong area of memory and crashing the system.

There are many ways that you can overcome this simple problem. The 1st would be to disable the VBI interrupt until the correct vector LSB and MSB is fully loaded. A 2nd way would be to wait until the electron-beam is drawing a scan-line somewhere else on the screen. But, a 3rd way is possible which also allows some other features.

To perform a clean change, then you load the Y-register with the LSB, the X-register with the MSB and the Accumulator with a value of 6 or 7 depending on whether you wanted to set the Immediate or Deferred vectors, respectively. In loading these 3 registers, you then do a JSR to SETVBV at $E45C.

In addition to this, you can also use this same routine to load many LSB/MSB register addresses/vectors. The table below shows you what vectors/addresses you can change, depending on the value you load into the Accumulator. Remember, though, that the Y-register is always the LSB and the X-register is always the MSB.

Accum. Value (hex)	Vector/ Address (hex)	Name	Description
00	216,217	VIMIRQ	IRQ Immediate vector
01	218,219	CDTMV1	Software Timer-1 value
02	21A,21B	CDTMV2	" " 2 "
03	21C,21D	CDTMV3	" " 3 "
04	21E,21F	CDTMV4	" " 4 "
05	220,221	CDTMV5	" " 5 "
06	222,223	VVBLKI	VBI Immediate vector
07	224,225	VVBLKD	" Deferred "
08	226,227	CDTMA1	Software Timer-1 JMP
address			
09	228,229	CDTMA2	" " 2 " "
0D	230,231	SDLSTL	Display List address
10	236,237	BRKKY	Break-key IRQ vector

The list really does go on for a fair distance...

APPENDIX E

APPENDIX E1:

XL/XE ENHANCEMENTS AND BUGS:

First the good news.
The XL computers fixed several bugs in the older Atari' and added many enhancements including relocatable handlers, new poll and extra graphics modes from Basic.
Now, the OS inserts an EOL character in the printer buffer if there isn't one already when you CLOSE the device. You don't have to force out the last characters in the buffer. Printer numbers P1 - P8 are also, now accessible.
When reading a record that's too long or one that is truncated with an EOF character, the OS inserts an EOL character into the input buffer to provide at least, as much as the buffer can handle without an error, so data isn't lost.
Note that, the cassette handling mechanics have also been greatly improved by a change in timing values.

Now the bad news.
If you have the older XL', usually the ones with the flatter keyboard, you may have the revision B ROM. If you PEEK(43234) and get 96, then you sure have B ROMS. B is for BUGged rotten, just write to Atari and ask for the C ROM replacement.
Here are some of the bugs by Matt Ratcliff: First, Basic appends 16 useless bytes to the end of a file on saving. This is a cumalative process; each time you load and save the same program, another 16 useless bytes are appended to the file. This can cause severe problems and errors like 164-truncated record. Make sure you have a blank DOS disk, and try this:

```
10 ? FRE(0):SAVE "D:JUNK":RUN"D:JUNK"
```

Repeat it over and watch your memory dwindle away, 16 bytes at a time! Eventually, the system will crash.

Now try this: Type CSAVE (even if you've not got a cassette unit) and turn up your TV volume. Press Return after the beeps and you'll hear the CSAVE tones. When the READY prompt re-appears, pump up the volume a little more. Hear that!? AAGH! It's the sound of the cassette load still on. You'll have to type END or SOUND 0,0,0,0 to get rid of it. CLOAD has the same problem. This is a bug in both ROMS, not just the B ROM.
Another problem is the unaccountable error 9-string not DIMed, occuring on the line where the DIM statement resides!
When you do too many loads, saves or even use the editor generally 'too much', your system will lock-up. Known as THE dreaded lock-up, this problem was a right pain up the rear, which is thankfully no-longer. Don't suffer! Just send to Atari for the new ROM.
All XE computers use the C ROMS, so there's no worries with these.

APPENDIX E2:

CHANGING A RAM BASED OS:

When you boot the translator-disk, use one of the commercial 'fix' disks such like FIX XL, use XL BOSS, or use Matt Ratcliff' "ROM OS to RAM OS" at the back of this book, you turn your XL/XE RAM-OS to an old 400/800 RAM-OS Revision B. When you run the ROM-RAM OS at location 54017, you turn your XL/XE ROM-OS to XL/XE RAM-OS. This appendix shows you a few changes you can make to both versions of the OS (also Rev.A where stated) when RAM. The labels "4/800" and "XL/XE" denote which RAM OS the selective changes are meant for.
If you have the hardware for blowing PROMS or EPROMS, you can make these changes permanent and replace the original chips in the board.

50104	C3B8	...	XL/XE

This is the initial value loaded down into CHACT at 755, you can change the value mainly to give the cursor a different format; invisible, opaque, solid etc..

50109	C3BD	...	XL/XE

This is the initial value loaded down into CHBASE at 756. Originally 224, which is the standard character-set, CHARSET1 below, you can put 204 here to point to the international character-set, CHARSET2.

52224	CC00	CHARSET2	XL/XE
57344	E000	CHARSET1	4/800 & XL/XE

CHARSET1 can be altered in both OSs, you can save 1K by changing it here because you don't have to reserve 1K of memory in normal RAM. In XL/XEs, you can also change the international character-set at 52224, saving you an additional 1K of memory.

50052	C384	...	XL/XE
59497	E869	...	4/800

The interval for the keyboard repeat. The original value is 6. Increase the cursor speed by lowering the value.

60294	EB86	...	4/800

You can increase the old cassette baud rate by almost a 3rd and reduce the leader time from 20 to 10 seconds by POKEing the following:

Addr:	Dec:	Comment:
60294/EB84	0	lo-byte, write baud
60299/EB8B	4	hi-byte
61250/EF42	0	lo-byte, baud write init routine
61255/EF47	4	hi-byte
61346/EFA2	0	baud-rate open routine
61351/EFA7	4	hi-byte
61371/EFBB	2	leader time

APPENDIX E2:

The XL/XEs already have their cassette routines considerably improved.

61683-61708	F0F3-F10C	...	4/800

Memo-pad mode startup message; "ATARI COMPUTER - MEMOPAD", followed with a carriage return character.

50237-50247	C43D-C447	...	XL/XE
61709-61719	F10D-F117	...	4/800

The "BOOT ERROR" message. Followed with Carriage Return.

50029	C36D	...	XL/XE
61812	F174	...	4/800

Initial value (2) which is loaded down to LMARGN at 82 for the left margin. The 4/800 location is for A and B revisions.

50033	C371	...	XL/XE
61816	F178	...	4/800

Initial right margin value of 39, loaded down to location 83. 4/800 is both A and B revisions.

63878-63880	F986-F988	...	XL/XE
63227-63229	F6FB-F6FD	...	4/800

If you POKE these 3 locations with 234, then you will disable the keyboard click and bell-buzzer. If you only want to disable the bell-buzzer on the XL/XE, then you can POKE locations 62808 - 62810 with 234.

64264-64268	FB08-FB0C	...	XL/XE
65217-65221	FEC1-FEC5	...	4/800

Default colour value tables upon startup. These values are moved to shadow registers 708 - 712 on power-up or Reset. Screen startup is blue; to change this to black, then POKE 65219 with 0.

64337	FB51	...	XL/XE
65278	FEFE	...	4/800

The keyboard definition table begins here. You can re-direct your own in the XL/XE series at locations 121 and 122. A nice trick that Ian Chadwick pointed out in Mapping, is that you can change the arrow keys so that they work without the use of the Control-key. Math signs with shift, and the remaining combinations work with Control. You can do this with:

```
10 FOR I=0 TO 5
20 READ A,D1,D2
30 POKE A,D1:POKE A+1,D2
40 NEXT I
50 DATA 64343,30,31,64351,28,29
60 DATA 64407,43,42,64415,45,61
70 DATA 64481,92,94,64489,95,124
```

Figures given are for XL/XE. 4/800 B-Roms will have to replace the locations above with: 65284, 65292, 65348, 65356, 65412 and 65420 accordingly.

65281 FF01 ... 1200XL

Owners of this XL can turn their function keys into cursor keys by POKEing 65281 with 30, 65282 with 31, 65297 with 28 and 65298 with 11.

62815 F55F ... XL/XE

Normally this would be the cursor to bottom left corner of the screen routine, which isn't on the keyboard. If you use my program at locations 121 and 122 then you can get this, but here's how to change it to a Character-set toggler, which 6/800XL users will find a treat:

```
100 POKE 62815,76:POKE 62816,159
102 POKE 62817,228
104 FOR I=0 TO 23
106 READ D:POKE 58527+I,D:NEXT I
108 DATA 173,158,228,240,8,169,204
110 DATA 206,158,228,76,177,228
112 DATA 238,158,228,169,224
114 DATA 141,244,2,76,12,249
116 POKE 58526,0
```

All you need to do is to type the program at locations 121 and 122, then add this routine. The program uses a small 'unused' patch of memory in the OS itself for storage of the machine-code routine.

65487 FFCF ... XL/XE

You can make the HELP-key act as a start/stop flag like the Control-"1" keypress by POKEing here with 17. See location 732.

65507 FFE3 ... 4/800

The time delay for the keyboard repeat feature; initially 3, POKE with 1 for full-speed ahead.

50056 C388 ... XL/XE
65516 FFEC ... 4/800

Key repeat delay. Initially 48, or 40 depending on your system being NTSC or PAL. Lower the value, the faster.

APPENDIX E3:

130XE MEMORY MANAGEMENT.

Owning the 130XE, you will know that you have an additional 64K in your machine, so as you turn your computer on and type in Basic; ? FRE(0), you would probably expect to see 103438 returned. But you dont! OK, so where is this extra 64K?

Well, take a look at the diagram below:

```
                  ADDRESS
  MAIN 64K  dec        hex   2ND  64K
 ---------- 0       $0000 ----------
 |        |                |        |
 |1st Bank|                |1st Bank|
 |        |                |        |
 ---------- 16384 $4000 ----------   Note:
 |        |                |        |   The 2nd Bank of the
 |2nd Bank|                |2nd Bank|   MAIN 64K can be
referred
 |        |                |        |   to as a 'Window'. It
can
 ---------- 32768 $8000 ----------   be used to access any
 |        |                |        |   of the 16K Banks in
 |3rd Bank|                |3rd Bank|   the 2ND 64K.
 |        |                |        |   Explained afterwards
 ---------- 49152 $C000 ----------
 |        |                |        |
 |4th Bank|                |4th Bank|
 |        |                |        |
 ---------- 65535 $FFFF ----------
```

The memory is divided up into 4 16K banks for both 64K groups inside your 130XE. I've given both the decimal and hexadecimal addresses at which each of the 16K banks begin.

Now then, if I told you that it is only possible to have 'full' access to 64K at any one time, then you would probably assume that you can either use the main 64K OR the 2nd 64K. Well, you're right, and to do this you would set or clear both bits 4 and 5 (decimal 8 and 16) at the bank-select location 54017; $D301, depending on whether you wanted the MAIN 64K, or the 2ND 64K, respectively.

```
BIT:   7   6    5   4    3   2   1   0      GROUP:
DEC:   -n/a-   32  16    -----n/a-----

                1   1                        MAIN 64K
                0   0                        2ND 64K
```

In addition to this method of accessing the extra 64K, you can also retain the MAIN 64K, but access an additional 16K from the 2ND 64K via a 'window' in the MAIN 64K at Bank 2. There are a few complications using this method, however.

APPENDIX E3:

See the table below:

BIT:	7 6	5	4	3	2	1 0	2ND 64K	
DEC:	-n/a-	32	16	8	4	-n/a-	BANK No:	USE:
		0	1	0	0		1	ANTIC
				0	1		2	
				1	0		3	
				1	1		4	
		1	0	repeat				CPU

Bits 2 and 3 simply denote which 16K bank from the 2ND 64K is accessed via the window in the MAIN 64K. But, if this bank replaces the original bank, then how on earth is the original bank accessed at the same time!? Easy! Bits 4 and 5 are the important bits: If bit 4 is set and bit 5 is clear then it means that the main 16K which is supposed to be in this area IS still in this area, BUT only accessible to the CPU! In this same case, the 16K bank selected from the 2ND 64K is only accessible to ANTIC! Of course, were bits 4 and 5 reversed, then access would also reverse.

So, now we know that we can have 2 banks accessed from the 'window' (address \$4000 - \$7FFF in the MAIN 64K), what does this CPU/ANTIC individual access complication mean!?

Put simply, the ANTIC chip is responsible for all the graphics you see on the screen, it uses a technique known as DMA (Direct Memory Access) to process any instructions, from its own instruction-set, to create the screen display. The CPU, on the other hand, directly accesses memory for everything else except graphics. So, as an example, if you set bit 4, clear bit 5 and set both bits 2 and 3, then you would be able to use the original 16K bank for standard program or data memory, but the 2ND 64K's 16K bank (no. 4 in this case) will only be accessed by the ANTIC chip, thus, using this memory for Display Lists and Display Memory.

Of course, this ANTIC memory would have to be loaded or POKEd with the necessary information in the first place, so you would have to give it CPU access so that you can fill it with what you want, and then when your program uses this memory for DL's and DM, you should then set the bits so that ANTIC can access it.

Having this extra memory is a good thing, but as you can see, it can become tedious some times. I'm sure that after a while it will all come easy.

APPENDIX E4:

OS 2.5 Memory assignment.

Unlike the earlier DOS 2.0, the better version now has a very different memory layout, and the code itself is not the same in many areas. The information in the map between memory locations 1792; $700 and 8191; $1FFF is for DOS 2.0. If you're a DOS 2.5 user, then the correct addresses are in this appendix.

1801	709	SABYTE

Maximum files that can be open simultaneously. Same as DOS 2.0.

1802	70A	DRVBYT

Maximum drives allowable in system. Same as DOS 2.0.

1804,5	70C,D	SASA

Buffer allocation address for drives and files.

1806	70E	DFSFLG

Reads 0 if there isn't a DOS.SYS file on the disk.

1807,8	70F,710	DFLINK

Pointer to the 1st sector of DOS.SYS.

1809	711	BLDISP

The number of displacement bytes to sector link bytes (the last 3 of each sector), which should read 125. In true double density DOS's, this byte would read 253.

1810,1	712,3	DFLADDR

Address of the FMS (D:) handler table at 1995; $7CB.

1812	714	XBCONT

The beginning of the boot program.

1900	76C	BSIO

FMS sector I/O routines.

1906	772	BSIOR

FMS disk handler routines.

APPENDIX E4:

1913	779	...

Write verify flag; POKE with 80 to disable verify, thus speeding up all write operations. Engage write verify by POKEing with 87.

1981	7BD	DFMSTA

STATUS routines.

1995	7CB	DFMSDH

FMS handler table. The handler table occupies the same memory as DOS 2.0, but the handlers themselves are now at different addresses, as below:

OPEN	2149; $865
CLOSE	2704; $A90
GET	2638; $A4E
PUT	2448; $990
STATUS	1981; $7BD

2016	7E0	DINIT

DOS initialization routine.

2149	865	DFMOPN

The new address for the OPEN routines.

2448	990	DFMPUT

The PUT routines.

2638	A4E	DFMGET

GET routines.

2704	A90	DFMCLS

IOCB CLOSE routines.

2859	B2B	DFMDDC

Device dependent command routines, including Basic XIO special commands.

2904	B58	INVCMD

Invalid command routines.

2213	8A5	WTBUR

Burst I/O routines (?).

3129 C39 XRENAME

RENAME routines.

3237 CA5 XDELETE

DELETE routines.

3296 CE0 XLOCK

LOCK file routines entry.

3299 CE3 XUNLOCK

UNLOCK file routines entry.

3346 D12 XPOINT

BASIC POINT command routines.

3421 D5D XNOTE

BASIC NOTE command routines.

3442 D72 XFORMAT

FORMAT disk routines.

3501 DAD LISTDIR

Disk directory routines.

3544 DD8 ...

"FREE SECTORS" message.

3709 E7D FNDCODE

Filename decode routines, including 'wildcard' validity tests. The current filename is pointed to by locations 67 and 68.

3747 EA3 ...

This is DOS 2.5's address of the validation check of the "*" wildcard. You can change the wildcard by putting the ASCII code of the character you want to replace it right here.

3760 EB0 ...

This is the other wildcard ("?"). You can change it in the same way as you do at location 3747.

3774,3778 EBE,EC2 ...

This is DOS 2.5's low/hi character acceptance range for filenames. You can POKE 3774 with 33 and 3778 with 123 to allow the use of punctuation and lowercase characters in your filenames.

3799 ED7 ...

By POKEing this location with 0, you can force DOS to accept any character from the filename character range in the initial character of the filename, you needn't begin with an alpha (A - Z). DOS 2.0's equivalent is by POKEing 3828; $EF4 with 4.

3732 E94 ...

This is the full-stop field separation character code. DOS 2.0's location is 3798; $ED6.

3810 EE2 ...

The 'space' character prevented from being in filenames.

3820 EEC SFDIR

Directory search routines; search for the user specified filename.

3872 F20 ...

When a disk directory has been read and displayed to the screen, the way in which DOS knows it has reached the end of all files to be displayed is either due to the fact that all 64 files have been read, or when it reaches an unused entry (all 0's). Occasionally, some programmers write messages in the directory sectors and they put the filenames after an 'unused entry', thus preventing them being displayed. To overcome this and display EVERY directory entry, POKE here with 0. DOS 2.0 users should POKE 3925; $F55 with 5.

3874 F22 ...

A handy little technique is being able to load deleted files. This is only possible if the sectors they pre-occupied haven't been overwritten by recent files saved on the disk. You can do this by POKEing here with 0. The DOS 2.0 equivalent is location 3927; $F57.

3952 F70 WRTNXS

Write data sector routines.

4066 FE2 RDNXTS

Read data sector routines.

4161 1041 RDDIR

Read and write directory sector routines.

4180 1054 RDVTOC

Read and write the volume table of contents (VTOC) sectors.

4365 110D FRESECT

Free sector(s) routines; returns the number of free sectors on a disk that are user accessible.

4426 114A GETSECTOR

Get sector routines; gets a free/unused sector for use.

4521 11A9 SETUP

Setup and initialization of the FMS parameters which basically prepares FMS to deal with the operation asked by the user.

4626 1212 ?

Data sector I/O.

4639 121F WRTDOS

Write new DOS and DUP files to disk routines.

4738 1282 ?

Test DOS.SYS filename.

4762 129A ...

"DOS.SYS" CR (Carriage Return) name.

4945 1351 FCB

Start of the FMS File Control Blocks. Mapping says that these FCB's begin at 4993, but I seem to find that they start here. There are 8 FCBs, each being 16 bytes in size. For a full description of these, refer to the old memory locations in the map.

5121 1401 FILDIR

128 byte buffer for a disk directory sector.

5361 14F1 ...

"D:RAMDISK.COM" CR name.

5439 153F ...

POKE with 49 (ASCII for "1") to re-route DOS to call the DUP.SYS file from drive-1 (D1:) instead of D8: when using the RAMDISK. You can then delete the DUP.SYS and MEM.SAV files from the RAMDISK for extra RAM.

5440 1540 MINIDUP

Beginning of permanently resident portion of the DUP.SYS file.

5446,5450 1546,154A ...

The values here are loaded down into DOSVEC (locations 10 and 11; $A and $B) upon pressing RESET. See relating locations for further details.

5540 15A4 SFLOAD

Mapping states this to be the entry point to the DUP.SYS binary-file load routine, but I find the disassembly in this area of DOS 2.5 is exactly the same as that in DOS 2.0 where it is described to be the routines to load a MEM.SAV file if it exists. I leave you to have your own beliefs, but I believe it is the latter.

5899 170B MEMLDD

Flags that the MEM.SAV file has been loaded. 0 means nope.

5900 170C ...

"D1:AUTORUN.SYS" CR name. This is the filename DOS executes in finding it on a disk. Of course, you can change this to any name you wish.

5915 171B ...

"NEED MEMSAVE TO LOAD THIS FILE" CR prompt.

5947 173B ...

The MEM.SAV file creation routines begins here. The immediate 11 bytes are "D1:MEM.SAV" CR.

6044,5 179C,D INISAV

DOSINI (locations 12 and 13) vector save location which is the entry point to DOS on exit from BASIC.

6046 179E MEMFLG

Flag to show if memory has been saved to disk using the MEM.SAV file.

6191 182F ...

"D1:DUP.SYS" CR name. The utility package DOS searches for on the disk when DOS is typed in BASIC. The DUP.SYS file is a normal binary-load file which has control passed to it after being loaded.

6202 183A ...

"ERROR-SAVING USER MEMORY ON DISK" CR prompt.

6235 185B ...

"TYPE Y TO RUN DOS" CR prompt.

6418 1912 CLMJMP

Test to see if DOS should load MEM.SAV prior to it executing a 'run-at-cartridge' address.

6432 1920 LMTR

Test to see if the MEM.SAV file should be loaded before a 'run-at-address' is executed.

6457 1939 LDMEM

MEM.SAV load routines.

6518 1979 INITIO

DUP.SYS warmstart entry.

7276 1C6C ...

In a standard DOS 2.5, this is where MEMLO normally points to when DOS is resident. See location 7420; $1CFC in the DOS 2.0 map for full descriptions.

Well, that's about as much as I could work out of DOS 2.5. It's pretty difficult when you don't have the source listing! Before I bring this appendix to an end, here's a handy way of finding out what DOS has loaded within your programs:

PEEK location	Value returned	DOS version
3889	0	SpartaDOS 2.3e
	13	DOS 4.0
	15	SpartaDOS 1.1
	19	DOS 2.5
	76	DOS 3.0
	78	" "
	89	SpartaDOS 3.2d

APPENDIX E4:

108	MYDOS 4.0
207	OSS OS/A+4.00
221	MYDOS 4.50
238	DOS 2.0
238	OSS DOS XL 2.3
244	DOS XE

You'll notice that DOS 2.0 and OSS DOS XL 2.3 have the same values, to seperate the 2 then just check this location:

1804	0	OSS DOS XL 2.3
	124	DOS 2.0

You can thank Dave Ewens of TWAUG for these handy tips because that's where they came from. Issue #5 of TWAUG newsletter to be exact.

APPENDIX E5:

FREE BYTES.

For quick and easy reference, here's a list of all the unused bytes inside your machine.

0-1 0-1

Free for use.

28-31 1C-1F

Free for use.

128-202 80-CA

Free outside of Basic. If you are in Basic, then you only get location 147 free.

203-209 CB-41

Always free except in the Assembler/Editor, where locations 203-207 are then unusable.

212-255 D4-FF

Free for non-Basic users if you don't use the Floating Point package.

583-618 247-26A

Free for 1200XL users.

590-618 24E-26A

Unused.

APPENDIX E5:

653 26D

Unconditionally Free.

704-707 2C0-2C3

Free if not using PMG's.

711 2C7

Free for use, except when in 5-colour modes.

736-739 2E0,2E3

Free if not using DOS.

775 307

Always free for use.

794-808 31A-328

Depending on which handlers you don't use in your program, then 3 bytes are free for each handler not used.

809-826 329-33A

Always free unless used for additional handlers. If using DOS, then avoid 809-811; $329-$32B.

827-828 33B-33C

Always free.

829-831 33D-33F

OK to use except if you press RESET. You should replace the original values if RESET should be pressed, since the system will coldstart otherwise.

832-959 340-3BF

Very tricky, especially using Basic. Except for IOCB's 0, 6 and 7, the rest can be completely free for use. The used IOCB's allow free use outside Basic, but in Basic: IOCB-0 is only free outside typical Graphics 0 operations such as PRINT, LIST etc.. IOCB-6 is free only outside Graphics commands such as PLOT, PRINT #6 etc., while IOCB-7 is free only outside device I/O commands such as LPRINT, LOAD etc..

960-999 3C0-3E7

Without using a printer, you get these 40 bytes free

1001 3E9

Free for use without booting the cassette.

1002 3EA

Free except when booting the cassette or disk via the OS routines.

1003 3EB

Always free, except in the 1200XL. To free its use in the 1200XL, then ommit its use in the VBLANK.

1021-1151 3FD-47F

Always free except for the initial booted sector of a disk and all cassette records being loaded.

1152-1535 480-5FF

Other than using Basic, this area is OK to use. If you have any applications/utilities loaded into memory, this is the low-memory area that they most often occupy.

1536-1791 600-6FF

Always free for your use, and even in Turbo Basic. Originally, I thought TB stopped you using this area, but this is not so. You can.

1792-MEMLO 700-MEMLO

MEMLO is the address at locations 743 and 744. This area is free except when using DOS and some other programming environments. When using any Basic without DOS, your programs occupy this area. When using DOS, your Basic programs occupy memory from MEMLO upwards. MEMLO is usually kept below 8192; $2000 with most DOS's.

8192-32767 2000-7FFF

Again, Basic programs occupy this memory depending on their size. If you've exited Basic to the DOS you're using, then most DOS's Utility Packages (DUP) occupy 8192-16384; $2000-$4000. The top end varies, but they never usually exceed the address given.

32768-40959 8000-9FFF

Using Basic, this is display memory. The amount of memory used depends on the mode in use. The memory being occupied always takes the higher end of this block, which is pointed to by locations 88 and 89; $58 and $59. If you're not using Basic, then this area is unsused and free for your use. Most hi-memory menu's and utilities occupy the higher area of this block, so be careful of conflict.

APPENDIX E5:

40960-49151   A000-BFFF

Always used by Basic. If you're out of Basic, then this area is occupied by the display mode in the same way as addresses 32768-40959. Most cartridges use this area including the Assembler/Editor. If you have a 16K cartridge inserted then the lower 8K is also used.

49152-53247   C000-CFFF
57344-65535   E000-FFFF

Both these areas can be turned into total RAM, though, not all of it can be used as such. It depends on your application. See location 54017; $D301.

Well, there you have the obvious memory free for your use. Besides this, there is much more memory that your programs can use, it all depends on what your programs don't need to use. Have fun!

APPENDIX E6:

THE XL/XE OS-SOURCE LISTING:

I was searching through my utilities for a program to disassemble the computers OS to make this appendix, but could I find one!? Could I heck! I found programs to disassemble to screen and printer but not to disk and all of them were heavily protected you could only RUN the program straight from disk! If I could've LISTed it, then I would have sent the output to a disk-file instead of the printer. Anyway, it worked out I had to write my own disassembler. I wrote it in Turbo Basic and then found that the OS-ROM was different to what it normally is! What a pain. So, I then was forced to convert my Turbo Basic program into normal Basic, which required some additional routines for DEC to HEX conversions etc. because I was previously using T/Basics HEX$ command for the conversion.

Anyway, here it is after all that unexpected trouble! The Operating System Source listing for XL/XE machines. Have fun!

NOTE: You will find the author comments below the appropriate lines, preceded by an upper case enclosed in brackets (C). The reason is to print the Source Code Listings in double column to save on paper and cost.

```
C000  11 92
(C) LSB/MSB ROM Checksum
C002  10
(C) Revision date in
C003  05
(C) form: DDMMYY
C004  83
C005  00
(C) Reserved option byte
C006  42
(C) Part-Number in
    form: AANNNNNN
    AA is ASCII Char and
    NNNNNN is a 4-bit BCD
    digit (A1)
C007  42 00 00 01
(C) A2 and N1-N6 where
    each N is 2 4-bit
    BCD values
C00B  02
(C) Revision Number

C00C  A9 40        LDA #$40
(C) INTERRUPT HANDLER
C00E  8D 0E D4     STA  $D40E
(C) INITIALIZATION
C011  AD 13 D0     LDA  $D013
C014  8D FA 03     STA  $03FA
C017  60           RTS
C018  2C 0F D4     BIT  $D40F
(C) NMI INITIALIZATION
C01B  10 03        BPL  $C020
C01D  6C 00 02     JMP ($0200)
C020  D8           CLD
C021  48           PHA
C022  8A           TXA
C023  48           PHA
C024  98           TYA
C025  48           PHA
C026  8D 0F D4     STA  $D40F
C029  6C 22 02     JMP ($0222)
C02C  D8           CLD
(C) IRQ PROCESSOR
C02D  6C 16 02     JMP ($0216)
C030  48           PHA
C031  AD 0E D2     LDA  $D20E
C034  29 20        AND #$20
C036  D0 0D        BNE  $C045
C038  A9 DF        LDA #$DF
C03A  8D 0E D2     STA  $D20E
C03D  A5 10        LDA  $10
C03F  8D 0E D2     STA  $D20E
C042  6C 0A 02     JMP ($020A)
C045  8A           TXA
C046  48           PHA
C047  AD FF D1     LDA  $D1FF
C04A  2D 49 02     AND  $0249
C04D  F0 03        BEQ  $C052
C04F  6C 38 02     JMP ($0238)
C052  A2 06        LDX #$06
C054  BD CF C0     LDA  $C0CF,X
C057  E0 05        CPX #$05
C059  D0 04        BNE  $C05F
C05B  25 10        AND  $10
C05D  F0 05        BEQ  $C064
C05F  2C 0E D2     BIT  $D20E
C062  F0 06        BEQ  $C06A
C064  CA           DEX
C065  10 ED        BPL  $C054
C067  4C A0 C0     JMP  $C0A0
C06A  49 FF        EOR #$FF
C06C  8D 0E D2     STA  $D20E
C06F  A5 10        LDA  $10
C071  8D 0E D2     STA  $D20E
C074  E0 00        CPX #$00
C076  D0 05        BNE  $C07D
C078  AD 6D 02     LDA  $026D
C07B  D0 23        BNE  $C0A0
C07D  BD D7 C0     LDA  $C0D7,X
C080  AA           TAX
C081  BD 00 02     LDA  $0200,X
C084  8D 8C 02     STA  $028C
C087  BD 01 02     LDA  $0201,X
C08A  8D 8D 02     STA  $028D
C08D  68           PLA
C08E  AA           TAX
C08F  6C 8C 02     JMP ($028C)
C092  A9 00        LDA #$00
(C) BREAK-KEY IRQ
C094  85 11        STA  $11
C096  8D FF 02     STA  $02FF
C099  8D F0 02     STA  $02F0
C09C  85 4D        STA  $4D
C09E  68           PLA
C09F  40           RTI
C0A0  68           PLA
(C) CONTINUE IRQ
C0A1  AA           TAX
(C) PROCESSING
C0A2  2C 02 D3     BIT  $D302
C0A5  10 06        BPL  $C0AD
C0A7  AD 00 D3     LDA  $D300
C0AA  6C 02 02     JMP ($0202)
C0AD  2C 03 D3     BIT  $D303
C0B0  10 06        BPL  $C0B8
C0B2  AD 01 D3     LDA  $D301
C0B5  6C 04 02     JMP ($0204)
C0B8  68           PLA
C0B9  8D 8C 02     STA  $028C
C0BC  68           PLA
```

```
C0BD  48           PHA
C0BE  29 10        AND #$10
C0C0  F0 07        BEQ  $C0C9
C0C2  AD 8C 02     LDA  $028C
C0C5  48           PHA
C0C6  6C 06 02     JMP ($0206)
C0C9  AD 8C 02     LDA  $028C
C0CC  48           PHA
C0CD  68           PLA
C0CE  40           RTI

C0CF  80 40 04 02 01 08 10 20
(C) TABLE OF IRQ TYPES
C0D7  36 08 14 12 10 0E 0C 0A
(C) AND OFFSETS

C0DF  4C DF C0     JMP  $C0DF
C0E2  E6 14        INC  $14
(C) IMMEDIATE VBLANK NMI
C0E4  D0 08        BNE  $C0EE
(C) PROCESSING
C0E6  E6 4D        INC  $4D
C0E8  E6 13        INC  $13
C0EA  D0 02        BNE  $C0EE
C0EC  E6 12        INC  $12
C0EE  A9 FE        LDA #$FE
C0F0  A2 00        LDX #$00
C0F2  A4 4D        LDY  $4D
C0F4  10 06        BPL  $C0FC
C0F6  85 4D        STA  $4D
C0F8  A6 13        LDX  $13
C0FA  A9 F6        LDA #$F6
C0FC  85 4E        STA  $4E
C0FE  86 4F        STX  $4F
C100  AD C5 02     LDA  $02C5
C103  45 4F        EOR  $4F
C105  25 4E        AND  $4E
C107  8D 17 D0     STA  $D017
C10A  A2 00        LDX #$00
C10C  20 55 C2     JSR  $C255
C10F  D0 03        BNE  $C114
C111  20 4F C2     JSR  $C24F
C114  A5 42        LDA  $42
C116  D0 08        BNE  $C120
C118  BA           TSX
C119  BD 04 01     LDA  $0104,X
C11C  29 04        AND #$04
C11E  F0 03        BEQ  $C123
C120  4C 8A C2     JMP  $C28A
C123  AD 13 D0     LDA  $D013
C126  CD FA 03     CMP  $03FA
C129  D0 B4        BNE  $C0DF
C12B  AD 0D D4     LDA  $D40D
C12E  8D 35 02     STA  $0235
C131  AD 0C D4     LDA  $D40C
C134  8D 34 02     STA  $0234
C137  AD 31 02     LDA  $0231
C13A  8D 03 D4     STA  $D403
C13D  AD 30 02     LDA  $0230
C140  8D 02 D4     STA  $D402
C143  AD 2F 02     LDA  $022F
C146  8D 00 D4     STA  $D400
C149  AD 6F 02     LDA  $026F
C14C  8D 1B D0     STA  $D01B
C14F  AD 6C 02     LDA  $026C
C152  F0 0E        BEQ  $C162
C154  CE 6C 02     DEC  $026C
C157  A9 08        LDA #$08
C159  38           SEC
C15A  ED 6C 02     SBC  $026C
C15D  29 07        AND #$07
C15F  8D 05 D4     STA  $D405
C162  A2 08        LDX #$08
C164  8E 1F D0     STX  $D01F
C167  58           CLI
C168  BD C0 02     LDA  $02C0,X
C16B  45 4F        EOR  $4F
C16D  25 4E        AND  $4E
C16F  9D 12 D0     STA  $D012,X
C172  CA           DEX
C173  10 F2        BPL  $C167
C175  AD F4 02     LDA  $02F4
C178  8D 09 D4     STA  $D409
C17B  AD F3 02     LDA  $02F3
C17E  8D 01 D4     STA  $D401
C181  A2 02        LDX #$02
C183  20 55 C2     JSR  $C255
C186  D0 03        BNE  $C18B
C188  20 52 C2     JSR  $C252
C18B  A2 02        LDX #$02
C18D  E8           INX
C18E  E8           INX
C18F  BD 18 02     LDA  $0218,X
C192  1D 19 02     ORA  $0219,X
C195  F0 06        BEQ  $C19D
C197  20 55 C2     JSR  $C255
C19A  9D 26 02     STA  $0226,X
C19D  E0 08        CPX #$08
C19F  D0 EC        BNE  $C18D
C1A1  AD 0F D2     LDA  $D20F
C1A4  29 04        AND #$04
C1A6  F0 08        BEQ  $C1B0
C1A8  AD F1 02     LDA  $02F1
C1AB  F0 03        BEQ  $C1B0
C1AD  CE F1 02     DEC  $02F1
C1B0  AD 2B 02     LDA  $022B
C1B3  F0 3E        BEQ  $C1F3
C1B5  AD 0F D2     LDA  $D20F
C1B8  29 04        AND #$04
C1BA  D0 32        BNE  $C1EE
```

```
C1BC   CE 2B 02      DEC   $022B
C1BF   D0 32         BNE   $C1F3
C1C1   AD 6D 02      LDA   $026D
C1C4   D0 2D         BNE   $C1F3
C1C6   AD DA 02      LDA   $02DA
C1C9   8D 2B 02      STA   $022B
C1CC   AD 09 D2      LDA   $D209
C1CF   C9 9F         CMP #$9F
C1D1   F0 20         BEQ   $C1F3
C1D3   C9 83         CMP #$83
C1D5   F0 1C         BEQ   $C1F3
C1D7   C9 84         CMP #$84
C1D9   F0 18         BEQ   $C1F3
C1DB   C9 94         CMP #$94
C1DD   F0 14         BEQ   $C1F3
C1DF   29 3F         AND #$3F
C1E1   C9 11         CMP #$11
C1E3   F0 0E         BEQ   $C1F3
C1E5   AD 09 D2      LDA   $D209
C1E8   8D FC 02      STA   $02FC
C1EB   4C F3 C1      JMP   $C1F3
C1EE   A9 00         LDA #$00
C1F0   8D 2B 02      STA   $022B
C1F3   AD 00 D3      LDA   $D300
C1F6   4A            LSR A
C1F7   4A            LSR A
C1F8   4A            LSR A
C1F9   4A            LSR A
C1FA   8D 79 02      STA   $0279
C1FD   8D 7B 02      STA   $027B
C200   AD 00 D3      LDA   $D300
C203   29 0F         AND #$0F
C205   8D 78 02      STA   $0278
C208   8D 7A 02      STA   $027A
C20B   AD 10 D0      LDA   $D010
C20E   8D 84 02      STA   $0284
C211   8D 86 02      STA   $0286
C214   AD 11 D0      LDA   $D011
C217   8D 85 02      STA   $0285
C21A   8D 87 02      STA   $0287
C21D   A2 03         LDX #$03
C21F   BD 00 D2      LDA   $D200,X
C222   9D 70 02      STA   $0270,X
C225   9D 74 02      STA   $0274,X
C228   CA            DEX
C229   10 F4         BPL   $C21F
C22B   8D 0B D2      STA   $D20B
C22E   A2 02         LDX #$02
C230   A0 01         LDY #$01
C232   B9 78 02      LDA   $0278,Y
C235   4A            LSR A
C236   4A            LSR A
C237   4A            LSR A
C238   9D 7D 02      STA   $027D,X
C23B   9D 81 02      STA   $0281,X
C23E   A9 00         LDA #$00
C240   2A            ROL A
C241   9D 7C 02      STA   $027C,X
C244   9D 80 02      STA   $0280,X
C247   CA            DEX
C248   CA            DEX
C249   88            DEY
C24A   10 E6         BPL   $C232
C24C   6C 24 02      JMP ($0224)
C24F   6C 26 02      JMP ($0226)
(C) TIMER-1 EXPIRED
C252   6C 28 02      JMP ($0228)
(C) TIMER-2 EXPIRED
C255   BC 18 02      LDY   $0218,X
(C) DECREMENT COUNTDOWN
C258   D0 08         BNE   $C262
(C) TIMER
C25A   BC 19 02      LDY   $0219,X
C25D   F0 10         BEQ   $C26F
C25F   DE 19 02      DEC   $0219,X
C262   DE 18 02      DEC   $0218,X
C265   D0 08         BNE   $C26F
C267   BC 19 02      LDY   $0219,X
C26A   D0 03         BNE   $C26F
C26C   A9 00         LDA #$00
C26E   60            RTS
C26F   A9 FF         LDA #$FF
C271   60            RTS
C272   0A            ASL A
(C) SET VBLANK PARAMETERS
C273   8D 2D 02      STA   $022D
C276   8A            TXA
C277   A2 05         LDX #$05
C279   8D 0A D4      STA   $D40A
C27C   CA            DEX
C27D   D0 FD         BNE   $C27C
C27F   AE 2D 02      LDX   $022D
C282   9D 17 02      STA   $0217,X
C285   98            TYA
C286   9D 16 02      STA   $0216,X
C289   60            RTS
C28A   68            PLA
(C) PROCESS DEFERRED
C28B   A8            TAY
(C) VBLANK NMI
C28C   68            PLA
C28D   AA            TAX
C28E   68            PLA
C28F   40            RTI
C290   78            SEI
(C) PERFORM WARMSTART
C291   AD 13 D0      LDA   $D013
C294   CD FA 03      CMP   $03FA
C297   D0 2F         BNE   $C2C8
C299   6A            ROR A
```

```
C29A   90 05      BCC  $C2A1
C29C   20 C9 C4   JSR  $C4C9
C29F   D0 27      BNE  $C2C8
C2A1   AD 44 02   LDA  $0244
C2A4   D0 22      BNE  $C2C8
C2A6   A9 FF      LDA #$FF
C2A8   D0 20      BNE  $C2CA
C2AA   78         SEI
(C) PROCESS RESET
C2AB   A2 8C      LDX #$8C
C2AD   88         DEY
C2AE   D0 FD      BNE  $C2AD
C2B0   CA         DEX
C2B1   D0 FA      BNE  $C2AD
C2B3   AD 3D 03   LDA  $033D
C2B6   C9 5C      CMP #$5C
C2B8   D0 0E      BNE  $C2C8
C2BA   AD 3E 03   LDA  $033E
C2BD   C9 93      CMP #$93
C2BF   D0 07      BNE  $C2C8
C2C1   AD 3F 03   LDA  $033F
C2C4   C9 25      CMP #$25
C2C6   F0 C8      BEQ  $C290
C2C8   A9 00      LDA #$00
(C) PERFORM COLDSTART
C2CA   85 08      STA  $08
(C) PRESET MEMORY;
C2CC   78         SEI
(C) COLD/WARM START
C2CD   D8         CLD
(C) CONTINUATION
C2CE   A2 FF      LDX #$FF
C2D0   9A         TXS
C2D1   20 71 C4   JSR  $C471
C2D4   A9 01      LDA #$01
C2D6   85 01      STA  $01
C2D8   A5 08      LDA  $08
C2DA   D0 52      BNE  $C32E
C2DC   A9 00      LDA #$00
C2DE   A0 08      LDY #$08
C2E0   85 04      STA  $04
C2E2   85 05      STA  $05
C2E4   A9 FF      LDA #$FF
C2E6   91 04      STA ($04),Y
C2E8   D1 04      CMP ($04),Y
C2EA   F0 02      BEQ  $C2EE
C2EC   46 01      LSR  $01
C2EE   A9 00      LDA #$00
C2F0   91 04      STA ($04),Y
C2F2   D1 04      CMP ($04),Y
C2F4   F0 02      BEQ  $C2F8
C2F6   46 01      LSR  $01
C2F8   C8         INY
C2F9   D0 E9      BNE  $C2E4
C2FB   E6 05      INC  $05
C2FD   A6 05      LDX  $05
C2FF   E4 06      CPX  $06
C301   D0 E1      BNE  $C2E4
C303   A9 23      LDA #$23
C305   85 0A      STA  $0A
C307   A9 F2      LDA #$F2
C309   85 0B      STA  $0B
C30B   AD 01 D3   LDA  $D301
C30E   29 7F      AND #$7F
C310   8D 01 D3   STA  $D301
C313   20 73 FF   JSR  $FF73
C316   B0 05      BCS  $C31D
C318   20 92 FF   JSR  $FF92
C31B   90 02      BCC  $C31F
C31D   46 01      LSR  $01
C31F   AD 01 D3   LDA  $D301
C322   09 80      ORA #$80
C324   8D 01 D3   STA  $D301
C327   A9 FF      LDA #$FF
C329   8D 44 02   STA  $0244
C32C   D0 22      BNE  $C350
C32E   A2 00      LDX #$00
C330   AD EC 03   LDA  $03EC
C333   F0 07      BEQ  $C33C
C335   8E 0E 00   STX  $000E
C338   8E 0F 00   STX  $000F
C33B   8A         TXA
C33C   9D 00 02   STA  $0200,X
C33F   E0 ED      CPX #$ED
C341   B0 03      BCS  $C346
C343   9D 00 03   STA  $0300,X
C346   CA         DEX
C347   D0 F3      BNE  $C33C
C349   A2 10      LDX #$10
C34B   95 00      STA  $00,X
C34D   E8         INX
C34E   10 FB      BPL  $C34B
C350   A2 00      LDX #$00
C352   AD 01 D3   LDA  $D301
C355   29 02      AND #$02
C357   F0 01      BEQ  $C35A
C359   E8         INX
C35A   8E F8 03   STX  $03F8
C35D   A9 5C      LDA #$5C
C35F   8D 3D 03   STA  $033D
C362   A9 93      LDA #$93
C364   8D 3E 03   STA  $033E
C367   A9 25      LDA #$25
C369   8D 3F 03   STA  $033F
C36C   A9 02      LDA #$02
C36E   85 52      STA  $52
C370   A9 27      LDA #$27
C372   85 53      STA  $53
C374   AD 14 D0   LDA  $D014
C377   29 0E      AND #$0E
```

```
C379   D0 08      BNE  $C383
C37B   A9 05      LDA #$05
C37D   A2 01      LDX #$01
C37F   A0 28      LDY #$28
C381   D0 06      BNE  $C389
C383   A9 06      LDA #$06
C385   A2 00      LDX #$00
C387   A0 30      LDY #$30
C389   8D DA 02   STA  $02DA
C38C   86 62      STX  $62
C38E   8C D9 02   STY  $02D9
C391   A2 25      LDX #$25
C393   BD 4B C4   LDA  $C44B,X
C396   9D 00 02   STA  $0200,X
C399   CA         DEX
C39A   10 F7      BPL  $C393
C39C   A2 0E      LDX #$0E
C39E   BD 2E C4   LDA  $C42E,X
C3A1   9D 1A 03   STA  $031A,X
C3A4   CA         DEX
C3A5   10 F7      BPL  $C39E
C3A7   20 35 C5   JSR  $C535
C3AA   58         CLI
C3AB   A5 01      LDA  $01
C3AD   D0 15      BNE  $C3C4
C3AF   AD 01 D3   LDA  $D301
C3B2   29 7F      AND #$7F
C3B4   8D 01 D3   STA  $D301
C3B7   A9 02      LDA #$02
C3B9   8D F3 02   STA  $02F3
C3BC   A9 E0      LDA #$E0
C3BE   8D F4 02   STA  $02F4
C3C1   4C 03 50   JMP  $5003
C3C4   A2 00      LDX #$00
C3C6   86 06      STX  $06
C3C8   AE E4 02   LDX  $02E4
C3CB   E0 B0      CPX #$B0
C3CD   B0 0D      BCS  $C3DC
C3CF   AE FC BF   LDX  $BFFC
C3D2   D0 08      BNE  $C3DC
C3D4   E6 06      INC  $06
C3D6   20 C9 C4   JSR  $C4C9
C3D9   20 29 C4   JSR  $C429
C3DC   A9 03      LDA #$03
C3DE   A2 00      LDX #$00
C3E0   9D 42 03   STA  $0342,X
C3E3   A9 48      LDA #$48
C3E5   9D 44 03   STA  $0344,X
C3E8   A9 C4      LDA #$C4
C3EA   9D 45 03   STA  $0345,X
C3ED   A9 0C      LDA #$0C
C3EF   9D 4A 03   STA  $034A,X
C3F2   20 56 E4   JSR  $E456
C3F5   10 03      BPL  $C3FA
C3F7   4C AA C2   JMP  $C2AA

C3FA   E8         INX
C3FB   D0 FD      BNE  $C3FA
C3FD   C8         INY
C3FE   10 FA      BPL  $C3FA
C400   20 6E C6   JSR  $C66E
C403   A5 06      LDA  $06
C405   F0 06      BEQ  $C40D
C407   AD FD BF   LDA  $BFFD
C40A   6A         ROR A
C40B   90 06      BCC  $C413
C40D   20 8B C5   JSR  $C58B
C410   20 39 E7   JSR  $E739
C413   A9 00      LDA #$00
C415   8D 44 02   STA  $0244
C418   A5 06      LDA  $06
C41A   F0 0A      BEQ  $C426
C41C   AD FD BF   LDA  $BFFD
C41F   29 04      AND #$04
C421   F0 03      BEQ  $C426
C423   6C FA BF   JMP ($BFFA)
C426   6C 0A 00   JMP ($000A)
C429   6C FE BF   JMP ($BFFE)
(C) INITIALIZE CARTRIDGE
C42C   18         CLC
(C) PROCESS ACMI
C42D   60         RTS
(C) INTERRUPT

C42E   50 30 E4 43 40 E4 45 00
C436   E4 53 10 E4 4B 20 E4

C43D   42 4F 4F 54 20 45 52 52
(C) "BOOT ERROR" (CR)
C445   4F 52 9B

C448   45 3A 9B
(C) E: (CR)

C44B   CE C0
(C) VDSLST VECTOR
C44D   CD C0
(C) VPRCED    "
C44F   CD C0
(C) VINTER    "
C451   CD C0
(C) VBREAK    "
C453   19 FC
(C) VKEYBD    "
C455   2C EB
(C) VSERIN    "
C457   AD EA
(C) VSEROR    "
C459   EC EA
(C) VSEROC    "
```

APPENDIX E6:

```
C45B  CD C0
(C) VTIMR1    "
C45D  CD C0
(C) VTIMR2    "
C45F  CD C0
(C) VTIMR3    "
C461  30 C0
(C) VIMIRQ    "

C463  00 00 00 00 00 00 00 00
C46B  00 00

C46D  E2 C0
(C) VVBLKI    "
C46F  8A C2
(C) VVBLKD    "
C471  AD 13 D0      LDA  $D013
C474  6A            ROR A
C475  90 0D         BCC  $C484
C477  AD FC BF      LDA  $BFFC
C47A  D0 08         BNE  $C484
C47C  AD FD BF      LDA  $BFFD
C47F  10 03         BPL  $C484
C481  6C FE BF      JMP ($BFFE)
(C) INITIALIZE CARTRIDGE
C484  20 DA C4      JSR  $C4DA
C487  AD 01 D3      LDA  $D301
C48A  09 02         ORA #$02
C48C  8D 01 D3      STA  $D301
C48F  A5 08         LDA  $08
C491  F0 07         BEQ  $C49A
C493  AD F8 03      LDA  $03F8
C496  D0 11         BNE  $C4A9
C498  F0 07         BEQ  $C4A1
C49A  AD 1F D0      LDA  $D01F
(C) CHECK OPTION-KEY
C49D  29 04         AND #$04
C49F  F0 08         BEQ  $C4A9
C4A1  AD 01 D3      LDA  $D301
(C) ENABLE BASIC
C4A4  29 FD         AND #$FD
C4A6  8D 01 D3      STA  $D301
C4A9  A9 00         LDA #$00
C4AB  A8            TAY
C4AC  85 05         STA  $05
C4AE  A9 28         LDA #$28
C4B0  85 06         STA  $06
C4B2  B1 05         LDA ($05),Y
C4B4  49 FF         EOR #$FF
C4B6  91 05         STA ($05),Y
C4B8  D1 05         CMP ($05),Y
C4BA  D0 0C         BNE  $C4C8
C4BC  49 FF         EOR #$FF
C4BE  91 05         STA ($05),Y
C4C0  D1 05         CMP ($05),Y
C4C2  D0 04         BNE  $C4C8
C4C4  E6 06         INC  $06
C4C6  D0 EA         BNE  $C4B2
C4C8  60            RTS
C4C9  A9 00         LDA #$00
(C) RAM-TEST & SET
C4CB  AA            TAX
C4CC  18            CLC
C4CD  7D F0 BF      ADC  $BFF0,X
C4D0  E8            INX
C4D1  D0 FA         BNE  $C4CD
C4D3  CD EB 03      CMP  $03EB
C4D6  8D EB 03      STA  $03EB
C4D9  60            RTS
C4DA  A9 00         LDA #$00
(C) INITIALIZE HARDWARE
C4DC  AA            TAX
(C) MEMORY
C4DD  8D 03 D3      STA  $D303
C4E0  9D 00 D0      STA  $D000,X
C4E3  9D 00 D4      STA  $D400,X
C4E6  9D 00 D2      STA  $D200,X
C4E9  E0 01         CPX #$01
C4EB  F0 03         BEQ  $C4F0
C4ED  9D 00 D3      STA  $D300,X
C4F0  E8            INX
C4F1  D0 ED         BNE  $C4E0
C4F3  A9 3C         LDA #$3C
C4F5  8D 03 D3      STA  $D303
C4F8  A9 FF         LDA #$FF
C4FA  8D 01 D3      STA  $D301
C4FD  A9 38         LDA #$38
C4FF  8D 02 D3      STA  $D302
C502  8D 03 D3      STA  $D303
C505  A9 00         LDA #$00
C507  8D 00 D3      STA  $D300
C50A  A9 FF         LDA #$FF
C50C  8D 01 D3      STA  $D301
C50F  A9 3C         LDA #$3C
C511  8D 02 D3      STA  $D302
C514  8D 03 D3      STA  $D303
C517  AD 01 D3      LDA  $D301
C51A  AD 00 D3      LDA  $D300
C51D  A9 22         LDA #$22
C51F  8D 0F D2      STA  $D20F
C522  A9 A0         LDA #$A0
C524  8D 05 D2      STA  $D205
C527  8D 07 D2      STA  $D207
C52A  A9 28         LDA #$28
C52C  8D 08 D2      STA  $D208
C52F  A9 FF         LDA #$FF
C531  8D 0D D2      STA  $D20D
C534  60            RTS
```

```
C535  C6 11         DEC  $11
(C) SOFTWARE & RAM
C537  A9 92         LDA #$92
(C) VARIABLE INITIALIZATION
C539  8D 36 02      STA  $0236
C53C  A9 C0         LDA #$C0
C53E  8D 37 02      STA  $0237
C541  A5 06         LDA  $06
C543  8D E4 02      STA  $02E4
C546  8D E6 02      STA  $02E6
C549  A9 00         LDA #$00
C54B  8D E5 02      STA  $02E5
C54E  A9 00         LDA #$00
C550  8D E7 02      STA  $02E7
C553  A9 07         LDA #$07
C555  8D E8 02      STA  $02E8
C558  20 0C E4      JSR  $E40C
C55B  20 1C E4      JSR  $E41C
C55E  20 2C E4      JSR  $E42C
C561  20 3C E4      JSR  $E43C
C564  20 4C E4      JSR  $E44C
C567  20 6E E4      JSR  $E46E
C56A  20 65 E4      JSR  $E465
C56D  20 6B E4      JSR  $E46B
C570  20 50 E4      JSR  $E450
C573  A9 6E         LDA #$6E
C575  8D 38 02      STA  $0238
C578  A9 C9         LDA #$C9
C57A  8D 39 02      STA  $0239
C57D  20 9B E4      JSR  $E49B
C580  AD 1F D0      LDA  $D01F
C583  29 01         AND #$01
C585  49 01         EOR #$01
C587  8D E9 03      STA  $03E9
C58A  60            RTS
C58B  A5 08         LDA  $08
(C) ATTEMPT DISK-BOOT
C58D  F0 09         BEQ  $C598
C58F  A5 09         LDA  $09
C591  29 01         AND #$01
C593  F0 33         BEQ  $C5C8
C595  4C 3B C6      JMP  $C63B
C598  A9 01         LDA #$01
C59A  8D 01 03      STA  $0301
C59D  A9 53         LDA #$53
C59F  8D 02 03      STA  $0302
C5A2  20 53 E4      JSR  $E453
C5A5  30 21         BMI  $C5C8
C5A7  A9 00         LDA #$00
C5A9  8D 0B 03      STA  $030B
C5AC  A9 01         LDA #$01
C5AE  8D 0A 03      STA  $030A
C5B1  A9 00         LDA #$00
C5B3  8D 04 03      STA  $0304
C5B6  A9 04         LDA #$04
C5B8  8D 05 03      STA  $0305
C5BB  20 59 C6      JSR  $C659
(C) BOOT & INITIALIZE
C5BE  10 09         BPL  $C5C9
(C) DISK
C5C0  20 3E C6      JSR  $C63E
C5C3  AD EA 03      LDA  $03EA
C5C6  F0 DF         BEQ  $C5A7
C5C8  60            RTS
C5C9  A2 03         LDX #$03
(C) COMPLETE BOOT &
C5CB  BD 00 04      LDA  $0400,X
(C) INITIALIZE
C5CE  9D 40 02      STA  $0240,X
C5D1  CA            DEX
C5D2  10 F7         BPL  $C5CB
C5D4  AD 42 02      LDA  $0242
C5D7  85 04         STA  $04
C5D9  AD 43 02      LDA  $0243
C5DC  85 05         STA  $05
C5DE  AD 04 04      LDA  $0404
C5E1  85 0C         STA  $0C
C5E3  AD 05 04      LDA  $0405
C5E6  85 0D         STA  $0D
C5E8  A0 7F         LDY #$7F
C5EA  B9 00 04      LDA  $0400,Y
C5ED  91 04         STA ($04),Y
C5EF  88            DEY
C5F0  10 F8         BPL  $C5EA
C5F2  18            CLC
C5F3  A5 04         LDA  $04
C5F5  69 80         ADC #$80
C5F7  85 04         STA  $04
C5F9  A5 05         LDA  $05
C5FB  69 00         ADC #$00
C5FD  85 05         STA  $05
C5FF  CE 41 02      DEC  $0241
C602  F0 12         BEQ  $C616
C604  EE 0A 03      INC  $030A
C607  20 59 C6      JSR  $C659
C60A  10 DC         BPL  $C5E8
C60C  20 3E C6      JSR  $C63E
C60F  AD EA 03      LDA  $03EA
C612  D0 AC         BNE  $C5C0
C614  F0 F1         BEQ  $C607
C616  AD EA 03      LDA  $03EA
C619  F0 03         BEQ  $C61E
C61B  20 59 C6      JSR  $C659
C61E  20 29 C6      JSR  $C629
C621  B0 9D         BCS  $C5C0
C623  20 3B C6      JSR  $C63B
C626  E6 09         INC  $09
C628  60            RTS
C629  18            CLC
(C) EXECUTE BOOT LOADER
```

```
C62A  AD 42 02   LDA  $0242
C62D  69 06      ADC #$06
C62F  85 04      STA  $04
C631  AD 43 02   LDA  $0243
C634  69 00      ADC #$00
C636  85 05      STA  $05
C638  6C 04 00   JMP ($0004)
C63B  6C 0C 00   JMP ($000C)
(C) INIT BOOTED SOFTWARE
C63E  A2 3D      LDX #$3D
(C) DISPLAY "BOOT ERROR"
C640  A0 C4      LDY #$C4
(C) MESSAGE
C642  8A         TXA
C643  A2 00      LDX #$00
C645  9D 44 03   STA  $0344,X
C648  98         TYA
C649  9D 45 03   STA  $0345,X
C64C  A9 09      LDA #$09
C64E  9D 42 03   STA  $0342,X
C651  A9 FF      LDA #$FF
C653  9D 48 03   STA  $0348,X
C656  4C 56 E4   JMP  $E456
C659  AD EA 03   LDA  $03EA
(C) GET NEXT SECTOR
C65C  F0 03      BEQ  $C661
C65E  4C 7A E4   JMP  $E47A
C661  A9 52      LDA #$52
C663  8D 02 03   STA  $0302
C666  A9 01      LDA #$01
C668  8D 01 03   STA  $0301
C66B  4C 53 E4   JMP  $E453
C66E  A5 08      LDA  $08
(C) ATTEMPT CASSETTE BOOT
C670  F0 09      BEQ  $C67B
C672  A5 09      LDA  $09
C674  29 02      AND #$02
C676  F0 27      BEQ  $C69F
C678  4C A0 C6   JMP  $C6A0
C67B  AD E9 03   LDA  $03E9
C67E  F0 1F      BEQ  $C69F
C680  A9 80      LDA #$80
C682  85 3E      STA  $3E
C684  EE EA 03   INC  $03EA
C687  20 7D E4   JSR  $E47D
C68A  20 BB C5   JSR  $C5BB
C68D  A9 00      LDA #$00
C68F  8D EA 03   STA  $03EA
C692  8D E9 03   STA  $03E9
C695  06 09      ASL  $09
C697  A5 0C      LDA  $0C
C699  85 02      STA  $02
C69B  A5 0D      LDA  $0D
C69D  85 03      STA  $03
C69F  60         RTS
C6A0  6C 02 00   JMP ($0002)
C6A3  A9 A0      LDA #$A0
(C) INITIALIZE DISK I/O
C6A5  8D 46 02   STA  $0246
C6A8  A9 80      LDA #$80
C6AA  8D D5 02   STA  $02D5
C6AD  A9 00      LDA #$00
C6AF  8D D6 02   STA  $02D6
C6B2  60         RTS
C6B3  A9 31      LDA #$31
(C) DISK I/O
C6B5  8D 00 03   STA  $0300
C6B8  AD 46 02   LDA  $0246
C6BB  AE 02 03   LDX  $0302
C6BE  E0 21      CPX #$21
C6C0  F0 02      BEQ  $C6C4
C6C2  A9 07      LDA #$07
C6C4  8D 06 03   STA  $0306
C6C7  A2 40      LDX #$40
C6C9  AD 02 03   LDA  $0302
C6CC  C9 50      CMP #$50
C6CE  F0 04      BEQ  $C6D4
C6D0  C9 57      CMP #$57
C6D2  D0 02      BNE  $C6D6
C6D4  A2 80      LDX #$80
C6D6  C9 53      CMP #$53
C6D8  D0 10      BNE  $C6EA
C6DA  A9 EA      LDA #$EA
C6DC  8D 04 03   STA  $0304
C6DF  A9 02      LDA #$02
C6E1  8D 05 03   STA  $0305
C6E4  A0 04      LDY #$04
C6E6  A9 00      LDA #$00
C6E8  F0 06      BEQ  $C6F0
C6EA  AC D5 02   LDY  $02D5
C6ED  AD D6 02   LDA  $02D6
C6F0  8E 03 03   STX  $0303
C6F3  8C 08 03   STY  $0308
C6F6  8D 09 03   STA  $0309
C6F9  20 59 E4   JSR  $E459
C6FC  10 01      BPL  $C6FF
C6FE  60         RTS
C6FF  AD 02 03   LDA  $0302
C702  C9 53      CMP #$53
C704  D0 0A      BNE  $C710
C706  20 3A C7   JSR  $C73A
C709  A0 02      LDY #$02
C70B  B1 15      LDA ($15),Y
C70D  8D 46 02   STA  $0246
C710  AD 02 03   LDA  $0302
C713  C9 21      CMP #$21
C715  D0 1F      BNE  $C736
C717  20 3A C7   JSR  $C73A
C71A  A0 FE      LDY #$FE
C71C  C8         INY
```

APPENDIX E6:

```
C71D  C8          INY
C71E  B1 15       LDA ($15),Y
C720  C9 FF       CMP #$FF
C722  D0 F8       BNE  $C71C
C724  C8          INY
C725  B1 15       LDA ($15),Y
C727  C8          INY
C728  C9 FF       CMP #$FF
C72A  D0 F2       BNE  $C71E
C72C  88          DEY
C72D  88          DEY
C72E  8C 08 03    STY  $0308
C731  A9 00       LDA #$00
C733  8D 09 03    STA  $0309
C736  AC 03 03    LDY  $0303
C739  60          RTS
C73A  AD 04 03    LDA  $0304
(C) SET BUFFER ADDRESS
C73D  85 15       STA  $15
C73F  AD 05 03    LDA  $0305
C742  85 16       STA  $16
C744  60          RTS
C745  A2 05       LDX #$05
(C) RELOCATE RELOCATABLE
C747  A9 00       LDA #$00
(C) ROUTINE TO NEW
C749  9D C9 02    STA  $02C9,X
(C) ADDRESS
C74C  CA          DEX
C74D  10 F8       BPL  $C747
C74F  A9 00       LDA #$00
C751  8D 33 02    STA  $0233
C754  20 CF C7    JSR  $C7CF
C757  A0 9C       LDY #$9C
C759  B0 39       BCS  $C794
C75B  8D 88 02    STA  $0288
C75E  20 CF C7    JSR  $C7CF
C761  A0 9C       LDY #$9C
C763  B0 2F       BCS  $C794
C765  8D 45 02    STA  $0245
C768  AD 88 02    LDA  $0288
C76B  C9 0B       CMP #$0B
C76D  F0 26       BEQ  $C795
C76F  2A          ROL A
C770  AA          TAX
C771  BD E4 C8    LDA  $C8E4,X
C774  8D C9 02    STA  $02C9
C777  BD E5 C8    LDA  $C8E5,X
C77A  8D CA 02    STA  $02CA
C77D  AD 45 02    LDA  $0245
C780  CD 33 02    CMP  $0233
C783  F0 CA       BEQ  $C74F
C785  20 CF C7    JSR  $C7CF
C788  A0 9C       LDY #$9C
C78A  B0 08       BCS  $C794
C78C  20 D2 C7    JSR  $C7D2
C78F  EE 33 02    INC  $0233
C792  D0 E9       BNE  $C77D
C794  60          RTS
C795  20 CF C7    JSR  $C7CF
C798  A0 9C       LDY #$9C
C79A  B0 2C       BCS  $C7C8
C79C  8D C9 02    STA  $02C9
C79F  20 CF C7    JSR  $C7CF
C7A2  A0 9C       LDY #$9C
C7A4  B0 22       BCS  $C7C8
C7A6  8D CA 02    STA  $02CA
C7A9  AD 45 02    LDA  $0245
C7AC  C9 01       CMP #$01
C7AE  F0 16       BEQ  $C7C6
C7B0  90 17       BCC  $C7C9
C7B2  18          CLC
C7B3  AD C9 02    LDA  $02C9
C7B6  6D D1 02    ADC  $02D1
C7B9  A8          TAY
C7BA  AD CA 02    LDA  $02CA
C7BD  6D D2 02    ADC  $02D2
C7C0  8C C9 02    STY  $02C9
C7C3  8D CA 02    STA  $02CA
C7C6  A0 01       LDY #$01
C7C8  60          RTS
C7C9  A0 00       LDY #$00
C7CB  A9 00       LDA #$00
C7CD  F0 F1       BEQ  $C7C0
C7CF  6C CF 02    JMP ($02CF)
C7D2  6C C9 02    JMP ($02C9)
C7D5  AC 33 02    LDY  $0233
(C) HANDLE TEXT RECORD
C7D8  C0 01       CPY #$01
C7DA  F0 0A       BEQ  $C7E6
C7DC  B0 73       BCS  $C851
C7DE  8D 4A 02    STA  $024A
C7E1  8D 8E 02    STA  $028E
C7E4  90 6A       BCC  $C850
C7E6  8D 4B 02    STA  $024B
C7E9  8D 8F 02    STA  $028F
C7EC  A2 00       LDX #$00
C7EE  AD 88 02    LDA  $0288
C7F1  F0 06       BEQ  $C7F9
C7F3  C9 0A       CMP #$0A
C7F5  F0 15       BEQ  $C80C
C7F7  A2 02       LDX #$02
C7F9  18          CLC
C7FA  AD 4A 02    LDA  $024A
C7FD  7D D1 02    ADC  $02D1,X
C800  8D 8E 02    STA  $028E
C803  AD 4B 02    LDA  $024B
C806  7D D2 02    ADC  $02D2,X
C809  8D 8F 02    STA  $028F
C80C  18          CLC
```

```
C80D   AD 8E 02      LDA  $028E
C810   6D 45 02      ADC  $0245
C813   48            PHA
C814   A9 00         LDA #$00
C816   6D 8F 02      ADC  $028F
C819   A8            TAY
C81A   68            PLA
C81B   38            SEC
C81C   E9 02         SBC #$02
C81E   B0 01         BCS  $C821
C820   88            DEY
C821   48            PHA
C822   98            TYA
C823   DD CC 02      CMP  $02CC,X
C826   68            PLA
C827   90 10         BCC  $C839
C829   D0 05         BNE  $C830
C82B   DD CB 02      CMP  $02CB,X
C82E   90 09         BCC  $C839
C830   9D CB 02      STA  $02CB,X
C833   48            PHA
C834   98            TYA
C835   9D CC 02      STA  $02CC,X
C838   68            PLA
C839   AE 88 02      LDX  $0288
C83C   E0 01         CPX #$01
C83E   F0 10         BEQ  $C850
C840   CC E6 02      CPY  $02E6
C843   90 0B         BCC  $C850
C845   D0 05         BNE  $C84C
C847   CD E5 02      CMP  $02E5
C84A   90 04         BCC  $C850
C84C   68            PLA
C84D   68            PLA
C84E   A0 9D         LDY #$9D
C850   60            RTS
C851   38            SEC
(C) RELOCATE TEXT
C852   48            PHA
(C) INTO MEMORY
C853   AD 33 02      LDA  $0233
C856   E9 02         SBC #$02
C858   18            CLC
C859   6D 8E 02      ADC  $028E
C85C   85 36         STA  $36
C85E   A9 00         LDA #$00
C860   6D 8F 02      ADC  $028F
C863   85 37         STA  $37
C865   68            PLA
C866   A0 00         LDY #$00
C868   91 36         STA ($36),Y
C86A   4C 50 C8      JMP  $C850
C86D   18            CLC
(C) HANDLE WORD REFERENCE
C86E   6D 8E 02      ADC  $028E
(C) RECORD TYPE
C871   85 36         STA  $36
C873   A9 00         LDA #$00
C875   6D 8F 02      ADC  $028F
C878   85 37         STA  $37
C87A   A0 00         LDY #$00
C87C   B1 36         LDA ($36),Y
C87E   18            CLC
C87F   6D D1 02      ADC  $02D1
C882   91 36         STA ($36),Y
C884   E6 36         INC  $36
C886   D0 02         BNE  $C88A
C888   E6 37         INC  $37
C88A   B1 36         LDA ($36),Y
C88C   6D D2 02      ADC  $02D2
C88F   91 36         STA ($36),Y
C891   60            RTS
C892   A2 00         LDX #$00
(C) HANDLE LOW-BYTE &
C894   AC 88 02      LDY  $0288
(C) ONE BYTE RECORD TYPE
C897   C0 04         CPY #$04
C899   90 02         BCC  $C89D
C89B   A2 02         LDX #$02
C89D   18            CLC
C89E   6D 8E 02      ADC  $028E
C8A1   85 36         STA  $36
C8A3   A9 00         LDA #$00
C8A5   6D 8F 02      ADC  $028F
C8A8   85 37         STA  $37
C8AA   A0 00         LDY #$00
C8AC   B1 36         LDA ($36),Y
C8AE   18            CLC
C8AF   7D D1 02      ADC  $02D1,X
C8B2   91 36         STA ($36),Y
C8B4   60            RTS
C8B5   48            PHA
C8B6   AD 33 02      LDA  $0233
C8B9   6A            ROR A
C8BA   68            PLA
C8BB   B0 15         BCS  $C8D2
C8BD   18            CLC
C8BE   6D 8E 02      ADC  $028E
C8C1   85 36         STA  $36
C8C3   A9 00         LDA #$00
C8C5   6D 8F 02      ADC  $028F
C8C8   85 37         STA  $37
C8CA   A0 00         LDY #$00
C8CC   B1 36         LDA ($36),Y
C8CE   8D 88 02      STA  $0288
C8D1   60            RTS
C8D2   18            CLC
C8D3   6D D1 02      ADC  $02D1
C8D6   A9 00         LDA #$00
```

```
C8D8   6D D2 02      ADC  $02D2
C8DB   6D 88 02      ADC  $0288
C8DE   A0 00         LDY #$00
C8E0   91 36         STA ($36),Y
C8E2   F0 ED         BEQ  $C8D1
C8E4   D5 C7         CMP  $C7,X
C8E6   D5 C7         CMP  $C7,X

C8E8   92 C8 92 C8 92 C8 92 C8
C8F0   6D C8 6D C8 B5 C8 B5 C8
C8F8   D5 C7 95 C7

C8FC   A9 FF         LDA #$FF
(C) SELECT & EXECUTE
C8FE   8D 44 02      STA  $0244
(C) SELF-TEST
C901   AD 01 D3      LDA  $D301
C904   29 7F         AND #$7F
C906   8D 01 D3      STA  $D301
C909   4C 83 E4      JMP  $E483
C90C   A9 01         LDA #$01
(C) INITIALIZE GENERIC
C90E   8D 48 02      STA  $0248
(C) PARALLEL DEVICE
C911   AD 48 02      LDA  $0248
C914   8D FF D1      STA  $D1FF
C917   AD 03 D8      LDA  $D803
C91A   C9 80         CMP #$80
C91C   D0 0A         BNE  $C928
C91E   AD 0B D8      LDA  $D80B
C921   C9 91         CMP #$91
C923   D0 03         BNE  $C928
C925   20 19 D8      JSR  $D819
C928   0E 48 02      ASL  $0248
C92B   D0 E4         BNE  $C911
C92D   A9 00         LDA #$00
C92F   8D FF D1      STA  $D1FF
C932   60            RTS
C933   A9 01         LDA #$01
(C) PIO-PARALLEL
C935   8D 42 00      STA  $0042
(C) DEVICE I/O
C938   AD 01 03      LDA  $0301
C93B   48            PHA
C93C   AD 47 02      LDA  $0247
C93F   F0 1A         BEQ  $C95B
C941   A2 08         LDX #$08
C943   20 AF C9      JSR  $C9AF
C946   F0 13         BEQ  $C95B
C948   8A            TXA
C949   48            PHA
C94A   20 05 D8      JSR  $D805
C94D   68            PLA
C94E   AA            TAX
C94F   90 F2         BCC  $C943
C951   A9 00         LDA #$00
C953   8D 48 02      STA  $0248
C956   8D FF D1      STA  $D1FF
C959   F0 03         BEQ  $C95E
C95B   20 71 E9      JSR  $E971
C95E   68            PLA
C95F   8D 01 03      STA  $0301
C962   A9 00         LDA #$00
C964   8D 42 00      STA  $0042
C967   8C 03 03      STY  $0303
C96A   AC 03 03      LDY  $0303
C96D   60            RTS
C96E   A2 08         LDX #$08
C970   6A            ROR A
C971   B0 03         BCS  $C976
C973   CA            DEX
C974   D0 FA         BNE  $C970
C976   AD 48 02      LDA  $0248
C979   48            PHA
C97A   BD 20 CA      LDA  $CA20,X
C97D   8D 48 02      STA  $0248
C980   8D FF D1      STA  $D1FF
C983   20 08 D8      JSR  $D808
C986   68            PLA
C987   8D 48 02      STA  $0248
C98A   8D FF D1      STA  $D1FF
C98D   68            PLA
C98E   AA            TAX
C98F   68            PLA
C990   40            RTI
C991   A0 01         LDY #$01
(C) PIO VECTOR TABLES
C993   4C DC C9      JMP  $C9DC
C996   A0 03         LDY #$03
C998   4C DC C9      JMP  $C9DC
C99B   A0 05         LDY #$05
C99D   4C DC C9      JMP  $C9DC
C9A0   A0 07         LDY #$07
C9A2   4C DC C9      JMP  $C9DC
C9A5   A0 09         LDY #$09
C9A7   4C DC C9      JMP  $C9DC
C9AA   A0 0B         LDY #$0B
C9AC   4C DC C9      JMP  $C9DC
C9AF   CA            DEX
(C) SELECT NEXT
C9B0   10 09         BPL  $C9BB
(C) PARALLEL DEVICE
C9B2   A9 00         LDA #$00
C9B4   8D 48 02      STA  $0248
C9B7   8D FF D1      STA  $D1FF
C9BA   60            RTS
C9BB   AD 47 02      LDA  $0247
C9BE   3D 21 CA      AND  $CA21,X
C9C1   F0 EC         BEQ  $C9AF
C9C3   8D 48 02      STA  $0248
```

APPENDIX E6:

```
C9C6   8D FF D1       STA   $D1FF
C9C9   60             RTS
C9CA   B9 0D D8       LDA   $D80D,Y
(C) INVOKE PARALLEL
C9CD   48             PHA
(C) DEVICE HANDLER
C9CE   88             DEY
C9CF   B9 0D D8       LDA   $D80D,Y
C9D2   48             PHA
C9D3   AD 4C 02       LDA   $024C
C9D6   AE 4D 02       LDX   $024D
C9D9   A0 92          LDY #$92
C9DB   60             RTS
C9DC   8D 4C 02       STA   $024C
C9DF   8E 4D 02       STX   $024D
C9E2   AD 42 00       LDA   $0042
C9E5   48             PHA
C9E6   A9 01          LDA #$01
C9E8   8D 42 00       STA   $0042
C9EB   A2 08          LDX #$08
C9ED   20 AF C9       JSR   $C9AF
C9F0   F0 11          BEQ   $CA03
C9F2   8A             TXA
C9F3   48             PHA
C9F4   98             TYA
C9F5   48             PHA
C9F6   20 CA C9       JSR   $C9CA
C9F9   90 20          BCC   $CA1B
C9FB   8D 4C 02       STA   $024C
C9FE   68             PLA
C9FF   68             PLA
CA00   4C 05 CA       JMP   $CA05
CA03   A0 82          LDY #$82
CA05   A9 00          LDA #$00
CA07   8D 48 02       STA   $0248
CA0A   8D FF D1       STA   $D1FF
CA0D   68             PLA
CA0E   8D 42 00       STA   $0042
CA11   AD 4C 02       LDA   $024C
CA14   8C 4D 02       STY   $024D
CA17   AC 4D 02       LDY   $024D
CA1A   60             RTS
CA1B   68             PLA
CA1C   A8             TAY
CA1D   68             PLA
CA1E   AA             TAX
CA1F   90 CC          BCC   $C9ED

CA21   80 40 20 10 08 04 02 01

CA29   AE 2E 00       LDX   $2E00
(C) LOAD & INITIALIZE
CA2C   BD 4D 03       LDA   $034D,X
(C) PERIPHERAL HANDLER
CA2F   20 DE E7       JSR   $E7DE
```

```
CA32   B0 20          BCS   $CA54
CA34   18             CLC
CA35   20 9E E8       JSR   $E89E
CA38   B0 1A          BCS   $CA54
CA3A   AE 2E 00       LDX   $002E
CA3D   BD 4C 03       LDA   $034C,X
CA40   20 16 E7       JSR   $E716
CA43   B0 0F          BCS   $CA54
CA45   AE 2E 00       LDX   $002E
CA48   9D 40 03       STA   $0340,X
CA4B   85 20          STA   $20
CA4D   A9 03          LDA #$03
CA4F   85 17          STA   $17
CA51   4C 5C E5       JMP   $E55C
CA54   4C 10 E5       JMP   $E510

CA57   00 13 16 D1 E4 E4 E8 29
(C) SELF-TEST OFFSETS
CA5F   EB EE 00 00 2D 25 2D 2F
(C) & TEXT
CA67   32 39 00 34 25 33 34 00
CA6F   00 00 32 2F 2D 32 21 2D
CA77   00 00 2B 25 39 22 2F 21
CA7F   32 24 00 34 25 33 34 00
CA87   00 00 B2 91 00 92 00 93
CA8F   00 94 00 A8 00 A1 00 A2
CA97   00 00 00 5B 00 11 00 12
CA9F   00 13 00 14 00 15 00 16
CAA7   00 17 00 18 00 19 00 10
CAAF   00 1C 00 1E 00 A2 80 B3
CAB7   00 00 00 FF FF 00 31 00
CABF   37 00 25 00 32 00 34 00
CAC7   39 00 35 00 29 00 2F 00
CACF   30 00 0D 00 1D 00 B2 B4
CAD7   00 00 00 80 DC 80 00 21
CADF   00 33 00 24 00 26 00 27
CAE7   00 28 00 2A 00 2B 00 2C
CAEF   00 1B 00 0B 00 0A 00 A3
CAF7   00 00 00 80 B3 A8 80 00
CAFF   3A 00 38 00 23 00 36 00
CB07   22 00 2E 00 2D 00 0C 00
CB0F   0E 00 0F 00 80 B3 A8 80
CB17   00 00 00 00 00 00 00 00
CB1F   80 B3 80 B0 80 A1 80 A3
CB27   80 A5 80 80 80 A2 80 A1
CB2F   80 B2 80 00 33 00 30 00
CB37   21 00 23 00 25 00 00 00
CB3F   22 00 21 00 32 00 00 33
CB47   28 00 22 00 33 00 5C 00
CB4F   36 2F 29 23 25 00 03 A0

CB56   A0 11          LDY #$11
(C) CHECKSUM LINKAGE
CB58   A9 00          LDA #$00
CB5A   18             CLC
```

```
CB5B   71 4A          ADC ($4A),Y
CB5D   88             DEY
CB5E   10 FB          BPL  $CB5B
CB60   69 00          ADC #$00
CB62   49 FF          EOR #$FF
CB64   60             RTS

CB65   00 00 00 00 00 00 00 00
(C) UNUSED
CB6D   00 00 00 00 00 00 00 00
CB75   00 00 00 00 00 00 00 00
CB7D   00 00 00 00 00 00 00 00
CB85   00 00 00 00 00 00 00 00
CB8D   00 00 00 00 00 00 00 00
CB95   00 00 00 00 00 00 00 00
CB9D   00 00 00 00 00 00 00 00
CBA5   00 00 00 00 00 00 00 00
CBAD   00 00 00 00 00 00 00 00
CBB5   00 00 00 00 00 00 00 00
CBBD   00 00 00 00 00 00 00 00
CBC5   00 00 00 00 00 00 00 00
CBCD   00 00 00 00 00 00 00 00
CBD5   00 00 00 00 00 00 00 00
CBDD   00 00 00 00 00 00 00 00
CBE5   00 00 00 00 00 00 00 00
CBED   00 00 00 00 00 00 00 00
CBFE   00 00 00 00 00 00 00 00
CBFD   00 00 00

CC00   00 00 00 00 00 00 00 00
(C) INTERNATIONAL
CC08   00 18 18 18 18 00 18 00
(C) CHARACTER-SET
CC10   00 66 66 66 00 00 00 00
CC18   00 66 FF 66 66 FF 66 00
CC20   18 3E 60 3C 06 7C 18 00
CC28   00 66 6C 18 30 66 46 00
CC30   1C 36 1C 38 6F 66 3B 00
CC38   00 18 18 18 00 00 00 00
CC40   00 0E 1C 18 18 1C 0E 00
CC48   00 70 38 18 18 38 70 00
CC50   00 66 3C FF 3C 66 00 00
CC58   00 18 18 7E 18 18 00 00
CC60   00 00 00 00 00 18 18 30
CC68   00 00 00 7E 00 00 00 00
CC70   00 00 00 00 00 18 18 00
CC78   00 06 0C 18 30 60 40 00
CC80   00 3C 66 6E 76 66 3C 00
CC88   00 18 38 18 18 18 7E 00
CC90   00 3C 66 0C 18 30 7E 00
CC98   00 7E 0C 18 0C 66 3C 00
CCA0   00 0C 1C 3C 6C 7E 0C 00
CCA8   00 7E 60 7C 06 66 3C 00
CCB0   00 3C 60 7C 66 66 3C 00
CCB8   00 7E 06 0C 18 30 30 00
CCC0   00 3C 66 3C 66 66 3C 00
CCC8   00 3C 66 3E 06 0C 38 00
CCD0   00 00 18 18 00 18 18 00
CCD8   00 00 18 18 00 18 18 30
CCE0   06 0C 18 30 18 0C 06 00
CCE8   00 00 7E 00 00 7E 00 00
CCF0   60 30 18 0C 18 30 60 00
CCF8   00 3C 66 0C 18 00 18 00
CD00   00 3C 66 6E 6E 60 3E 00
CD08   00 18 3C 66 66 7E 66 00
CD10   00 7C 66 7C 66 66 7C 00
CD18   00 3C 66 60 60 66 3C 00
CD20   00 78 6C 66 66 6C 78 00
CD28   00 7E 60 7C 60 60 7E 00
CD30   00 7E 60 7C 60 60 60 00
CD38   00 3E 60 60 6E 66 3E 00
CD40   00 66 66 7E 66 66 66 00
CD48   00 7E 18 18 18 18 7E 00
CD50   00 06 06 06 06 66 3C 00
CD58   00 66 6C 78 78 6C 66 00
CD60   00 60 60 60 60 60 7E 00
CD68   00 63 77 7F 6B 63 63 00
CD70   00 66 76 7E 7E 6E 66 00
CD78   00 3C 66 66 66 66 3C 00
CD80   00 7C 66 66 7C 60 60 00
CD88   00 3C 66 66 66 6C 36 00
CD90   00 7C 66 66 7C 6C 66 00
CD98   00 3C 60 3C 06 06 3C 00
CDA0   00 7E 18 18 18 18 18 00
CDA8   00 66 66 66 66 66 7E 00
CDB0   00 66 66 66 66 3C 18 00
CDB8   00 63 63 6B 7F 77 63 00
CDC0   00 66 66 3C 3C 66 66 00
CDC8   00 66 66 3C 18 18 18 00
CDD0   00 7E 0C 18 30 60 7E 00
CDD8   00 1E 18 18 18 18 1E 00
CDE0   00 40 60 30 18 0C 06 00
CDE8   00 78 18 18 18 18 78 00
CDF0   00 08 1C 36 63 00 00 00
CDF8   00 00 00 00 00 00 FF 00
CE00   0C 18 3C 06 3E 66 3E 00
CE08   30 18 00 66 66 66 3E 00
CE10   36 6C 00 76 76 7E 6E 00
CE18   0C 18 7E 60 7C 60 7E 00
CE20   00 00 3C 60 60 3C 18 30
CE28   3C 66 00 3C 66 66 3C 00
CE30   30 18 00 3C 66 66 3C 00
CE38   30 18 00 38 18 18 3C 00
CE40   1C 30 30 78 30 30 7E 00
CE48   00 66 00 38 18 18 3C 00
CE50   00 66 00 66 66 66 3E 00
CE58   36 00 3C 06 3E 66 3E 00
CE60   66 00 3C 66 66 66 3C 00
CE68   0C 18 00 66 66 66 3E 00
CE70   0C 18 00 3C 66 66 3C 00
```

```
CE78   00 66 00 3C 66 66 3C 00
CE80   66 00 66 66 66 66 7E 00
CE88   3C 66 1C 06 3E 66 3E 00
CE90   3C 66 00 66 66 66 3E 00
CE98   3C 66 00 38 18 18 3C 00
CEA0   0C 18 3C 66 7E 60 3C 00
CEA8   30 18 3C 66 7E 60 3C 00
CEB0   36 6C 00 7C 66 66 66 00
CEB8   3C C3 3C 66 7E 60 3C 00
CEC0   18 00 3C 06 3E 66 3E 00
CEC8   30 18 3C 06 3E 66 3E 00
CED0   18 00 18 3C 66 7E 66 00
CED8   78 60 78 60 7E 18 1E 00
CEE0   00 18 3C 7E 18 18 18 00
CEE8   00 18 18 18 7E 3C 18 00
CEF0   00 18 30 7E 30 18 00 00
CEF8   00 18 0C 7E 0C 18 00 00
CF00   18 00 18 18 18 18 18 00
CF08   00 00 3C 06 3E 66 3E 00
CF10   00 60 60 7C 66 66 7C 00
CF18   00 00 3C 60 60 60 3C 00
CF20   00 06 06 3E 66 66 3E 00
CF28   00 00 3C 66 7E 60 3C 00
CF30   00 0E 18 3E 18 18 18 00
CF38   00 00 3E 66 66 3E 06 7C
CF40   00 60 60 7C 66 66 66 00
CF48   00 18 00 38 18 18 3C 00
CF50   00 06 00 06 06 06 06 3C
CF58   00 60 60 6C 78 6C 66 00
CF60   00 38 18 18 18 18 3C 00
CF68   00 00 66 7F 7F 6B 63 00
CF70   00 00 7C 66 66 66 66 00
CF78   00 00 3C 66 66 66 3C 00
CF80   00 00 7C 66 66 7C 60 60
CF88   00 00 3E 66 66 3E 06 06
CF90   00 00 7C 66 60 60 60 00
CF98   00 00 3E 60 3C 06 7C 00
CFA0   00 18 7E 18 18 18 0E 00
CFA8   00 00 66 66 66 66 3E 00
CFB0   00 00 66 66 66 3C 18 00
CFB8   00 00 63 6B 7F 3E 36 00
CFC0   00 00 66 3C 18 3C 66 00
CFC8   00 00 66 66 66 3E 0C 78
CFD0   00 00 7E 0C 18 30 7E 00
CFD8   66 66 18 3C 66 7E 66 00
CFE0   18 18 18 18 18 18 18 18
CFE8   00 7E 78 7C 6E 66 06 00
CFF0   08 18 38 78 38 18 08 00
CFF8   10 18 1C 1E 1C 18 10 00

D800   20 A1 DB       JSR   $DBA1
(C) ASCII TO FP
D803   20 BB DB       JSR   $DBBB
(C) CONVERSION
D806   B0 39          BCS   $D841
```

```
D808   A2 ED          LDX #$ED
D80A   A0 04          LDY #$04
D80C   20 48 DA       JSR  $DA48
D80F   A2 FF          LDX #$FF
D811   86 F1          STX  $F1
D813   20 44 DA       JSR  $DA44
D816   F0 04          BEQ  $D81C
D818   A9 FF          LDA #$FF
D81A   85 F0          STA  $F0
D81C   20 94 DB       JSR  $DB94
D81F   B0 21          BCS  $D842
D821   48             PHA
D822   A6 D5          LDX  $D5
D824   D0 11          BNE  $D837
D826   20 EB DB       JSR  $DBEB
D829   68             PLA
D82A   05 D9          ORA  $D9
D82C   85 D9          STA  $D9
D82E   A6 F1          LDX  $F1
D830   30 E6          BMI  $D818
D832   E8             INX
D833   86 F1          STX  $F1
D835   D0 E1          BNE  $D818
D837   68             PLA
D838   A6 F1          LDX  $F1
D83A   10 02          BPL  $D83E
D83C   E6 ED          INC  $ED
D83E   4C 18 D8       JMP  $D818
D841   60             RTS
D842   C9 2E          CMP #$2E
D844   F0 14          BEQ  $D85A
D846   C9 45          CMP #$45
D848   F0 19          BEQ  $D863
D84A   A6 F0          LDX  $F0
D84C   D0 68          BNE  $D8B6
D84E   C9 2B          CMP #$2B
D850   F0 C6          BEQ  $D818
D852   C9 2D          CMP #$2D
D854   F0 00          BEQ  $D856
D856   85 EE          STA  $EE
D858   F0 BE          BEQ  $D818
D85A   A6 F1          LDX  $F1
D85C   10 58          BPL  $D8B6
D85E   E8             INX
D85F   86 F1          STX  $F1
D861   F0 B5          BEQ  $D818
D863   A5 F2          LDA  $F2
D865   85 EC          STA  $EC
D867   20 94 DB       JSR  $DB94
D86A   B0 37          BCS  $D8A3
D86C   AA             TAX
D86D   A5 ED          LDA  $ED
D86F   48             PHA
D870   86 ED          STX  $ED
D872   20 94 DB       JSR  $DB94
```

```
D875   B0 17      BCS  $D88E
D877   48         PHA
D878   A5 ED      LDA  $ED
D87A   0A         ASL A
D87B   85 ED      STA  $ED
D87D   0A         ASL A
D87E   0A         ASL A
D87F   65 ED      ADC  $ED
D881   85 ED      STA  $ED
D883   68         PLA
D884   18         CLC
D885   65 ED      ADC  $ED
D887   85 ED      STA  $ED
D889   A4 F2      LDY  $F2
D88B   20 9D DB   JSR  $DB9D
D88E   A5 EF      LDA  $EF
D890   F0 09      BEQ  $D89B
D892   A5 ED      LDA  $ED
D894   49 FF      EOR #$FF
D896   18         CLC
D897   69 01      ADC #$01
D899   85 ED      STA  $ED
D89B   68         PLA
D89C   18         CLC
D89D   65 ED      ADC  $ED
D89F   85 ED      STA  $ED
D8A1   D0 13      BNE  $D8B6
D8A3   C9 2B      CMP #$2B
D8A5   F0 06      BEQ  $D8AD
D8A7   C9 2D      CMP #$2D
D8A9   D0 07      BNE  $D8B2
D8AB   85 EF      STA  $EF
D8AD   20 94 DB   JSR  $DB94
D8B0   90 BA      BCC  $D86C
D8B2   A5 EC      LDA  $EC
D8B4   85 F2      STA  $F2
D8B6   C6 F2      DEC  $F2
D8B8   A5 ED      LDA  $ED
D8BA   A6 F1      LDX  $F1
D8BC   30 05      BMI  $D8C3
D8BE   F0 03      BEQ  $D8C3
D8C0   38         SEC
D8C1   E5 F1      SBC  $F1
D8C3   48         PHA
D8C4   2A         ROL A
D8C5   68         PLA
D8C6   6A         ROR A
D8C7   85 ED      STA  $ED
D8C9   90 03      BCC  $D8CE
D8CB   20 EB DB   JSR  $DBEB
D8CE   A5 ED      LDA  $ED
D8D0   18         CLC
D8D1   69 44      ADC #$44
D8D3   85 D4      STA  $D4
D8D5   20 00 DC   JSR  $DC00
```

```
D8D8   B0 0B      BCS  $D8E5
D8DA   A6 EE      LDX  $EE
D8DC   F0 06      BEQ  $D8E4
D8DE   A5 D4      LDA  $D4
D8E0   09 80      ORA #$80
D8E2   85 D4      STA  $D4
D8E4   18         CLC
D8E5   60         RTS
D8E6   20 51 DA   JSR  $DA51
(C) FP TO ASCII
D8E9   A9 30      LDA #$30
(C) CONVERSION
D8EB   8D 7F 05   STA  $057F
D8EE   A5 D4      LDA  $D4
D8F0   F0 28      BEQ  $D91A
D8F2   29 7F      AND #$7F
D8F4   C9 3F      CMP #$3F
D8F6   90 28      BCC  $D920
D8F8   C9 45      CMP #$45
D8FA   B0 24      BCS  $D920
D8FC   38         SEC
D8FD   E9 3F      SBC #$3F
D8FF   20 70 DC   JSR  $DC70
D902   20 A4 DC   JSR  $DCA4
D905   09 80      ORA #$80
D907   9D 80 05   STA  $0580,X
D90A   AD 80 05   LDA  $0580
D90D   C9 2E      CMP #$2E
D90F   F0 03      BEQ  $D914
D911   4C 88 D9   JMP  $D988
D914   20 C1 DC   JSR  $DCC1
D917   4C 9C D9   JMP  $D99C
D91A   A9 B0      LDA #$B0
D91C   8D 80 05   STA  $0580
D91F   60         RTS
D920   A9 01      LDA #$01
D922   20 70 DC   JSR  $DC70
D925   20 A4 DC   JSR  $DCA4
D928   E8         INX
D929   86 F2      STX  $F2
D92B   A5 D4      LDA  $D4
D92D   0A         ASL A
D92E   38         SEC
D92F   E9 80      SBC #$80
D931   AE 80 05   LDX  $0580
D934   E0 30      CPX #$30
D936   F0 17      BEQ  $D94F
D938   AE 81 05   LDX  $0581
D93B   AC 82 05   LDY  $0582
D93E   8E 82 05   STX  $0582
D941   8C 81 05   STY  $0581
D944   A6 F2      LDX  $F2
D946   E0 02      CPX #$02
D948   D0 02      BNE  $D94C
D94A   E6 F2      INC  $F2
```

```
D94C   18         CLC
D94D   69 01      ADC #$01
D94F   85 ED      STA  $ED
D951   A9 45      LDA #$45
D953   A4 F2      LDY  $F2
D955   20 9F DC   JSR  $DC9F
D958   84 F2      STY  $F2
D95A   A5 ED      LDA  $ED
D95C   10 0B      BPL  $D969
D95E   A9 00      LDA #$00
D960   38         SEC
D961   E5 ED      SBC  $ED
D963   85 ED      STA  $ED
D965   A9 2D      LDA #$2D
D967   D0 02      BNE  $D96B
D969   A9 2B      LDA #$2B
D96B   20 9F DC   JSR  $DC9F
D96E   A2 00      LDX #$00
D970   A5 ED      LDA  $ED
D972   38         SEC
D973   E9 0A      SBC #$0A
D975   90 03      BCC  $D97A
D977   E8         INX
D978   D0 F8      BNE  $D972
D97A   18         CLC
D97B   69 0A      ADC #$0A
D97D   48         PHA
D97E   8A         TXA
D97F   20 9D DC   JSR  $DC9D
D982   68         PLA
D983   09 80      ORA #$80
D985   20 9D DC   JSR  $DC9D
D988   AD 80 05   LDA  $0580
D98B   C9 30      CMP #$30
D98D   D0 0D      BNE  $D99C
D98F   18         CLC
D990   A5 F3      LDA  $F3
D992   69 01      ADC #$01
D994   85 F3      STA  $F3
D996   A5 F4      LDA  $F4
D998   69 00      ADC #$00
D99A   85 F4      STA  $F4
D99C   A5 D4      LDA  $D4
D99E   10 09      BPL  $D9A9
D9A0   20 C1 DC   JSR  $DCC1
D9A3   A0 00      LDY #$00
D9A5   A9 2D      LDA #$2D
D9A7   91 F3      STA ($F3),Y
D9A9   60         RTS
D9AA   A5 D4      LDA  $D4
(C) INTEGER TO FP
D9AC   85 F8      STA  $F8
(C) CONVERSION
D9AE   A5 D5      LDA  $D5
D9B0   85 F7      STA  $F7
D9B2   20 44 DA   JSR  $DA44
D9B5   F8         SED
D9B6   A0 10      LDY #$10
D9B8   06 F8      ASL  $F8
D9BA   26 F7      ROL  $F7
D9BC   A2 03      LDX #$03
D9BE   B5 D4      LDA  $D4,X
D9C0   75 D4      ADC  $D4,X
D9C2   95 D4      STA  $D4,X
D9C4   CA         DEX
D9C5   D0 F7      BNE  $D9BE
D9C7   88         DEY
D9C8   D0 EE      BNE  $D9B8
D9CA   D8         CLD
D9CB   A9 42      LDA #$42
D9CD   85 D4      STA  $D4
D9CF   4C 00 DC   JMP  $DC00
D9D2   A9 00      LDA #$00
(C) FP TO INTEGER
D9D4   85 F7      STA  $F7
(C) CONVERSION
D9D6   85 F8      STA  $F8
D9D8   A5 D4      LDA  $D4
D9DA   30 66      BMI  $DA42
D9DC   C9 43      CMP #$43
D9DE   B0 62      BCS  $DA42
D9E0   38         SEC
D9E1   E9 40      SBC #$40
D9E3   90 3F      BCC  $DA24
D9E5   69 00      ADC #$00
D9E7   0A         ASL A
D9E8   85 F5      STA  $F5
D9EA   20 5A DA   JSR  $DA5A
D9ED   B0 53      BCS  $DA42
D9EF   A5 F7      LDA  $F7
D9F1   85 F9      STA  $F9
D9F3   A5 F8      LDA  $F8
D9F5   85 FA      STA  $FA
D9F7   20 5A DA   JSR  $DA5A
D9FA   B0 46      BCS  $DA42
D9FC   20 5A DA   JSR  $DA5A
D9FF   B0 41      BCS  $DA42
DA01   18         CLC
DA02   A5 F8      LDA  $F8
DA04   65 FA      ADC  $FA
DA06   85 F8      STA  $F8
DA08   A5 F7      LDA  $F7
DA0A   65 F9      ADC  $F9
DA0C   85 F7      STA  $F7
DA0E   B0 32      BCS  $DA42
DA10   20 B9 DC   JSR  $DCB9
DA13   18         CLC
DA14   65 F8      ADC  $F8
DA16   85 F8      STA  $F8
DA18   A5 F7      LDA  $F7
```

APPENDIX E6:

```
DA1A   69 00      ADC #$00
DA1C   B0 24      BCS  $DA42
DA1E   85 F7      STA  $F7
DA20   C6 F5      DEC  $F5
DA22   D0 C6      BNE  $D9EA
DA24   20 B9 DC   JSR  $DCB9
DA27   C9 05      CMP #$05
DA29   90 0D      BCC  $DA38
DA2B   18         CLC
DA2C   A5 F8      LDA  $F8
DA2E   69 01      ADC #$01
DA30   85 F8      STA  $F8
DA32   A5 F7      LDA  $F7
DA34   69 00      ADC #$00
DA36   85 F7      STA  $F7
DA38   A5 F8      LDA  $F8
DA3A   85 D4      STA  $D4
DA3C   A5 F7      LDA  $F7
DA3E   85 D5      STA  $D5
DA40   18         CLC
DA41   60         RTS
DA42   38         SEC
DA43   60         RTS
DA44   A2 D4      LDX #$D4
(C) CLEAR FR0
DA46   A0 06      LDY #$06
(C) CLEAR FR1
DA48   A9 00      LDA #$00
DA4A   95 00      STA  $00,X
DA4C   E8         INX
DA4D   88         DEY
DA4E   D0 FA      BNE  $DA4A
DA50   60         RTS
DA51   A9 05      LDA #$05
DA53   85 F4      STA  $F4
DA55   A9 80      LDA #$80
DA57   85 F3      STA  $F3
DA59   60         RTS
DA5A   18         CLC
DA5B   26 F8      ROL  $F8
DA5D   26 F7      ROL  $F7
DA5F   60         RTS
DA60   A5 E0      LDA  $E0
(C) FP SUBTRACT
DA62   49 80      EOR #$80
DA64   85 E0      STA  $E0
DA66   A5 E0      LDA  $E0
(C) FP ADDITION
DA68   29 7F      AND #$7F
DA6A   85 F7      STA  $F7
DA6C   A5 D4      LDA  $D4
DA6E   29 7F      AND #$7F
DA70   38         SEC
DA71   E5 F7      SBC  $F7
DA73   10 10      BPL  $DA85
```

```
DA75   A2 05      LDX #$05
DA77   B5 D4      LDA  $D4,X
DA79   B4 E0      LDY  $E0,X
DA7B   95 E0      STA  $E0,X
DA7D   98         TYA
DA7E   95 D4      STA  $D4,X
DA80   CA         DEX
DA81   10 F4      BPL  $DA77
DA83   30 E1      BMI  $DA66
DA85   F0 07      BEQ  $DA8E
DA87   C9 05      CMP #$05
DA89   B0 19      BCS  $DAA4
DA8B   20 3E DC   JSR  $DC3E
DA8E   F8         SED
DA8F   A5 D4      LDA  $D4
DA91   45 E0      EOR  $E0
DA93   30 1E      BMI  $DAB3
DA95   A2 04      LDX #$04
DA97   18         CLC
DA98   B5 D5      LDA  $D5,X
DA9A   75 E1      ADC  $E1,X
DA9C   95 D5      STA  $D5,X
DA9E   CA         DEX
DA9F   10 F7      BPL  $DA98
DAA1   D8         CLD
DAA2   B0 03      BCS  $DAA7
DAA4   4C 00 DC   JMP  $DC00
DAA7   A9 01      LDA #$01
DAA9   20 3A DC   JSR  $DC3A
DAAC   A9 01      LDA #$01
DAAE   85 D5      STA  $D5
DAB0   4C 00 DC   JMP  $DC00
DAB3   A2 04      LDX #$04
DAB5   38         SEC
DAB6   B5 D5      LDA  $D5,X
DAB8   F5 E1      SBC  $E1,X
DABA   95 D5      STA  $D5,X
DABC   CA         DEX
DABD   10 F7      BPL  $DAB6
DABF   90 04      BCC  $DAC5
DAC1   D8         CLD
DAC2   4C 00 DC   JMP  $DC00
DAC5   A5 D4      LDA  $D4
DAC7   49 80      EOR #$80
DAC9   85 D4      STA  $D4
DACB   38         SEC
DACC   A2 04      LDX #$04
DACE   A9 00      LDA #$00
DAD0   F5 D5      SBC  $D5,X
DAD2   95 D5      STA  $D5,X
DAD4   CA         DEX
DAD5   10 F7      BPL  $DACE
DAD7   D8         CLD
DAD8   4C 00 DC   JMP  $DC00
```

```
DADB  A5 D4       LDA  $D4
(C) FP MULTIPLICATION
DADD  F0 45       BEQ  $DB24
DADF  A5 E0       LDA  $E0
DAE1  F0 3E       BEQ  $DB21
DAE3  20 CF DC    JSR  $DCCF
DAE6  38          SEC
DAE7  E9 40       SBC #$40
DAE9  38          SEC
DAEA  65 E0       ADC  $E0
DAEC  30 38       BMI  $DB26
DAEE  20 E0 DC    JSR  $DCE0
DAF1  A5 DF       LDA  $DF
DAF3  29 0F       AND #$0F
DAF5  85 F6       STA  $F6
DAF7  C6 F6       DEC  $F6
DAF9  30 06       BMI  $DB01
DAFB  20 01 DD    JSR  $DD01
DAFE  4C F7 DA    JMP  $DAF7
DB01  A5 DF       LDA  $DF
DB03  4A          LSR A
DB04  4A          LSR A
DB05  4A          LSR A
DB06  4A          LSR A
DB07  85 F6       STA  $F6
DB09  C6 F6       DEC  $F6
DB0B  30 06       BMI  $DB13
DB0D  20 05 DD    JSR  $DD05
DB10  4C 09 DB    JMP  $DB09
DB13  20 62 DC    JSR  $DC62
DB16  C6 F5       DEC  $F5
DB18  D0 D7       BNE  $DAF1
DB1A  A5 ED       LDA  $ED
DB1C  85 D4       STA  $D4
DB1E  4C 04 DC    JMP  $DC04
DB21  20 44 DA    JSR  $DA44
DB24  18          CLC
DB25  60          RTS
DB26  38          SEC
DB27  60          RTS
DB28  A5 E0       LDA  $E0
(C) FP DIVISION
DB2A  F0 FA       BEQ  $DB26
DB2C  A5 D4       LDA  $D4
DB2E  F0 F4       BEQ  $DB24
DB30  20 CF DC    JSR  $DCCF
DB33  38          SEC
DB34  E5 E0       SBC  $E0
DB36  18          CLC
DB37  69 40       ADC #$40
DB39  30 EB       BMI  $DB26
DB3B  20 E0 DC    JSR  $DCE0
DB3E  E6 F5       INC  $F5
DB40  4C 4E DB    JMP  $DB4E
DB43  A2 00       LDX #$00
```

```
DB45  B5 D5       LDA  $D5,X
DB47  95 D4       STA  $D4,X
DB49  E8          INX
DB4A  E0 0C       CPX #$0C
DB4C  D0 F7       BNE  $DB45
DB4E  A0 05       LDY #$05
DB50  38          SEC
DB51  F8          SED
DB52  B9 DA 00    LDA  $00DA,Y
DB55  F9 E6 00    SBC  $00E6,Y
DB58  99 DA 00    STA  $00DA,Y
DB5B  88          DEY
DB5C  10 F4       BPL  $DB52
DB5E  D8          CLD
DB5F  90 04       BCC  $DB65
DB61  E6 D9       INC  $D9
DB63  D0 E9       BNE  $DB4E
DB65  20 0F DD    JSR  $DD0F
DB68  06 D9       ASL  $D9
DB6A  06 D9       ASL  $D9
DB6C  06 D9       ASL  $D9
DB6E  06 D9       ASL  $D9
DB70  A0 05       LDY #$05
DB72  38          SEC
DB73  F8          SED
DB74  B9 DA 00    LDA  $00DA,Y
DB77  F9 E0 00    SBC  $00E0,Y
DB7A  99 DA 00    STA  $00DA,Y
DB7D  88          DEY
DB7E  10 F4       BPL  $DB74
DB80  D8          CLD
DB81  90 04       BCC  $DB87
DB83  E6 D9       INC  $D9
DB85  D0 E9       BNE  $DB70
DB87  20 09 DD    JSR  $DD09
DB8A  C6 F5       DEC  $F5
DB8C  D0 B5       BNE  $DB43
DB8E  20 62 DC    JSR  $DC62
DB91  4C 1A DB    JMP  $DB1A
DB94  20 AF DB    JSR  $DBAF
DB97  A4 F2       LDY  $F2
DB99  90 02       BCC  $DB9D
DB9B  B1 F3       LDA ($F3),Y
DB9D  C8          INY
DB9E  84 F2       STY  $F2
DBA0  60          RTS
DBA1  A4 F2       LDY  $F2
DBA3  A9 20       LDA #$20
DBA5  D1 F3       CMP ($F3),Y
DBA7  D0 03       BNE  $DBAC
DBA9  C8          INY
DBAA  D0 F9       BNE  $DBA5
DBAC  84 F2       STY  $F2
DBAE  60          RTS
DBAF  A4 F2       LDY  $F2
```

```
DBB1  B1 F3     LDA ($F3),Y    DC17  C0 05     CPY #$05
DBB3  38        SEC            DC19  90 F5     BCC  $DC10
DBB4  E9 30     SBC #$30       DC1B  C6 D4     DEC  $D4
DBB6  90 18     BCC  $DBD0     DC1D  CA        DEX
DBB8  C9 0A     CMP #$0A       DC1E  D0 EA     BNE  $DC0A
DBBA  60        RTS            DC20  A5 D5     LDA  $D5
DBBB  A5 F2     LDA  $F2       DC22  D0 04     BNE  $DC28
DBBD  48        PHA            DC24  85 D4     STA  $D4
DBBE  20 94 DB  JSR  $DB94     DC26  18        CLC
DBC1  90 1F     BCC  $DBE2     DC27  60        RTS
DBC3  C9 2E     CMP #$2E       DC28  A5 D4     LDA  $D4
DBC5  F0 14     BEQ  $DBDB     DC2A  29 7F     AND #$7F
DBC7  C9 2B     CMP #$2B       DC2C  C9 71     CMP #$71
DBC9  F0 07     BEQ  $DBD2     DC2E  90 01     BCC  $DC31
DBCB  C9 2D     CMP #$2D       DC30  60        RTS
DBCD  F0 03     BEQ  $DBD2     DC31  C9 0F     CMP #$0F
DBCF  68        PLA            DC33  B0 03     BCS  $DC38
DBD0  38        SEC            DC35  20 44 DA  JSR  $DA44
DBD1  60        RTS            DC38  18        CLC
DBD2  20 94 DB  JSR  $DB94     DC39  60        RTS
DBD5  90 0B     BCC  $DBE2     DC3A  A2 D4     LDX #$D4
DBD7  C9 2E     CMP #$2E       DC3C  D0 02     BNE  $DC40
DBD9  D0 F4     BNE  $DBCF     DC3E  A2 E0     LDX #$E0
DBDB  20 94 DB  JSR  $DB94     DC40  86 F9     STX  $F9
DBDE  90 02     BCC  $DBE2     DC42  85 F7     STA  $F7
DBE0  B0 ED     BCS  $DBCF     DC44  85 F8     STA  $F8
DBE2  68        PLA            DC46  A0 04     LDY #$04
DBE3  85 F2     STA  $F2       DC48  B5 04     LDA  $04,X
DBE5  18        CLC            DC4A  95 05     STA  $05,X
DBE6  60        RTS            DC4C  CA        DEX
DBE7  A2 E7     LDX #$E7       DC4D  88        DEY
DBE9  D0 02     BNE  $DBED     DC4E  D0 F8     BNE  $DC48
DBEB  A2 D5     LDX #$D5       DC50  A9 00     LDA #$00
DBED  A0 04     LDY #$04       DC52  95 05     STA  $05,X
DBEF  18        CLC            DC54  A6 F9     LDX  $F9
DBF0  36 04     ROL  $04,X     DC56  C6 F7     DEC  $F7
DBF2  36 03     ROL  $03,X     DC58  D0 EC     BNE  $DC46
DBF4  36 02     ROL  $02,X     DC5A  B5 00     LDA  $00,X
DBF6  36 01     ROL  $01,X     DC5C  18        CLC
DBF8  36 00     ROL  $00,X     DC5D  65 F8     ADC  $F8
DBFA  26 EC     ROL  $EC       DC5F  95 00     STA  $00,X
DBFC  88        DEY            DC61  60        RTS
DBFD  D0 F0     BNE  $DBEF     DC62  A2 0A     LDX #$0A
DBFF  60        RTS            DC64  B5 D4     LDA  $D4,X
DC00  A2 00     LDX #$00       DC66  95 D5     STA  $D5,X
DC02  86 DA     STX  $DA       DC68  CA        DEX
DC04  A2 04     LDX #$04       DC69  10 F9     BPL  $DC64
DC06  A5 D4     LDA  $D4       DC6B  A9 00     LDA #$00
DC08  F0 2E     BEQ  $DC38     DC6D  85 D4     STA  $D4
DC0A  A5 D5     LDA  $D5       DC6F  60        RTS
DC0C  D0 1A     BNE  $DC28     DC70  85 F7     STA  $F7
DC0E  A0 00     LDY #$00       DC72  A2 00     LDX #$00
DC10  B9 D6 00  LDA  $00D6,Y   DC74  A0 00     LDY #$00
DC13  99 D5 00  STA  $00D5,Y   DC76  20 93 DC  JSR  $DC93
DC16  C8        INY            DC79  38        SEC
```

```
DC7A   E9 01        SBC #$01
DC7C   85 F7        STA  $F7
DC7E   B5 D5        LDA  $D5,X
DC80   4A           LSR A
DC81   4A           LSR A
DC82   4A           LSR A
DC83   4A           LSR A
DC84   20 9D DC     JSR  $DC9D
DC87   B5 D5        LDA  $D5,X
DC89   29 0F        AND #$0F
DC8B   20 9D DC     JSR  $DC9D
DC8E   E8           INX
DC8F   E0 05        CPX #$05
DC91   90 E3        BCC  $DC76
DC93   A5 F7        LDA  $F7
DC95   D0 05        BNE  $DC9C
DC97   A9 2E        LDA #$2E
DC99   20 9F DC     JSR  $DC9F
DC9C   60           RTS
DC9D   09 30        ORA #$30
DC9F   99 80 05     STA  $0580,Y
DCA2   C8           INY
DCA3   60           RTS
DCA4   A2 0A        LDX #$0A
DCA6   BD 80 05     LDA  $0580,X
DCA9   C9 2E        CMP #$2E
DCAB   F0 07        BEQ  $DCB4
DCAD   C9 30        CMP #$30
DCAF   D0 07        BNE  $DCB8
DCB1   CA           DEX
DCB2   D0 F2        BNE  $DCA6
DCB4   CA           DEX
DCB5   BD 80 05     LDA  $0580,X
DCB8   60           RTS
DCB9   20 EB DB     JSR  $DBEB
DCBC   A5 EC        LDA  $EC
DCBE   29 0F        AND #$0F
DCC0   60           RTS
DCC1   38           SEC
DCC2   A5 F3        LDA  $F3
DCC4   E9 01        SBC #$01
DCC6   85 F3        STA  $F3
DCC8   A5 F4        LDA  $F4
DCCA   E9 00        SBC #$00
DCCC   85 F4        STA  $F4
DCCE   60           RTS
DCCF   A5 D4        LDA  $D4
DCD1   45 E0        EOR  $E0
DCD3   29 80        AND #$80
DCD5   85 EE        STA  $EE
DCD7   06 E0        ASL  $E0
DCD9   46 E0        LSR  $E0
DCDB   A5 D4        LDA  $D4
DCDD   29 7F        AND #$7F
DCDF   60           RTS
DCE0   05 EE        ORA  $EE
DCE2   85 ED        STA  $ED
DCE4   A9 00        LDA #$00
DCE6   85 D4        STA  $D4
DCE8   85 E0        STA  $E0
DCEA   20 28 DD     JSR  $DD28
DCED   20 E7 DB     JSR  $DBE7
DCF0   A5 EC        LDA  $EC
DCF2   29 0F        AND #$0F
DCF4   85 E6        STA  $E6
DCF6   A9 05        LDA #$05
DCF8   85 F5        STA  $F5
DCFA   20 34 DD     JSR  $DD34
DCFD   20 44 DA     JSR  $DA44
DD00   60           RTS
DD01   A2 D9        LDX #$D9
DD03   D0 06        BNE  $DD0B
DD05   A2 D9        LDX #$D9
DD07   D0 08        BNE  $DD11
DD09   A2 DF        LDX #$DF
DD0B   A0 E5        LDY #$E5
DD0D   D0 04        BNE  $DD13
DD0F   A2 DF        LDX #$DF
DD11   A0 EB        LDY #$EB
DD13   A9 05        LDA #$05
DD15   85 F7        STA  $F7
DD17   18           CLC
DD18   F8           SED
DD19   B5 00        LDA  $00,X
DD1B   79 00 00     ADC  $0000,Y
DD1E   95 00        STA  $00,X
DD20   CA           DEX
DD21   88           DEY
DD22   C6 F7        DEC  $F7
DD24   10 F3        BPL  $DD19
DD26   D8           CLD
DD27   60           RTS
DD28   A0 05        LDY #$05
DD2A   B9 E0 00     LDA  $00E0,Y
DD2D   99 E6 00     STA  $00E6,Y
DD30   88           DEY
DD31   10 F7        BPL  $DD2A
DD33   60           RTS
DD34   A0 05        LDY #$05
DD36   B9 D4 00     LDA  $00D4,Y
DD39   99 DA 00     STA  $00DA,Y
DD3C   88           DEY
DD3D   10 F7        BPL  $DD36
DD3F   60           RTS
DD40   86 FE        STX  $FE
(C) FP POLYNOMIAL EVALUATION
DD42   84 FF        STY  $FF
DD44   85 EF        STA  $EF
DD46   A2 E0        LDX #$E0
DD48   A0 05        LDY #$05
```

```
DD4A   20 A7 DD   JSR   $DDA7
DD4D   20 B6 DD   JSR   $DDB6
DD50   A6 FE      LDX   $FE
DD52   A4 FF      LDY   $FF
DD54   20 89 DD   JSR   $DD89
DD57   C6 EF      DEC   $EF
DD59   F0 2D      BEQ   $DD88
DD5B   20 DB DA   JSR   $DADB
DD5E   B0 28      BCS   $DD88
DD60   18         CLC
DD61   A5 FE      LDA   $FE
DD63   69 06      ADC #$06
DD65   85 FE      STA   $FE
DD67   90 06      BCC   $DD6F
DD69   A5 FF      LDA   $FF
DD6B   69 00      ADC #$00
DD6D   85 FF      STA   $FF
DD6F   A6 FE      LDX   $FE
DD71   A4 FF      LDY   $FF
DD73   20 98 DD   JSR   $DD98
DD76   20 66 DA   JSR   $DA66
DD79   B0 0D      BCS   $DD88
DD7B   C6 EF      DEC   $EF
DD7D   F0 09      BEQ   $DD88
DD7F   A2 E0      LDX #$E0
DD81   A0 05      LDY #$05
DD83   20 98 DD   JSR   $DD98
DD86   30 D3      BMI   $DD5B
DD88   60         RTS
DD89   86 FC      STX   $FC
(C) LOAD FR0 WITH FP
DD8B   84 FD      STY   $FD
(C) FROM 6502 X & Y
DD8D   A0 05      LDY #$05
(C) LOAD FR0 WITH FP
DD8F   B1 FC      LDA ($FC),Y
(C) FROM USER ROUTINE
DD91   99 D4 00   STA   $00D4,Y
DD94   88         DEY
DD95   10 F8      BPL   $DD8F
DD97   60         RTS
DD98   86 FC      STX   $FC
(C) LOAD FR1 WITH FP
DD9A   84 FD      STY   $FD
(C) FROM 6502 X & Y
DD9C   A0 05      LDY #$05
(C) LOAD FR1 WITH FP
DD9E   B1 FC      LDA ($FC),Y
(C) FROM USER ROUTINE
DDA0   99 E0 00   STA   $00E0,Y
DDA3   88         DEY
DDA4   10 F8      BPL   $DD9E
DDA6   60         RTS
DDA7   86 FC      STX   $FC
(C) STORE FR0 INTO
DDA9   84 FD      STY   $FD
(C) 6502 X & Y
DDAB   A0 05      LDY #$05
(C) STORE FR0 USING FLPTR
DDAD   B9 D4 00   LDA   $00D4,Y
DDB0   91 FC      STA ($FC),Y
DDB2   88         DEY
DDB3   10 F8      BPL   $DDAD
DDB5   60         RTS
DDB6   A2 05      LDX #$05
(C) MOVE FROM FR0 TO FR1
DDB8   B5 D4      LDA   $D4,X
DDBA   95 E0      STA   $E0,X
DDBC   CA         DEX
DDBD   10 F9      BPL   $DDB8
DDBF   60         RTS
DDC0   A2 89      LDX #$89
(C) BASE e EXPONENTIATION
DDC2   A0 DE      LDY #$DE
DDC4   20 98 DD   JSR   $DD98
DDC7   20 DB DA   JSR   $DADB
DDCA   B0 7F      BCS   $DE4B
DDCC   A9 00      LDA #$00
(C) BASE 10 EXPONENTIATION
DDCE   85 F1      STA   $F1
DDD0   A5 D4      LDA   $D4
DDD2   85 F0      STA   $F0
DDD4   29 7F      AND #$7F
DDD6   85 D4      STA   $D4
DDD8   38         SEC
DDD9   E9 40      SBC #$40
DDDB   30 26      BMI   $DE03
DDDD   C9 04      CMP #$04
DDDF   10 6A      BPL   $DE4B
DDE1   A2 E6      LDX #$E6
DDE3   A0 05      LDY #$05
DDE5   20 A7 DD   JSR   $DDA7
DDE8   20 D2 D9   JSR   $D9D2
DDEB   A5 D4      LDA   $D4
DDED   85 F1      STA   $F1
DDEF   A5 D5      LDA   $D5
DDF1   D0 58      BNE   $DE4B
DDF3   20 AA D9   JSR   $D9AA
DDF6   20 B6 DD   JSR   $DDB6
DDF9   A2 E6      LDX #$E6
DDFB   A0 05      LDY #$05
DDFD   20 89 DD   JSR   $DD89
DE00   20 60 DA   JSR   $DA60
DE03   A9 0A      LDA #$0A
DE05   A2 4D      LDX #$4D
DE07   A0 DE      LDY #$DE
DE09   20 40 DD   JSR   $DD40
DE0C   20 B6 DD   JSR   $DDB6
DE0F   20 DB DA   JSR   $DADB
DE12   A5 F1      LDA   $F1
```

APPENDIX E6:

```
DE14  F0 23          BEQ  $DE39
DE16  18             CLC
DE17  6A             ROR A
DE18  85 E0          STA  $E0
DE1A  A9 01          LDA #$01
DE1C  90 02          BCC  $DE20
DE1E  A9 10          LDA #$10
DE20  85 E1          STA  $E1
DE22  A2 04          LDX #$04
DE24  A9 00          LDA #$00
DE26  95 E2          STA  $E2,X
DE28  CA             DEX
DE29  10 FB          BPL  $DE26
DE2B  A5 E0          LDA  $E0
DE2D  18             CLC
DE2E  69 40          ADC #$40
DE30  B0 19          BCS  $DE4B
DE32  30 17          BMI  $DE4B
DE34  85 E0          STA  $E0
DE36  20 DB DA       JSR  $DADB
DE39  A5 F0          LDA  $F0
DE3B  10 0D          BPL  $DE4A
DE3D  20 B6 DD       JSR  $DDB6
DE40  A2 8F          LDX #$8F
DE42  A0 DE          LDY #$DE
DE44  20 89 DD       JSR  $DD89
DE47  20 28 DB       JSR  $DB28
DE4A  60             RTS
DE4B  38             SEC
DE4C  60             RTS

DE4D  3D 17 94 19 00 00 3D 57
DE55  33 05 00 00 3E 05 54 76
DE5D  62 00 3E 32 19 62 27 00
DE65  3F 01 68 60 30 36 3F 07
DE6D  32 03 27 41 3F 25 43 34
DE75  56 75 3F 66 27 37 30 50
DE7D  40 01 15 12 92 55 3F 99
DE85  99 99 99 99 3F 43 42 94
DE8D  48 19 40 01 00 00 00 00

DE95  86 FE          STX  $FE
DE97  84 FF          STY  $FF
DE99  A2 E0          LDX #$E0
DE9B  A0 05          LDY #$05
DE9D  20 A7 DD       JSR  $DDA7
DEA0  A6 FE          LDX  $FE
DEA2  A4 FF          LDY  $FF
DEA4  20 98 DD       JSR  $DD98
DEA7  20 66 DA       JSR  $DA66
DEAA  A2 E6          LDX #$E6
DEAC  A0 05          LDY #$00
DEAE  20 A7 DD       JSR  $DDA7
DEB1  A2 E0          LDX #$E0
DEB3  A0 05          LDY #$05
```

```
DEB5  20 89 DD       JSR  $DD89
DEB8  A6 FE          LDX  $FE
DEBA  A4 FF          LDY  $FF
DEBC  20 98 DD       JSR  $DD98
DEBF  20 60 DA       JSR  $DA60
DEC2  A2 E6          LDX #$E6
DEC4  A0 05          LDY #$05
DEC6  20 98 DD       JSR  $DD98
DEC9  20 28 DB       JSR  $DB28
DECC  60             RTS
DECD  A9 01          LDA #$01
(C) NATURAL LOGARITHM
DECF  D0 02          BNE  $DED3
DED1  A9 00          LDA #$00
(C) BASE 10 LOGARITHM
DED3  85 F0          STA  $F0
DED5  A5 D4          LDA  $D4
DED7  F0 05          BEQ  $DEDE
DED9  30 03          BMI  $DEDE
DEDB  4C F6 DF       JMP  $DFF6
DEDE  38             SEC
DEDF  60             RTS
DEE0  E9 40          SBC #$40
DEE2  0A             ASL A
DEE3  85 F1          STA  $F1
DEE5  A5 D5          LDA  $D5
DEE7  29 F0          AND #$F0
DEE9  D0 04          BNE  $DEEF
DEEB  A9 01          LDA #$01
DEED  D0 04          BNE  $DEF3
DEEF  E6 F1          INC  $F1
DEF1  A9 10          LDA #$10
DEF3  85 E1          STA  $E1
DEF5  A2 04          LDX #$04
DEF7  A9 00          LDA #$00
DEF9  95 E2          STA  $E2,X
DEFB  CA             DEX
DEFC  10 FB          BPL  $DEF9
DEFE  20 28 DB       JSR  $DB28
DF01  A2 66          LDX #$66
DF03  A0 DF          LDY #$DF
DF05  20 95 DE       JSR  $DE95
DF08  A2 E6          LDX #$E6
DF0A  A0 05          LDY #$05
DF0C  20 A7 DD       JSR  $DDA7
DF0F  20 B6 DD       JSR  $DDB6
DF12  20 DB DA       JSR  $DADB
DF15  A9 0A          LDA #$0A
DF17  A2 72          LDX #$72
DF19  A0 DF          LDY #$DF
DF1B  20 40 DD       JSR  $DD40
DF1E  A2 E6          LDX #$E6
DF20  A0 05          LDY #$05
DF22  20 98 DD       JSR  $DD98
DF25  20 DB DA       JSR  $DADB
```

```
DF28   A2 6C          LDX #$6C
DF2A   A0 DF          LDY #$DF
DF2C   20 98 DD       JSR  $DD98
DF2F   20 66 DA       JSR  $DA66
DF32   20 B6 DD       JSR  $DDB6
DF35   A9 00          LDA #$00
DF37   85 D5          STA  $D5
DF39   A5 F1          LDA  $F1
DF3B   85 D4          STA  $D4
DF3D   10 07          BPL  $DF46
DF3F   49 FF          EOR #$FF
DF41   18             CLC
DF42   69 01          ADC #$01
DF44   85 D4          STA  $D4
DF46   20 AA D9       JSR  $D9AA
DF49   24 F1          BIT  $F1
DF4B   10 06          BPL  $DF53
DF4D   A9 80          LDA #$80
DF4F   05 D4          ORA  $D4
DF51   85 D4          STA  $D4
DF53   20 66 DA       JSR  $DA66
DF56   A5 F0          LDA  $F0
DF58   F0 0A          BEQ  $DF64
DF5A   A2 89          LDX #$89
DF5C   A0 DE          LDY #$DE
DF5E   20 98 DD       JSR  $DD98
DF61   20 28 DB       JSR  $DB28
DF64   18             CLC
DF65   60             RTS

DF66   40 03 16 22 77 66 3F 50
DF6E   00 00 00 00 3F 49 15 57
DF76   11 08 BF 51 70 49 47 08
DF7E   3F 39 20 57 61 95 BF 04
DF86   39 63 03 55 3F 10 09 30
DF8E   12 64 3F 09 39 08 04 60
DF96   3F 12 42 58 47 42 3F 17
DF9E   37 12 06 08 3F 28 95 29
DFA6   71 17 3F 86 85 88 96 44
DFAE   3E 16 05 44 49 00 BE 95
DFB6   68 38 45 00 3F 02 68 79
DFBE   94 16 BF 04 92 78 90 80
DFC6   3F 07 03 15 20 00 BF 08
DFCE   92 29 12 44 3F 11 08 40
DFD6   09 11 BF 14 28 31 56 04
DFDE   3F 19 99 98 77 44 BF 33
DFE6   33 33 31 13 3F 99 99 99
DFEE   99 99 3F 78 53 98 16 34

DFF6   A5 D4          LDA  $D4
DFF8   85 E0          STA  $E0
DFFA   38             SEC
DFFB   4C E0 DE       JMP  $DEE0

DFFE   00 00
```

```
E000   00 00 00 00 00 00 00 00
(C) STANDARD
E008   00 18 18 18 18 00 18 00
(C) CHARACTER-SET
E010   00 66 66 66 00 00 00 00
E018   00 66 FF 66 66 FF 66 00
E020   18 3E 60 3C 06 7C 18 00
E028   00 66 6C 18 30 66 46 00
E030   1C 36 1C 38 6F 66 3B 00
E038   00 18 18 18 00 00 00 00
E040   00 0E 1C 18 18 1C 0E 00
E048   00 70 38 18 18 38 70 00
E050   00 66 3C FF 3C 66 00 00
E058   00 18 18 7E 18 18 00 00
E060   00 00 00 00 00 18 18 30
E068   00 00 00 7E 00 00 00 00
E070   00 00 00 00 00 18 18 00
E078   00 06 0C 18 30 60 40 00
E080   00 3C 66 6E 76 66 3C 00
E088   00 18 38 18 18 18 7E 00
E090   00 3C 66 0C 18 30 7E 00
E098   00 7E 0C 18 0C 66 3C 00
E0A0   00 0C 1C 3C 6C 7E 0C 00
E0A8   00 7E 60 7C 06 66 3C 00
E0B0   00 3C 60 7C 66 66 3C 00
E0B8   00 7E 06 0C 18 30 30 00
E0C0   00 3C 66 3C 66 66 3C 00
E0C8   00 3C 66 3E 06 0C 38 00
E0D0   00 00 18 18 00 18 18 00
E0D8   00 00 18 18 00 18 18 30
E0E0   06 0C 18 30 18 0C 06 00
E0E8   00 00 7E 00 00 7E 00 00
E0F0   60 30 18 0C 18 30 60 00
E0F8   00 3C 66 0C 18 00 18 00
E100   00 3C 66 6E 6E 60 3E 00
E108   00 18 3C 66 66 7E 66 00
E110   00 7C 66 7C 66 66 7C 00
E118   00 3C 66 60 60 66 3C 00
E120   00 78 6C 66 66 6C 78 00
E128   00 7E 60 7C 60 60 7E 00
E130   00 7E 60 7C 60 60 60 00
E138   00 3E 60 60 6E 66 3E 00
E140   00 66 66 7E 66 66 66 00
E148   00 7E 18 18 18 18 7E 00
E150   00 06 06 06 06 66 3C 00
E158   00 66 6C 78 78 6C 66 00
E160   00 60 60 60 60 60 7E 00
E168   00 63 77 7F 6B 63 63 00
E170   00 66 76 7E 7E 6E 66 00
E178   00 3C 66 66 66 66 3C 00
E180   00 7C 66 66 7C 60 60 00
E188   00 3C 66 66 66 6C 36 00
E190   00 7C 66 66 7C 6C 66 00
E198   00 3C 60 3C 06 06 3C 00
E1A0   00 7E 18 18 18 18 18 00
```

APPENDIX E6:

```
E1A8  00 66 66 66 66 66 7E 00
E1B0  00 66 66 66 66 3C 18 00
E1B8  00 63 63 6B 7F 77 63 00
E1C0  00 66 66 3C 3C 66 66 00
E1C8  00 66 66 3C 18 18 18 00
E1D0  00 7E 0C 18 30 60 7E 00
E1D8  00 1E 18 18 18 18 1E 00
E1E0  00 40 60 30 18 0C 06 00
E1E8  00 78 18 18 18 18 78 00
E1F0  00 08 1C 36 63 00 00 00
E1F8  00 00 00 00 00 00 FF 00
E200  00 36 7F 7F 3E 1C 08 00
E208  18 18 18 1F 1F 18 18 18
E210  03 03 03 03 03 03 03 03
E218  18 18 18 F8 F8 00 00 00
E220  18 18 18 F8 F8 18 18 18
E228  00 00 00 F8 F8 18 18 18
E230  03 07 0E 1C 38 70 E0 C0
E238  C0 E0 70 38 1C 0E 07 03
E240  01 03 07 0F 1F 3F 7F FF
E248  00 00 00 00 0F 0F 0F 0F
E250  80 C0 E0 F0 F8 FC FE FF
E258  0F 0F 0F 0F 00 00 00 00
E260  F0 F0 F0 F0 00 00 00 00
E268  FF FF 00 00 00 00 00 00
E270  00 00 00 00 00 00 FF FF
E278  00 00 00 00 F0 F0 F0 F0
E280  00 1C 1C 77 77 08 1C 00
E288  00 00 00 1F 1F 18 18 18
E290  00 00 00 FF FF 00 00 00
E298  18 18 18 FF FF 18 18 18
E2A0  00 00 3C 7E 7E 7E 3C 00
E2A8  00 00 00 00 FF FF FF FF
E2B0  C0 C0 C0 C0 C0 C0 C0 C0
E2B8  00 00 00 FF FF 18 18 18
E2C0  18 18 18 FF FF 00 00 00
E2C8  F0 F0 F0 F0 F0 F0 F0 F0
E2D0  18 18 18 1F 1F 00 00 00
E2D8  78 60 78 60 7E 18 1E 00
E2E0  00 18 3C 7E 18 18 18 00
E2E8  00 18 18 18 7E 3C 18 00
E2F0  00 18 30 7E 30 18 00 00
E2F8  00 18 0C 7E 0C 18 00 00
E300  00 18 3C 7E 7E 3C 18 00
E308  00 00 3C 06 3E 66 3E 00
E310  00 60 60 7C 66 66 7C 00
E318  00 00 3C 60 60 60 3C 00
E320  00 06 06 3E 66 66 3E 00
E328  00 00 3C 66 7E 60 3C 00
E330  00 0E 18 3E 18 18 18 00
E338  00 00 3E 66 66 3E 06 7C
E340  00 60 60 7C 66 66 66 00
E348  00 18 00 38 18 18 3C 00
E350  00 06 00 06 06 06 06 3C
E358  00 60 60 6C 78 6C 66 00
```

```
E360  00 38 18 18 18 18 3C 00
E368  00 00 66 7F 7F 6B 63 00
E370  00 00 7C 66 66 66 66 00
E378  00 00 3C 66 66 66 3C 00
E380  00 00 7C 66 66 7C 60 60
E388  00 00 3E 66 66 3E 06 06
E390  00 00 7C 66 60 60 60 00
E398  00 00 3E 60 3C 06 7C 00
E3A0  00 18 7E 18 18 18 0E 00
E3A8  00 00 66 66 66 66 3E 00
E3B0  00 00 66 66 66 3C 18 00
E3B8  00 00 63 6B 7F 3E 36 00
E3C0  00 00 66 3C 18 3C 66 00
E3C8  00 00 66 66 66 3E 0C 78
E3D0  00 00 7E 0C 18 30 7E 00
E3D8  00 18 3C 7E 7E 18 3C 00
E3E0  18 18 18 18 18 18 18 18
E3E8  00 7E 78 7C 6E 66 06 00
E3F0  08 18 38 78 38 18 08 00
E3F8  10 18 1C 1E 1C 18 10 00

E400  93 EF 2D F2 49 F2 AF F2
(C) E: HANDLER VECTORS
E408  1D F2 2C F2 4C 6E EF 00
E410  8D EF 2D F2 7F F1 A3 F1
(C) S:      "         "
E418  1D F2 AE F9 4C 6E EF 00
E420  1D F2 1D F2 FC F2 2C F2
(C) K:      "         "
E428  1D F2 2C F2 4C 6E EF 00
E430  C1 FE 06 FF C0 FE CA FE
(C) P:      "         "
E438  A2 FE C0 FE 4C 99 FE 00
E440  E5 FC CE FD 79 FD B3 FD
(C) C:      "         "
E448  CB FD E4 FC 4C DB FC 00

E450  4C A3 C6      JMP  $C6A3
(C) DISK INIT VECTOR
E453  4C B3 C6      JMP  $C6B3
(C) DISK I/O      "
E456  4C DF E4      JMP  $E4DF
(C) CIO  ENTRY    "
E459  4C 33 C9      JMP  $C933
(C) SIO     "     "
E45C  4C 72 C2      JMP  $C272
(C) SET VBLANK  PARAMETERS
E45F  4C E2 C0      JMP  $C0E2
(C) STAGE-1 VBLANK ENTRY
E462  4C 8A C2      JMP  $C28A
(C) EXIT FROM VBLANK
E465  4C 5C E9      JMP  $E95C
(C) SIO INIT VECTOR
E468  4C 17 EC      JMP  $EC17
(C) SEND ENABLE ENTRY
```

APPENDIX E6:

```
E46B  4C 0C C0      JMP  $C00C
(C) INTERRUPT HANDLER ENTRY
E46E  4C C1 E4      JMP  $E4C1
(C) CIO INIT VECTOR
E471  4C 23 F2      JMP  $F223
(C) SELF-TEST  "
E474  4C 90 C2      JMP  $C290
(C) WARMSTART  "
E477  4C C8 C2      JMP  $C2C8
(C) COLDSTART VECTOR
E47A  4C 8D FD      JMP  $FD8D
(C) CASSETTE READ-BLOCK
E47D  4C F7 FC      JMP  $FCF7
(C) CASSETTE OPEN FOR INPUT
E480  4C 23 F2      JMP  $F223
(C) PUPDIV ENTRY
E483  4C 00 50      JMP  $5000
(C) SELF-TEST VECTOR
E486  4C BC EE      JMP  $EEBC
(C) PENT VECTOR
E489  4C 15 E9      JMP  $E915
(C) PHUNL  "
E48C  4C 98 E8      JMP  $E898
(C) PHINI VECTOR

E48F  90 C9
(C) GPDV OPEN VECTOR
E491  95 C9
(C)  "   CLOSE  "
E493  9A C9
(C)  "   GET    "
E495  9F C9
(C)  "   PUT    "
E497  A4 C9
(C)  "   STATUS "
E499  A9 C9
(C)  "   XIO    "
E49B  4C 0C C9      JMP  $C90C
(C)  "   INIT   "

E49E  00 00 00 00 00 00 00 00
(C) UNUSED
E4A6  00 00 00 00 00 00 00 00
E4AE  00 00 00 00 00 00 00 00
E4B6  00 00 00 00 00 00 00 00
E4BE  00 00

E4C0  60            RTS
E4C1  A2 00         LDX #$00
(C) INITIALIZE CIO
E4C3  A9 FF         LDA #$FF
E4C5  9D 40 03      STA  $0340,X
E4C8  A9 DB         LDA #$DB
E4CA  9D 46 03      STA  $0346,X
E4CD  A9 E4         LDA #$E4
E4CF  9D 47 03      STA  $0347,X
E4D2  8A            TXA
E4D3  18            CLC
E4D4  69 10         ADC #$10
E4D6  AA            TAX
E4D7  C9 80         CMP #$80
E4D9  90 E8         BCC  $E4C3
E4DB  60            RTS
E4DC  A0 85         LDY #$85
(C) IOCB NOT OPEN ERROR
E4DE  60            RTS
E4DF  85 2F         STA  $2F
(C) CIO
E4E1  86 2E         STX  $2E
E4E3  8A            TXA
E4E4  29 0F         AND #$0F
E4E6  D0 04         BNE  $E4EC
E4E8  E0 80         CPX #$80
E4EA  90 05         BCC  $E4F1
E4EC  A0 86         LDY #$86
E4EE  4C 70 E6      JMP  $E670
E4F1  A0 00         LDY #$00
E4F3  BD 40 03      LDA  $0340,X
E4F6  99 20 00      STA  $0020,Y
E4F9  E8            INX
E4FA  C8            INY
E4FB  C0 0C         CPY #$0C
E4FD  90 F4         BCC  $E4F3
E4FF  A5 20         LDA  $20
E501  C9 7F         CMP #$7F
E503  D0 15         BNE  $E51A
E505  A5 22         LDA  $22
E507  C9 0C         CMP #$0C
E509  F0 71         BEQ  $E57C
E50B  AD E9 02      LDA  $02E9
E50E  D0 05         BNE  $E515
E510  A0 82         LDY #$82
(C) NONEXISTENT DEVICE
E512  4C 70 E6      JMP  $E670
E515  20 29 CA      JSR  $CA29
(C) LOAD PERIPHERAL HANDLER
E518  30 F8         BMI  $E512
(C) FOR OPEN
E51A  A0 84         LDY #$84
(C) PERFORM CIO COMMAND
E51C  A5 22         LDA  $22
E51E  C9 03         CMP #$03
E520  90 25         BCC  $E547
E522  A8            TAY
E523  C0 0E         CPY #$0E
E525  90 02         BCC  $E529
E527  A0 0E         LDY #$0E
E529  84 17         STY  $17
E52B  B9 2A E7      LDA  $E72A,Y
E52E  F0 0F         BEQ  $E53F
```

```
E530  C9 02      CMP #$02
E532  F0 48      BEQ  $E57C
E534  C9 08      CMP #$08
E536  B0 5F      BCS  $E597
E538  C9 04      CMP #$04
E53A  F0 76      BEQ  $E5B2
E53C  4C 1E E6   JMP  $E61E
E53F  A5 20      LDA  $20
(C) EXECUTE OPEN COMMAND
E541  C9 FF      CMP #$FF
E543  F0 05      BEQ  $E54A
E545  A0 81      LDY #$81
E547  4C 70 E6   JMP  $E670
E54A  AD E9 02   LDA  $02E9
E54D  D0 27      BNE  $E576
E54F  20 FF E6   JSR  $E6FF
E552  B0 22      BCS  $E576
E554  A9 00      LDA #$00
E556  8D EA 02   STA  $02EA
E559  8D EB 02   STA  $02EB
E55C  20 95 E6   JSR  $E695
(C) INIT IOCB FOR OPEN
E55F  B0 E6      BCS  $E547
E561  20 EA E6   JSR  $E6EA
E564  A9 0B      LDA #$0B
E566  85 17      STA  $17
E568  20 95 E6   JSR  $E695
E56B  A5 2C      LDA  $2C
E56D  85 26      STA  $26
E56F  A5 2D      LDA  $2D
E571  85 27      STA  $27
E573  4C 72 E6   JMP  $E672
E576  20 F9 EE   JSR  $EEF9
(C) POLL PERIPH FOR OPEN
E579  4C 70 E6   JMP  $E670
E57C  A0 01      LDY #$01
(C) EXECUTE CLOSE COMMAND
E57E  84 23      STY  $23
E580  20 95 E6   JSR  $E695
E583  B0 03      BCS  $E588
E585  20 EA E6   JSR  $E6EA
E588  A9 FF      LDA #$FF
E58A  85 20      STA  $20
E58C  A9 E4      LDA #$E4
E58E  85 27      STA  $27
E590  A9 DB      LDA #$DB
E592  85 26      STA  $26
E594  4C 72 E6   JMP  $E672
E597  A5 20      LDA  $20
(C) EXECUTE GET COMMAND
E599  C9 FF      CMP #$FF
E59B  D0 05      BNE  $E5A2
E59D  20 FF E6   JSR  $E6FF
E5A0  B0 A5      BCS  $E547
E5A2  20 95 E6   JSR  $E695
E5A5  20 EA E6   JSR  $E6EA
E5A8  A6 2E      LDX  $2E
E5AA  BD 40 03   LDA  $0340,X
E5AD  85 20      STA  $20
E5AF  4C 72 E6   JMP  $E672
E5B2  A5 22      LDA  $22
E5B4  25 2A      AND  $2A
E5B6  D0 05      BNE  $E5BD
E5B8  A0 83      LDY #$83
E5BA  4C 70 E6   JMP  $E670
E5BD  20 95 E6   JSR  $E695
E5C0  B0 F8      BCS  $E5BA
E5C2  A5 28      LDA  $28
E5C4  05 29      ORA  $29
E5C6  D0 08      BNE  $E5D0
E5C8  20 EA E6   JSR  $E6EA
E5CB  85 2F      STA  $2F
E5CD  4C 72 E6   JMP  $E672
E5D0  20 EA E6   JSR  $E6EA
E5D3  85 2F      STA  $2F
E5D5  30 41      BMI  $E618
E5D7  A0 00      LDY #$00
E5D9  91 24      STA ($24),Y
E5DB  20 D1 E6   JSR  $E6D1
E5DE  A5 22      LDA  $22
E5E0  29 02      AND #$02
E5E2  D0 0C      BNE  $E5F0
E5E4  A5 2F      LDA  $2F
E5E6  C9 9B      CMP #$9B
E5E8  D0 06      BNE  $E5F0
E5EA  20 BB E6   JSR  $E6BB
E5ED  4C 18 E6   JMP  $E618
E5F0  20 BB E6   JSR  $E6BB
E5F3  D0 DB      BNE  $E5D0
E5F5  A5 22      LDA  $22
E5F7  29 02      AND #$02
E5F9  D0 1D      BNE  $E618
E5FB  20 EA E6   JSR  $E6EA
E5FE  85 2F      STA  $2F
E600  30 0A      BMI  $E60C
E602  A5 2F      LDA  $2F
E604  C9 9B      CMP #$9B
E606  D0 F3      BNE  $E5FB
E608  A9 89      LDA #$89
E60A  85 23      STA  $23
E60C  20 C8 E6   JSR  $E6C8
E60F  A0 00      LDY #$00
E611  A9 9B      LDA #$9B
E613  91 24      STA ($24),Y
E615  20 D1 E6   JSR  $E6D1
E618  20 D8 E6   JSR  $E6D8
E61B  4C 72 E6   JMP  $E672
E61E  A5 22      LDA  $22
(C) EXECUTE PUT COMMAND
E620  25 2A      AND  $2A
```

```
E622  D0 05     BNE  $E629
E624  A0 87     LDY #$87
E626  4C 70 E6  JMP  $E670
E629  20 95 E6  JSR  $E695
E62C  B0 F8     BCS  $E626
E62E  A5 28     LDA  $28
E630  05 29     ORA  $29
E632  D0 06     BNE  $E63A
E634  A5 2F     LDA  $2F
E636  E6 28     INC  $28
E638  D0 06     BNE  $E640
E63A  A0 00     LDY #$00
E63C  B1 24     LDA ($24),Y
E63E  85 2F     STA  $2F
E640  20 EA E6  JSR  $E6EA
E643  08        PHP
E644  20 D1 E6  JSR  $E6D1
E647  20 BB E6  JSR  $E6BB
E64A  28        PLP
E64B  30 1D     BMI  $E66A
E64D  A5 22     LDA  $22
E64F  29 02     AND #$02
E651  D0 06     BNE  $E659
E653  A5 2F     LDA  $2F
E655  C9 9B     CMP #$9B
E657  F0 11     BEQ  $E66A
E659  A5 28     LDA  $28
E65B  05 29     ORA  $29
E65D  D0 DB     BNE  $E63A
E65F  A5 22     LDA  $22
E661  29 02     AND #$02
E663  D0 05     BNE  $E66A
E665  A9 9B     LDA #$9B
E667  20 EA E6  JSR  $E6EA
E66A  20 D8 E6  JSR  $E6D8
E66D  4C 72 E6  JMP  $E672
E670  84 23     STY  $23
(C) SET STATUS
E672  A4 2E     LDY  $2E
(C) COMPLETE CIO OPERATION
E674  B9 44 03  LDA  $0344,Y
E677  85 24     STA  $24
E679  B9 45 03  LDA  $0345,Y
E67C  85 25     STA  $25
E67E  A2 00     LDX #$00
E680  8E E9 02  STX  $02E9
E683  B5 20     LDA  $20,X
E685  99 40 03  STA  $0340,Y
E688  E8        INX
E689  C8        INY
E68A  E0 0C     CPX #$0C
E68C  90 F5     BCC  $E683
E68E  A5 2F     LDA  $2F
E690  A6 2E     LDX  $2E
E692  A4 23     LDY  $23
E694  60        RTS
E695  A4 20     LDY  $20
(C) COMPUTE HANDLER ENTRY
E697  C0 22     CPY #$22
E699  90 04     BCC  $E69F
E69B  A0 85     LDY #$85
E69D  B0 1B     BCS  $E6BA
E69F  B9 1B 03  LDA  $031B,Y
E6A2  85 2C     STA  $2C
E6A4  B9 1C 03  LDA  $031C,Y
E6A7  85 2D     STA  $2D
E6A9  A4 17     LDY  $17
E6AB  B9 2A E7  LDA  $E72A,Y
E6AE  A8        TAY
E6AF  B1 2C     LDA ($2C),Y
E6B1  AA        TAX
E6B2  C8        INY
E6B3  B1 2C     LDA ($2C),Y
E6B5  85 2D     STA  $2D
E6B7  86 2C     STX  $2C
E6B9  18        CLC
E6BA  60        RTS
E6BB  A5 28     LDA  $28
(C) DECREMENT BUFFER LENGTH
E6BD  D0 02     BNE  $E6C1
E6BF  C6 29     DEC  $29
E6C1  C6 28     DEC  $28
E6C3  A5 28     LDA  $28
E6C5  05 29     ORA  $29
E6C7  60        RTS
E6C8  A5 24     LDA  $24
(C) DECREMENT BUFFER POINTER
E6CA  D0 02     BNE  $E6CE
E6CC  C6 25     DEC  $25
E6CE  C6 24     DEC  $24
E6D0  60        RTS
E6D1  E6 24     INC  $24
(C) INCREMENT BUFFER POINTER
E6D3  D0 02     BNE  $E6D7
E6D5  E6 25     INC  $25
E6D7  60        RTS
E6D8  A6 2E     LDX  $2E
(C) SET FINAL BUFFER LENGTH
E6DA  38        SEC
E6DB  BD 48 03  LDA  $0348,X
E6DE  E5 28     SBC  $28
E6E0  85 28     STA  $28
E6E2  BD 49 03  LDA  $0349,X
E6E5  E5 29     SBC  $29
E6E7  85 29     STA  $29
E6E9  60        RTS
E6EA  A0 92     LDY #$92
(C) EXECUTE HANDLER COMMAND
E6EC  20 F4 E6  JSR  $E6F4
E6EF  84 23     STY  $23
```

APPENDIX E6:

```
E6F1   C0 00         CPY #$00
E6F3   60            RTS
E6F4   AA            TAX
(C) INVOKE DEVICE HANDLER
E6F5   A5 2D         LDA  $2D
E6F7   48            PHA
E6F8   A5 2C         LDA  $2C
E6FA   48            PHA
E6FB   8A            TXA
E6FC   A6 2E         LDX  $2E
E6FE   60            RTS
E6FF   38            SEC
(C) SEARCH HANDLER TABLE
E700   A0 01         LDY #$01
E702   B1 24         LDA ($24),Y
E704   E9 31         SBC #$31
E706   30 04         BMI  $E70C
E708   C9 09         CMP #$09
E70A   90 02         BCC  $E70E
E70C   A9 00         LDA #$00
E70E   85 21         STA  $21
E710   E6 21         INC  $21
E712   A0 00         LDY #$00
E714   B1 24         LDA ($24),Y
E716   F0 0C         BEQ  $E724
(C) FIND DEVICE HANDLER
E718   A0 21         LDY #$21
E71A   D9 1A 03      CMP  $031A,Y
E71D   F0 09         BEQ  $E728
E71F   88            DEY
E720   88            DEY
E721   88            DEY
E722   10 F6         BPL  $E71A
E724   A0 82         LDY #$82
E726   38            SEC
E727   60            RTS
E728   98            TYA
E729   85 20         STA  $20
E72B   18            CLC
E72C   60            RTS

E72D   00 04 04 04 04 06 06 06
E735   06 02 08 0A

E739   A5 08         LDA  $08
(C) PERIPHERAL HANDLER
E73B   F0 25         BEQ  $E762
(C) LOADER INITIALIZATION
E73D   A9 E9         LDA #$E9
E73F   85 4A         STA  $4A
E741   A9 03         LDA #$03
E743   85 4B         STA  $4B
E745   A0 12         LDY #$12
E747   18            CLC
E748   B1 4A         LDA ($4A),Y
E74A   AA            TAX
E74B   C8            INY
E74C   71 4A         ADC ($4A),Y
E74E   F0 26         BEQ  $E776
E750   B1 4A         LDA ($4A),Y
E752   85 4B         STA  $4B
E754   86 4A         STX  $4A
E756   20 56 CB      JSR  $CB56
E759   D0 1B         BNE  $E776
E75B   20 94 E8      JSR  $E894
E75E   B0 16         BCS  $E776
E760   90 E3         BCC  $E745
E762   A9 00         LDA #$00
E764   8D FB 03      STA  $03FB
E767   8D FC 03      STA  $03FC
E76A   A9 4F         LDA #$4F
E76C   D0 2D         BNE  $E79B
E76E   A9 00         LDA #$00
E770   A8            TAY
E771   20 BE E7      JSR  $E7BE
E774   10 01         BPL  $E777
E776   60            RTS
E777   18            CLC
E778   AD E7 02      LDA  $02E7
E77B   6D EA 02      ADC  $02EA
E77E   8D 12 03      STA  $0312
E781   AD E8 02      LDA  $02E8
E784   6D EB 02      ADC  $02EB
E787   8D 13 03      STA  $0313
E78A   38            SEC
E78B   AD E5 02      LDA  $02E5
E78E   ED 12 03      SBC  $0312
E791   AD E6 02      LDA  $02E6
E794   ED 13 03      SBC  $0313
E797   B0 09         BCS  $E7A2
E799   A9 4E         LDA #$4E
E79B   A8            TAY
E79C   20 BE E7      JSR  $E7BE
E79F   4C 6E E7      JMP  $E76E
E7A2   AD EC 02      LDA  $02EC
E7A5   AE E7 02      LDX  $02E7
E7A8   8E EC 02      STX  $02EC
E7AB   AE E8 02      LDX  $02E8
E7AE   8E ED 02      STX  $02ED
E7B1   20 DE E7      JSR  $E7DE
E7B4   30 E3         BMI  $E799
E7B6   38            SEC
E7B7   20 9E E8      JSR  $E89E
E7BA   B0 DD         BCS  $E799
E7BC   90 B0         BCC  $E76E
E7BE   48            PHA
(C) PERFORM POLL
E7BF   A2 09         LDX #$09
E7C1   BD D4 E7      LDA  $E7D4,X
E7C4   9D 00 03      STA  $0300,X
```

```
E7C7   CA            DEX
E7C8   10 F7         BPL   $E7C1
E7CA   8C 0B 03      STY   $030B
E7CD   68            PLA
E7CE   8D 0A 03      STA   $030A
E7D1   4C 59 E4      JMP   $E459

E7D4   4F 01 40 40 EA 02 1E 00
E7DC   04 00

E7DE   8D 13 03      STA   $0313
(C) LOAD HANDLER
E7E1   A2 00         LDX #$00
E7E3   8E 12 03      STX   $0312
E7E6   CA            DEX
E7E7   8E 15 03      STX   $0315
E7EA   AD EC 02      LDA   $02EC
E7ED   6A            ROR A
E7EE   90 08         BCC   $E7F8
E7F0   EE EC 02      INC   $02EC
E7F3   D0 03         BNE   $E7F8
E7F5   EE ED 02      INC   $02ED
E7F8   AD EC 02      LDA   $02EC
E7FB   8D D1 02      STA   $02D1
E7FE   AD ED 02      LDA   $02ED
E801   8D D2 02      STA   $02D2
E804   A9 16         LDA #$16
E806   8D CF 02      STA   $02CF
E809   A9 E8         LDA #$E8
E80B   8D D0 02      STA   $02D0
E80E   A9 80         LDA #$80
E810   8D D3 02      STA   $02D3
E813   4C 45 C7      JMP   $C745
E816   AE 15 03      LDX   $0315
(C) GET BYTE
E819   E8            INX
E81A   8E 15 03      STX   $0315
E81D   F0 08         BEQ   $E827
E81F   AE 15 03      LDX   $0315
E822   BD 7D 03      LDA   $037D,X
E825   18            CLC
E826   60            RTS
E827   A9 80         LDA #$80
E829   8D 15 03      STA   $0315
E82C   20 33 E8      JSR   $E833
E82F   10 EE         BPL   $E81F
E831   38            SEC
E832   60            RTS
E833   A2 0B         LDX #$0B
(C) GET NEXT LOAD BLOCK
E835   BD 51 E8      LDA   $E851,X
E838   9D 00 03      STA   $0300,X
E83B   CA            DEX
E83C   10 F7         BPL   $E835
E83E   AE 12 03      LDX   $0312
```

```
E841   8E 0A 03      STX   $030A
E844   E8            INX
E845   8E 12 03      STX   $0312
E848   AD 13 03      LDA   $0313
E84B   8D 00 03      STA   $0300
E84E   4C 59 E4      JMP   $E459

E851   00 01 26 40 FD 03 1E 00
E859   80 00 00 00

E85D   8C 12 03      STY   $0312
(C) SEARCH HANDLER CHAIN
E860   8D 13 03      STA   $0313
E863   A9 E9         LDA #$E9
E865   85 4A         STA   $4A
E867   A9 03         LDA #$03
E869   85 4B         STA   $4B
E86B   A0 12         LDY #$12
E86D   B1 4A         LDA ($4A),Y
E86F   AA            TAX
E870   C8            INY
E871   B1 4A         LDA ($4A),Y
E873   CD 13 03      CMP   $0313
E876   D0 07         BNE   $E87F
E878   EC 12 03      CPX   $0312
E87B   D0 02         BNE   $E87F
E87D   18            CLC
E87E   60            RTS
E87F   C9 00         CMP #$00
E881   D0 06         BNE   $E889
E883   E0 00         CPX #$00
E885   D0 02         BNE   $E889
E887   38            SEC
E888   60            RTS
E889   86 4A         STX   $4A
E88B   85 4B         STA   $4B
E88D   20 56 CB      JSR   $CB56
E890   D0 F5         BNE   $E887
E892   F0 D7         BEQ   $E86B
E894   38            SEC
(C) HANDLER WARMSTART INIT
E895   08            PHP
E896   B0 28         BCS   $E8C0
E898   8D ED 02      STA   $02ED
(C) WARMSTART INIT WITH
E89B   8C EC 02      STY   $02EC
(C) CHAINING
E89E   08            PHP
(C) COLDSTART INIT
E89F   A9 00         LDA #$00
E8A1   A8            TAY
E8A2   20 5D E8      JSR   $E85D
E8A5   B0 27         BCS   $E8CE
E8A7   A0 12         LDY #$12
E8A9   AD EC 02      LDA   $02EC
```

```
E8AC   91 4A       STA ($4A),Y
E8AE   AA          TAX
E8AF   C8          INY
E8B0   AD ED 02    LDA  $02ED
E8B3   91 4A       STA ($4A),Y
E8B5   86 4A       STX  $4A
E8B7   85 4B       STA  $4B
E8B9   A9 00       LDA #$00
E8BB   91 4A       STA ($4A),Y
E8BD   88          DEY
E8BE   91 4A       STA ($4A),Y
E8C0   20 00 E9    JSR  $E900
(C) INIT HANDLER & UPDATE
E8C3   90 0C       BCC  $E8D1
(C) MEMLO
E8C5   AD ED 02    LDA  $02ED
E8C8   AC EC 02    LDY  $02EC
E8CB   20 15 E9    JSR  $E915
E8CE   28          PLP
E8CF   38          SEC
E8D0   60          RTS
E8D1   28          PLP
E8D2   B0 09       BCS  $E8DD
E8D4   A9 00       LDA #$00
E8D6   A0 10       LDY #$10
E8D8   91 4A       STA ($4A),Y
E8DA   C8          INY
E8DB   91 4A       STA ($4A),Y
E8DD   18          CLC
E8DE   A0 10       LDY #$10
E8E0   AD E7 02    LDA  $02E7
E8E3   71 4A       ADC ($4A),Y
E8E5   8D E7 02    STA  $02E7
E8E8   C8          INY
E8E9   AD E8 02    LDA  $02E8
E8EC   71 4A       ADC ($4A),Y
E8EE   8D E8 02    STA  $02E8
E8F1   A0 0F       LDY #$0F
E8F3   A9 00       LDA #$00
E8F5   91 4A       STA ($4A),Y
E8F7   20 56 CB    JSR  $CB56
E8FA   A0 0F       LDY #$0F
E8FC   91 4A       STA ($4A),Y
E8FE   18          CLC
E8FF   60          RTS
E900   18          CLC
(C) INITIALIZE HANDLER
E901   A5 4A       LDA  $4A
E903   69 0C       ADC #$0C
E905   8D 12 03    STA  $0312
E908   A5 4B       LDA  $4B
E90A   69 00       ADC #$00
E90C   8D 13 03    STA  $0313
E90F   6C 12 03    JMP ($0312)
E912   4C 72 C2    JMP  $C272
E915   20 5D E8    JSR  $E85D
(C) HANDLER UNLINKING
E918   B0 3B       BCS  $E955
E91A   A8          TAY
E91B   A5 4A       LDA  $4A
E91D   48          PHA
E91E   A5 4B       LDA  $4B
E920   48          PHA
E921   86 4A       STX  $4A
E923   84 4B       STY  $4B
E925   AD 44 02    LDA  $0244
E928   D0 0F       BNE  $E939
E92A   A0 10       LDY #$10
E92C   18          CLC
E92D   B1 4A       LDA ($4A),Y
E92F   C8          INY
E930   71 4A       ADC ($4A),Y
E932   D0 1F       BNE  $E953
E934   20 56 CB    JSR  $CB56
E937   D0 1A       BNE  $E953
E939   A0 12       LDY #$12
E93B   B1 4A       LDA ($4A),Y
E93D   AA          TAX
E93E   C8          INY
E93F   B1 4A       LDA ($4A),Y
E941   A8          TAY
E942   68          PLA
E943   85 4B       STA  $4B
E945   68          PLA
E946   85 4A       STA  $4A
E948   98          TYA
E949   A0 13       LDY #$13
E94B   91 4A       STA ($4A),Y
E94D   88          DEY
E94E   8A          TXA
E94F   91 4A       STA ($4A),Y
E951   18          CLC
E952   60          RTS
E953   68          PLA
E954   68          PLA
E955   38          SEC
E956   60          RTS

E957   00 00
(C) UNUSED
E959   4C 33 C9    JMP  $C933
E95C   A9 3C       LDA #$3C
(C)W SIO INIT
E95E   8D 02 D3    STA  $D302
E961   A9 3C       LDA #$3C
E963   8D 03 D3    STA  $D303
E966   A9 03       LDA #$03
E968   8D 32 02    STA  $0232
E96B   85 41       STA  $41
```

```
E96D  8D 0F D2   STA  $D20F
E970  60         RTS
E971  BA         TSX
(C) SIO MAIN ROUTINE
E972  8E 18 03   STX  $0318
E975  A9 01      LDA #$01
E977  85 42      STA  $42
E979  AD 00 03   LDA  $0300
E97C  C9 60      CMP #$60
E97E  D0 03      BNE  $E983
E980  4C 9D EB   JMP  $EB9D
E983  A9 00      LDA #$00
E985  8D 0F 03   STA  $030F
E988  A9 01      LDA #$01
E98A  8D BD 02   STA  $02BD
E98D  A9 0D      LDA #$0D
E98F  8D 9C 02   STA  $029C
E992  A9 28      LDA #$28
E994  8D 04 D2   STA  $D204
E997  A9 00      LDA #$00
E999  8D 06 D2   STA  $D206
E99C  18         CLC
E99D  AD 00 03   LDA  $0300
E9A0  6D 01 03   ADC  $0301
E9A3  69 FF      ADC #$FF
E9A5  8D 3A 02   STA  $023A
E9A8  AD 02 03   LDA  $0302
E9AB  8D 3B 02   STA  $023B
E9AE  AD 0A 03   LDA  $030A
E9B1  8D 3C 02   STA  $023C
E9B4  AD 0B 03   LDA  $030B
E9B7  8D 3D 02   STA  $023D
E9BA  18         CLC
E9BB  A9 3A      LDA #$3A
E9BD  85 32      STA  $32
E9BF  69 04      ADC #$04
E9C1  85 34      STA  $34
E9C3  A9 02      LDA #$02
E9C5  85 33      STA  $33
E9C7  85 35      22
E9F3  20 9A EC   JSR  $EC9A
E9F6  A9 00      LDA #$00
E9F8  8D 3F 02   STA  $023F
E9FB  20 C0 EC   JSR  $ECC0
E9FE  F0 12      BEQ  $EA12
EA00  2C 03 03   BIT  $0303
EA03  70 07      BVS  $EA0C
EA05  AD 3F 02   LDA  $023F
EA08  D0 18      BNE  $EA22
EA0A  F0 1E      BEQ  $EA2A
EA0C  20 87 EB   JSR  $EB87
EA0F  20 FD EA   JSR  $EAFD
EA12  AD 3F 02   LDA  $023F
EA15  F0 05      BEQ  $EA1C
EA17  AD 19 03   LDA  $0319
EA1A  85 30      STA  $30
EA1C  A5 30      LDA  $30
EA1E  C9 01      CMP #$01
EA20  F0 08      BEQ  $EA2A
EA22  CE BD 02   DEC  $02BD
EA25  30 03      BMI  $EA2A
EA27  4C 8D E9   JMP  $E98D
EA2A  20 84 EC   JSR  $EC84
(C) COMPLETE SIO OPERATION
EA2D  A9 00      LDA #$00
EA2F  85 42      STA  $42
EA31  A4 30      LDY  $30
EA33  8C 03 03   STY  $0303
EA36  60         RTS
EA37  A9 00      LDA #$00
(C) WAIT FOR COMPLETION
EA39  8D 3F 02   STA  $023F
(C) OR ACK
EA3C  18         CLC
EA3D  A9 3E      LDA #$3E
EA3F  85 32      STA  $32
EA41  69 01      ADC #$01
EA43  85 34      STA  $34
EA45  A9 02      LDA #$02
EA47  85 33      STA  $33
EA49  85 35      STA  $35
EA4B  A9 FF      LDA #$FF
EA4D  85 3C      STA  $3C
EA4F  20 FD EA   JSR  $EAFD
EA52  A0 FF      LDY #$FF
EA54  A5 30      LDA  $30
EA56  C9 01      CMP #$01
EA58  D0 19      BNE  $EA73
EA5A  AD 3E 02   LDA  $023E
EA5D  C9 41      CMP #$41
EA5F  F0 21      BEQ  $EA82
EA61  C9 43      CMP #$43
EA63  F0 1D      BEQ  $EA82
EA65  C9 45      CMP #$45
EA67  D0 06      BNE  $EA6F
EA69  A9 90      LDA #$90
EA6B  85 30      STA  $30
EA6D  D0 04      BNE  $EA73
EA6F  A9 8B      LDA #$8B
EA71  85 30      STA  $30
EA73  A5 30      LDA  $30
EA75  C9 8A      CMP #$8A
EA77  F0 07      BEQ  $EA80
EA79  A9 FF      LDA #$FF
EA7B  8D 3F 02   STA  $023F
EA7E  D0 02      BNE  $EA82
EA80  A0 00      LDY #$00
EA82  A5 30      LDA  $30
EA84  8D 19 03   STA  $0319
EA87  60         RTS
```

APPENDIX E6:

```
EA88  A9 01       LDA #$01
(C) SEND BUFFER TO
EA8A  85 30       STA  $30
(C) SERIAL BUS
EA8C  20 17 EC    JSR  $EC17
EA8F  A0 00       LDY #$00
EA91  84 31       STY  $31
EA93  84 3B       STY  $3B
EA95  84 3A       STY  $3A
EA97  B1 32       LDA ($32),Y
EA99  8D 0D D2    STA  $D20D
EA9C  85 31       STA  $31
EA9E  A5 11       LDA  $11
EAA0  D0 03       BNE  $EAA5
EAA2  4C C7 ED    JMP  $EDC7
EAA5  A5 3A       LDA  $3A
EAA7  F0 F5       BEQ  $EA9E
EAA9  20 84 EC    JSR  $EC84
EAAC  60          RTS
EAAD  98          TYA
(C) SERIAL O/P READY IRQ
EAAE  48          PHA
EAAF  E6 32       INC  $32
EAB1  D0 02       BNE  $EAB5
EAB3  E6 33       INC  $33
EAB5  A5 32       LDA  $32
EAB7  C5 34       CMP  $34
EAB9  A5 33       LDA  $33
EABB  E5 35       SBC  $35
EABD  90 1C       BCC  $EADB
EABF  A5 3B       LDA  $3B
EAC1  D0 0B       BNE  $EACE
EAC3  A5 31       LDA  $31
EAC5  8D 0D D2    STA  $D20D
EAC8  A9 FF       LDA #$FF
EACA  85 3B       STA  $3B
EACC  D0 09       BNE  $EAD7
EACE  A5 10       LDA  $10
EAD0  09 08       ORA #$08
EAD2  85 10       STA  $10
EAD4  8D 0E D2    STA  $D20E
EAD7  68          PLA
EAD8  A8          TAY
EAD9  68          PLA
EADA  40          RTI
EADB  A0 00       LDY #$00
EADD  B1 32       LDA ($32),Y
EADF  8D 0D D2    STA  $D20D
EAE2  18          CLC
EAE3  65 31       ADC  $31
EAE5  69 00       ADC #$00
EAE7  85 31       STA  $31
EAE9  4C D7 EA    JMP  $EAD7
EAEC  A5 3B       LDA  $3B
(C) SERIAL O/P COMPLETE
EAEE  F0 0B       BEQ  $EAFB
EAF0  85 3A       STA  $3A
EAF2  A5 10       LDA  $10
EAF4  29 F7       AND #$F7
EAF6  85 10       STA  $10
EAF8  8D 0E D2    STA  $D20E
EAFB  68          PLA
EAFC  40          RTI
EAFD  A9 00       LDA #$00
(C) RECEIVE
EAFF  AC 0F 03    LDY  $030F
EB02  D0 02       BNE  $EB06
EB04  85 31       STA  $31
EB06  85 38       STA  $38
EB08  85 39       STA  $39
EB0A  A9 01       LDA #$01
EB0C  85 30       STA  $30
EB0E  20 40 EC    JSR  $EC40
EB11  A9 3C       LDA #$3C
EB13  8D 03 D3    STA  $D303
EB16  A5 11       LDA  $11
EB18  D0 03       BNE  $EB1D
EB1A  4C C7 ED    JMP  $EDC7
EB1D  AD 17 03    LDA  $0317
EB20  F0 05       BEQ  $EB27
EB22  A5 39       LDA  $39
EB24  F0 F0       BEQ  $EB16
EB26  60          RTS
EB27  A9 8A       LDA #$8A
(C) INDICATE TIMEOUT
EB29  85 30       STA  $30
EB2B  60          RTS
EB2C  98          TYA
(C) SERIAL I/P READY IRQ
EB2D  48          PHA
EB2E  AD 0F D2    LDA  $D20F
EB31  8D 0A D2    STA  $D20A
EB34  30 04       BMI  $EB3A
EB36  A0 8C       LDY #$8C
EB38  84 30       STY  $30
EB3A  29 20       AND #$20
EB3C  D0 04       BNE  $EB42
EB3E  A0 8E       LDY #$8E
EB40  84 30       STY  $30
EB42  A5 38       LDA  $38
EB44  F0 13       BEQ  $EB59
EB46  AD 0D D2    LDA  $D20D
EB49  C5 31       CMP  $31
EB4B  F0 04       BEQ  $EB51
EB4D  A0 8F       LDY #$8F
EB4F  84 30       STY  $30
EB51  A9 FF       LDA #$FF
EB53  85 39       STA  $39
EB55  68          PLA
EB56  A8          TAY
```

```
EB57   68          PLA
EB58   40          RTI
EB59   AD 0D D2    LDA   $D20D
EB5C   A0 00       LDY  #$00
EB5E   91 32       STA  ($32),Y
EB60   18          CLC
EB61   65 31       ADC   $31
EB63   69 00       ADC  #$00
EB65   85 31       STA   $31
EB67   E6 32       INC   $32
EB69   D0 02       BNE   $EB6D
EB6B   E6 33       INC   $33
EB6D   A5 32       LDA   $32
EB6F   C5 34       CMP   $34
EB71   A5 33       LDA   $33
EB73   E5 35       SBC   $35
EB75   90 DE       BCC   $EB55
EB77   A5 3C       LDA   $3C
EB79   F0 06       BEQ   $EB81
EB7B   A9 00       LDA  #$00
EB7D   85 3C       STA   $3C
EB7F   F0 D0       BEQ   $EB51
EB81   A9 FF       LDA  #$FF
EB83   85 38       STA   $38
EB85   D0 CE       BNE   $EB55
EB87   18          CLC
(C) SET BUFFER POINTERS
EB88   AD 04 03    LDA   $0304
EB8B   85 32       STA   $32
EB8D   6D 08 03    ADC   $0308
EB90   85 34       STA   $34
EB92   AD 05 03    LDA   $0305
EB95   85 33       STA   $33
EB97   6D 09 03    ADC   $0309
EB9A   85 35       STA   $35
EB9C   60          RTS
EB9D   AD 03 03    LDA   $0303
(C) CASSETTE I/O
EBA0   10 32       BPL   $EBD4
EBA2   A9 CC       LDA  #$CC
EBA4   8D 04 D2    STA   $D204
EBA7   A9 05       LDA  #$05
EBA9   8D 06 D2    STA   $D206
EBAC   20 17 EC    JSR   $EC17
EBAF   A6 62       LDX   $62
EBB1   BC 15 EE    LDY   $EE15,X
EBB4   AD 0B 03    LDA   $030B
EBB7   30 03       BMI   $EBBC
EBB9   BC 11 EE    LDY   $EE11,X
EBBC   A2 00       LDX  #$00
EBBE   20 E2 ED    JSR   $EDE2
EBC1   A9 34       LDA  #$34
EBC3   8D 02 D3    STA   $D302
EBC6   AD 17 03    LDA   $0317
EBC9   D0 FB       BNE   $EBC6
EBCB   20 87 EB    JSR   $EB87
EBCE   20 88 EA    JSR   $EA88
EBD1   4C 04 EC    JMP   $EC04
EBD4   A9 FF       LDA  #$FF
EBD6   8D 0F 03    STA   $030F
EBD9   A6 62       LDX   $62
EBDB   BC 17 EE    LDY   $EE17,X
EBDE   AD 0B 03    LDA   $030B
EBE1   30 03       BMI   $EBE6
EBE3   BC 13 EE    LDY   $EE13,X
EBE6   A2 00       LDX  #$00
EBE8   20 E2 ED    JSR   $EDE2
EBEB   A9 34       LDA  #$34
EBED   8D 02 D3    STA   $D302
EBF0   AD 17 03    LDA   $0317
EBF3   D0 FB       BNE   $EBF0
EBF5   20 87 EB    JSR   $EB87
EBF8   20 9A EC    JSR   $EC9A
EBFB   20 E2 ED    JSR   $EDE2
EBFE   20 3D ED    JSR   $ED3D
EC01   20 FD EA    JSR   $EAFD
EC04   AD 0B 03    LDA   $030B
EC07   30 05       BMI   $EC0E
EC09   A9 3C       LDA  #$3C
EC0B   8D 02 D3    STA   $D302
EC0E   4C 2A EA    JMP   $EA2A
EC11   A9 00       LDA  #$00
(C) TIMER EXPIRATION
EC13   8D 17 03    STA   $0317
EC16   60          RTS
EC17   A9 07       LDA  #$07
(C) ENABLE SIO SEND
EC19   2D 32 02    AND   $0232
EC1C   09 20       ORA  #$20
EC1E   AC 00 03    LDY   $0300
EC21   C0 60       CPY  #$60
EC23   D0 0C       BNE   $EC31
EC25   09 08       ORA  #$08
EC27   A0 07       LDY  #$07
EC29   8C 02 D2    STY   $D202
EC2C   A0 05       LDY  #$05
EC2E   8C 00 D2    STY   $D200
EC31   8D 32 02    STA   $0232
EC34   8D 0F D2    STA   $D20F
EC37   A9 C7       LDA  #$C7
EC39   25 10       AND   $10
EC3B   09 10       ORA  #$10
EC3D   4C 56 EC    JMP   $EC56
EC40   A9 07       LDA  #$07
(C) ENABLE SIO RECEIVE
EC42   2D 32 02    AND   $0232
EC45   09 10       ORA  #$10
EC47   8D 32 02    STA   $0232
EC4A   8D 0F D2    STA   $D20F
EC4D   8D 0A D2    STA   $D20A
```

APPENDIX E6:

```
EC50  A9 C7       LDA #$C7
EC52  25 10       AND  $10
EC54  09 20       ORA #$20
EC56  85 10       STA  $10
(C) SET FOR SEND OR RECEIVE
EC58  8D 0E D2    STA  $D20E
EC5B  A9 28       LDA #$28
EC5D  8D 08 D2    STA  $D208
EC60  A2 06       LDX #$06
EC62  A9 A8       LDA #$A8
EC64  A4 41       LDY  $41
EC66  D0 02       BNE  $EC6A
EC68  A9 A0       LDA #$A0
EC6A  9D 01 D2    STA  $D201,X
EC6D  CA          DEX
EC6E  CA          DEX
EC6F  10 F9       BPL  $EC6A
EC71  A9 A0       LDA #$A0
EC73  8D 05 D2    STA  $D205
EC76  AC 00 03    LDY  $0300
EC79  C0 60       CPY #$60
EC7B  F0 06       BEQ  $EC83
EC7D  8D 01 D2    STA  $D201
EC80  8D 03 D2    STA  $D203
EC83  60          RTS
EC84  EA          NOP
(C) DISABLE SEND OR RECEIVE
EC85  A9 C7       LDA #$C7
EC87  25 10       AND  $10
EC89  85 10       STA  $10
EC8B  8D 0E D2    STA  $D20E
EC8E  A2 06       LDX #$06
EC90  A9 00       LDA #$00
EC92  9D 01 D2    STA  $D201,X
EC95  CA          DEX
EC96  CA          DEX
EC97  10 F9       BPL  $EC92
EC99  60          RTS
EC9A  AD 06 03    LDA  $0306
(C) GET DEVICE TIMEOUT
EC9D  6A          ROR A
EC9E  6A          ROR A
EC9F  A8          TAY
ECA0  29 3F       AND #$3F
ECA2  AA          TAX
ECA3  98          TYA
ECA4  6A          ROR A
ECA5  29 C0       AND #$C0
ECA7  A8          TAY
ECA8  60          RTS

ECA9  2C EB AD EA EC EA
(C) SIO INTERRUPT HANDLERS
```

```
ECAF  A2 01       LDX #$01
(C) SEND TO INTELLIGENT
ECB1  A0 FF       LDY #$FF
(C) DEVICE
ECB3  88          DEY
ECB4  D0 FD       BNE  $ECB3
ECB6  CA          DEX
ECB7  D0 F8       BNE  $ECB1
ECB9  20 88 EA    JSR  $EA88
ECBC  A0 02       LDY #$02
ECBE  A2 00       LDX #$00
ECC0  20 E2 ED    JSR  $EDE2
(C) SET TIMER & WAIT
ECC3  20 37 EA    JSR  $EA37
ECC6  98          TYA
ECC7  60          RTS
ECC8  8D 10 03    STA  $0310
(C) COMPUTE BAUD RATE
ECCB  8C 11 03    STY  $0311
ECCE  20 2E ED    JSR  $ED2E
ECD1  8D 10 03    STA  $0310
ECD4  AD 0C 03    LDA  $030C
ECD7  20 2E ED    JSR  $ED2E
ECDA  8D 0C 03    STA  $030C
ECDD  AD 10 03    LDA  $0310
ECE0  38          SEC
ECE1  ED 0C 03    SBC  $030C
ECE4  8D 12 03    STA  $0312
ECE7  AD 11 03    LDA  $0311
ECEA  38          SEC
ECEB  ED 0D 03    SBC  $030D
ECEE  A8          TAY
ECEF  A6 62       LDX  $62
ECF1  A9 00       LDA #$00
ECF3  38          SEC
ECF4  FD 19 EE    SBC  $EE19,X
ECF7  18          CLC
ECF8  7D 19 EE    ADC  $EE19,X
ECFB  88          DEY
ECFC  10 F9       BPL  $ECF7
ECFE  18          CLC
ECFF  6D 12 03    ADC  $0312
ED02  A8          TAY
ED03  4A          LSR A
ED04  4A          LSR A
ED05  4A          LSR A
ED06  0A          ASL A
ED07  38          SEC
ED08  E9 16       SBC #$16
ED0A  AA          TAX
ED0B  98          TYA
ED0C  29 07       AND #$07
ED0E  A8          TAY
ED0F  A9 F5       LDA #$F5
ED11  18          CLC
```

```
ED12   69 0B       ADC #$0B
ED14   88          DEY
ED15   10 FA       BPL  $ED11
ED17   A0 00       LDY #$00
ED19   38          SEC
ED1A   E9 07       SBC #$07
ED1C   10 01       BPL  $ED1F
ED1E   88          DEY
ED1F   18          CLC
ED20   7D F9 ED    ADC  $EDF9,X
ED23   8D EE 02    STA  $02EE
ED26   98          TYA
ED27   7D FA ED    ADC  $EDFA,X
ED2A   8D EF 02    STA  $02EF
ED2D   60          RTS
ED2E   C9 7C       CMP #$7C
(C) ADJUST VCOUNT VALUE
ED30   30 04       BMI  $ED36
ED32   38          SEC
ED33   E9 7C       SBC #$7C
ED35   60          RTS
ED36   18          CLC
ED37   A6 62       LDX  $62
ED39   7D 1B EE    ADC  $EE1B,X
ED3C   60          RTS
ED3D   A5 11       LDA  $11
(C) SET INITIAL BAUD RATE
ED3F   D0 03       BNE  $ED44
ED41   4C C7 ED    JMP  $EDC7
ED44   78          SEI
ED45   AD 17 03    LDA  $0317
ED48   D0 02       BNE  $ED4C
ED4A   F0 25       BEQ  $ED71
ED4C   AD 0F D2    LDA  $D20F
ED4F   29 10       AND #$10
ED51   D0 EA       BNE  $ED3D
ED53   8D 16 03    STA  $0316
ED56   AE 0B D4    LDX  $D40B
ED59   A4 14       LDY  $14
ED5B   8E 0C 03    STX  $030C
ED5E   8C 0D 03    STY  $030D
ED61   A2 01       LDX #$01
ED63   8E 15 03    STX  $0315
ED66   A0 0A       LDY #$0A
ED68   A5 11       LDA  $11
ED6A   F0 5B       BEQ  $EDC7
ED6C   AD 17 03    LDA  $0317
ED6F   D0 04       BNE  $ED75
ED71   58          CLI
ED72   4C 27 EB    JMP  $EB27
ED75   AD 0F D2    LDA  $D20F
ED78   29 10       AND #$10
ED7A   CD 16 03    CMP  $0316
ED7D   F0 E9       BEQ  $ED68
ED7F   8D 16 03    STA  $0316
ED82   88          DEY
ED83   D0 E3       BNE  $ED68
ED85   CE 15 03    DEC  $0315
ED88   30 0C       BMI  $ED96
ED8A   AD 0B D4    LDA  $D40B
ED8D   A4 14       LDY  $14
ED8F   20 C8 EC    JSR  $ECC8
ED92   A0 09       LDY #$09
ED94   D0 D2       BNE  $ED68
ED96   AD EE 02    LDA  $02EE
ED99   8D 04 D2    STA  $D204
ED9C   AD EF 02    LDA  $02EF
ED9F   8D 06 D2    STA  $D206
EDA2   A9 00       LDA #$00
EDA4   8D 0F D2    STA  $D20F
EDA7   AD 32 02    LDA  $0232
EDAA   8D 0F D2    STA  $D20F
EDAD   A9 55       LDA #$55
EDAF   91 32       STA ($32),Y
EDB1   C8          INY
EDB2   91 32       STA ($32),Y
EDB4   A9 AA       LDA #$AA
EDB6   85 31       STA  $31
EDB8   18          CLC
EDB9   A5 32       LDA  $32
EDBB   69 02       ADC #$02
EDBD   85 32       STA  $32
EDBF   A5 33       LDA  $33
EDC1   69 00       ADC #$00
EDC3   85 33       STA  $33
EDC5   58          CLI
EDC6   60          RTS
EDC7   20 84 EC    JSR  $EC84
(C) PROCESS BREAK-KEY
EDCA   A9 3C       LDA #$3C
EDCC   8D 02 D3    STA  $D302
EDCF   A9 3C       LDA #$3C
EDD1   8D 03 D3    STA  $D303
EDD4   A9 80       LDA #$80
EDD6   85 30       STA  $30
EDD8   AE 18 03    LDX  $0318
EDDB   9A          TXS
EDDC   C6 11       DEC  $11
EDDE   58          CLI
EDDF   4C 2A EA    JMP  $EA2A
EDE2   A9 11       LDA #$11
(C) SET SIO VBLANK
EDE4   8D 26 02    STA  $0226
(C) PARAMETERS
EDE7   A9 EC       LDA #$EC
EDE9   8D 27 02    STA  $0227
EDEC   A9 01       LDA #$01
EDEE   78          SEI
EDEF   20 5C E4    JSR  $E45C
EDF2   A9 01       LDA #$01
```

```
EDF4  8D 17 03        STA  $0317
EDF7  58              CLI
EDF8  60              RTS

EDF9  E8 03 43 04 9E 04 F9 04
(C) POKEY FREQUENCY
EE01  54 05 AF 05 0A 06 65 06
(C) VALUES
EE09  C0 06 1A 07 75 07 D0 07

EE11  B4 96 78 64 0F 0D 0A 08
(C) NTSC/PAL CONSTANT
EE19  83 9C 07 20 18 10 0A 0A
(C) VALUES

EE1D  18 10 0A 0A 10 1C 34 64
(C) SCREEN MEMORY ALLOC.
EE25  C4 C4 C4 C4 1C 10 64 C4
EE2D  17 17 0B 17 2F 2F 5F 5F
(C) DL ENTRY COUNTS
EE35  61 61 61 61 17 0B BF 61
EE3D  13 13 09 13 27 27 4F 4F
EE45  41 41 41 41 13 09 9F 41
EE4D  02 06 07 08 09 0A 0B 0D
(C) ANTIC GRAPHIC MODES
EE55  0F 0F 0F 0F 04 05 0C 0E
EE5D  00 00 00 00 00 00 00 01
(C) DL VULNERABILITY
EE65  01 01 01 01 00 00 01 01
EE6D  03 02 02 01 01 02 02 03
(C) LEFT SHIFT COLUMNS
EE75  03 03 03 03 03 03 02 03
EE7D  28 14 14 28 50 50 A0 A0
(C) MODE COLUMN COUNTS
EE85  40 50 50 50 28 28 A0 A0
EE8D  18 18 0C 18 30 30 60 60
(C) MODE ROW COUNTS
EE95  C0 C0 C0 C0 18 0C C0 C0
EE9D  00 00 00 02 03 02 03 02
(C) RIGHT SHIFT COUNTS
EEA5  03 01 01 01 00 00 03 02
EEAD  FF F0 0F C0 30 0C 03 80
(C) DISPLAY MASKS
EEB5  40 20 10 08 04 02 01

EEBC  48              PHA
(C) PERIPHERAL HANDLER ENTRY
EEBD  98              TYA
EEBE  48              PHA
EEBF  8A              TXA
EEC0  A2 00           LDX #$00
EEC2  DD 1A 03        CMP  $031A,X
EEC5  F0 1E           BEQ  $EEE5
EEC7  E8              INX
EEC8  E8              INX
EEC9  E8              INX
EECA  E0 22           CPX #$22
EECC  30 F4           BMI  $EEC2
EECE  A2 00           LDX #$00
EED0  A8              TAY
EED1  A9 00           LDA #$00
EED3  DD 1A 03        CMP  $031A,X
EED6  F0 13           BEQ  $EEEB
EED8  E8              INX
EED9  E8              INX
EEDA  E8              INX
EEDB  E0 22           CPX #$22
EEDD  30 F4           BMI  $EED3
EEDF  68              PLA
EEE0  68              PLA
EEE1  A0 FF           LDY #$FF
EEE3  38              SEC
EEE4  60              RTS
EEE5  68              PLA
EEE6  A8              TAY
EEE7  68              PLA
EEE8  E8              INX
EEE9  38              SEC
EEEA  60              RTS
EEEB  98              TYA
EEEC  9D 1A 03        STA  $031A,X
EEEF  68              PLA
EEF0  9D 1B 03        STA  $031B,X
EEF3  68              PLA
EEF4  9D 1C 03        STA  $031C,X
EEF7  18              CLC
EEF8  60              RTS
EEF9  A0 00           LDY #$00
(C) PERIPHERAL HANDLER
EEFB  B1 24           LDA ($24),Y
(C) POLL AT OPEN
EEFD  A4 21           LDY  $21
EEFF  20 BE E7        JSR  $E7BE
EF02  10 03           BPL  $EF07
EF04  A0 82           LDY #$82
EF06  60              RTS
EF07  A9 7F           LDA #$7F
EF09  85 20           STA  $20
EF0B  A9 25           LDA #$25
EF0D  85 26           STA  $26
EF0F  A9 EF           LDA #$EF
EF11  85 27           STA  $27
EF13  AD EC 02        LDA  $02EC
EF16  AE 2E 00        LDX  $002E
EF19  9D 4D 03        STA  $034D,X
EF1C  A0 00           LDY #$00
EF1E  B1 24           LDA ($24),Y
EF20  9D 4C 03        STA  $034C,X
EF23  A0 01           LDY #$01
EF25  60              RTS
```

```
EF26  48              PHA
(C) PUT BYTE FOR
EF27  8A              TXA
(C) PROVISIONALLY OPEN
EF28  48              PHA
(C) IOCB
EF29  29 0F           AND #$0F
EF2B  D0 10           BNE  $EF3D
EF2D  E0 80           CPX #$80
EF2F  10 0C           BPL  $EF3D
EF31  AD E9 02        LDA  $02E9
EF34  D0 0B           BNE  $EF41
EF36  A0 82           LDY #$82
EF38  68              PLA
EF39  68              PLA
EF3A  C0 00           CPY #$00
EF3C  60              RTS
EF3D  A0 86           LDY #$86
EF3F  30 F7           BMI  $EF38
EF41  8E 2E 00        STX  $002E
EF44  A0 00           LDY #$00
EF46  BD 40 03        LDA  $0340,X
EF49  99 20 00        STA  $0020,Y
EF4C  E8              INX
EF4D  C8              INY
EF4E  C0 0C           CPY #$0C
EF50  30 F4           BMI  $EF46
EF52  20 29 CA        JSR  $CA29
EF55  30 E1           BMI  $EF38
EF57  68              PLA
EF58  AA              TAX
EF59  68              PLA
EF5A  A8              TAY
EF5B  A5 27           LDA  $27
EF5D  48              PHA
EF5E  A5 26           LDA  $26
EF60  48              PHA
EF61  98              TYA
EF62  A0 92           LDY #$92
EF64  60              RTS

EF65  00 00 00 00 00 00
(C) UNUSED

EF6B  4C 05 FD        JMP  $FD05
EF6E  A9 FF           LDA #$FF
(C) INITIALIZE SCREEN
EF70  8D FC 02        STA  $02FC
EF73  AD E4 02        LDA  $02E4
EF76  85 6A           STA  $6A
EF78  A9 40           LDA #$40
EF7A  8D BE 02        STA  $02BE
EF7D  A9 51           LDA #$51
EF7F  85 79           STA  $79
EF81  A9 FB           LDA #$FB
EF83  85 7A           STA  $7A
EF85  A9 11           LDA #$11
EF87  85 60           STA  $60
EF89  A9 FC           LDA #$FC
EF8B  85 61           STA  $61
EF8D  60              RTS
EF8E  A5 2B           LDA  $2B
(C) SCREEN OPEN
EF90  29 0F           AND #$0F
EF92  D0 08           BNE  $EF9C
EF94  A5 2A           LDA  $2A
(C) EDITOR OPEN
EF96  29 0F           AND #$0F
EF98  85 2A           STA  $2A
EF9A  A9 00           LDA #$00
EF9C  85 57           STA  $57
(C) COMPLETE OPEN
EF9E  C9 10           CMP #$10
EFA0  90 05           BCC  $EFA7
EFA2  A9 91           LDA #$91
EFA4  4C 54 F1        JMP  $F154
EFA7  A9 E0           LDA #$E0
EFA9  8D F4 02        STA  $02F4
EFAC  A9 CC           LDA #$CC
EFAE  8D 6B 02        STA  $026B
EFB1  A9 02           LDA #$02
EFB3  8D F3 02        STA  $02F3
EFB6  8D 2F 02        STA  $022F
EFB9  A9 01           LDA #$01
EFBB  85 4C           STA  $4C
EFBD  A9 C0           LDA #$C0
EFBF  05 10           ORA  $10
EFC1  85 10           STA  $10
EFC3  8D 0E D2        STA  $D20E
EFC6  A9 40           LDA #$40
EFC8  8D 0E D4        STA  $D40E
EFCB  2C 6E 02        BIT  $026E
EFCE  10 0C           BPL  $EFDC
EFD0  A9 C4           LDA #$C4
EFD2  8D 00 02        STA  $0200
EFD5  A9 FC           LDA #$FC
EFD7  8D 01 02        STA  $0201
EFDA  A9 C0           LDA #$C0
EFDC  8D 0E D4        STA  $D40E
EFDF  A9 00           LDA #$00
EFE1  8D 93 02        STA  $0293
EFE4  85 64           STA  $64
EFE6  85 7B           STA  $7B
EFE8  8D F0 02        STA  $02F0
EFEB  A0 0E           LDY #$0E
EFED  A9 01           LDA #$01
EFEF  99 A3 02        STA  $02A3,Y
EFF2  88              DEY
EFF3  10 FA           BPL  $EFEF
EFF5  A2 04           LDX #$04
```

APPENDIX E6:

```
EFF7  BD 08 FB   LDA  $FB08,X
EFFA  9D C4 02   STA  $02C4,X
EFFD  CA         DEX
EFFE  10 F7      BPL  $EFF7
F000  A4 6A      LDY  $6A
F002  88         DEY
F003  8C 95 02   STY  $0295
F006  A9 60      LDA #$60
F008  8D 94 02   STA  $0294
F00B  A6 57      LDX  $57
F00D  BD 4D EE   LDA  $EE4D,X
F010  85 51      STA  $51
F012  A5 6A      LDA  $6A
F014  85 65      STA  $65
F016  BC 1D EE   LDY  $EE1D,X
F019  A9 28      LDA #$28
F01B  20 7A F5   JSR  $F57A
F01E  88         DEY
F01F  D0 F8      BNE  $F019
F021  AD 6F 02   LDA  $026F
F024  29 3F      AND #$3F
F026  85 67      STA  $67
F028  A8         TAY
F029  E0 08      CPX #$08
F02B  90 1F      BCC  $F04C
F02D  E0 0F      CPX #$0F
F02F  F0 0D      BEQ  $F03E
F031  E0 0C      CPX #$0C
F033  B0 17      BCS  $F04C
F035  8A         TXA
F036  6A         ROR A
F037  6A         ROR A
F038  6A         ROR A
F039  29 C0      AND #$C0
F03B  05 67      ORA  $67
F03D  A8         TAY
F03E  A9 10      LDA #$10
F040  20 7A F5   JSR  $F57A
F043  E0 0B      CPX #$0B
F045  D0 05      BNE  $F04C
F047  A9 06      LDA #$06
F049  8D C8 02   STA  $02C8
F04C  8C 6F 02   STY  $026F
F04F  A5 64      LDA  $64
F051  85 58      STA  $58
F053  A5 65      LDA  $65
F055  85 59      STA  $59
F057  AD 0B D4   LDA  $D40B
F05A  C9 7A      CMP #$7A
F05C  D0 F9      BNE  $F057
F05E  20 78 F5   JSR  $F578
F061  BD 5D EE   LDA  $EE5D,X
F064  F0 06      BEQ  $F06C
F066  A9 FF      LDA #$FF
F068  85 64      STA  $64
F06A  C6 65      DEC  $65
F06C  20 65 F5   JSR  $F565
F06F  A5 64      LDA  $64
F071  85 68      STA  $68
F073  A5 65      LDA  $65
F075  85 69      STA  $69
F077  A9 41      LDA #$41
F079  20 70 F5   JSR  $F570
F07C  86 66      STX  $66
F07E  A9 18      LDA #$18
F080  8D BF 02   STA  $02BF
F083  A5 57      LDA  $57
F085  C9 0C      CMP #$0C
F087  B0 04      BCS  $F08D
F089  C9 09      CMP #$09
F08B  B0 39      BCS  $F0C6
F08D  A5 2A      LDA  $2A
F08F  29 10      AND #$10
F091  F0 33      BEQ  $F0C6
F093  A9 04      LDA #$04
F095  8D BF 02   STA  $02BF
F098  A2 02      LDX #$02
F09A  AD 6E 02   LDA  $026E
F09D  F0 03      BEQ  $F0A2
F09F  20 A0 F5   JSR  $F5A0
F0A2  A9 02      LDA #$02
F0A4  20 69 F5   JSR  $F569
F0A7  CA         DEX
F0A8  10 F8      BPL  $F0A2
F0AA  A4 6A      LDY  $6A
F0AC  88         DEY
F0AD  98         TYA
F0AE  20 70 F5   JSR  $F570
F0B1  A9 60      LDA #$60
F0B3  20 70 F5   JSR  $F570
F0B6  A9 42      LDA #$42
F0B8  20 69 F5   JSR  $F569
F0BB  18         CLC
F0BC  A9 10      LDA #$10
F0BE  65 66      ADC  $66
F0C0  A8         TAY
F0C1  BE 2D EE   LDX  $EE2D,Y
F0C4  D0 15      BNE  $F0DB
F0C6  A4 66      LDY  $66
F0C8  BE 2D EE   LDX  $EE2D,Y
F0CB  A5 57      LDA  $57
F0CD  D0 0C      BNE  $F0DB
F0CF  AD 6E 02   LDA  $026E
F0D2  F0 07      BEQ  $F0DB
F0D4  20 A0 F5   JSR  $F5A0
F0D7  A9 22      LDA #$22
F0D9  85 51      STA  $51
F0DB  A5 51      LDA  $51
F0DD  20 70 F5   JSR  $F570
F0E0  CA         DEX
```

```
F0E1   D0 F8       BNE  $F0DB
F0E3   A5 57       LDA  $57
F0E5   C9 08       CMP #$08
F0E7   90 26       BCC  $F10F
F0E9   C9 0F       CMP #$0F
F0EB   F0 04       BEQ  $F0F1
F0ED   C9 0C       CMP #$0C
F0EF   B0 1E       BCS  $F10F
F0F1   A2 5D       LDX #$5D
F0F3   A5 6A       LDA  $6A
F0F5   38          SEC
F0F6   E9 10       SBC #$10
F0F8   20 70 F5    JSR  $F570
F0FB   A9 00       LDA #$00
F0FD   20 70 F5    JSR  $F570
F100   A5 51       LDA  $51
F102   09 40       ORA #$40
F104   20 70 F5    JSR  $F570
F107   A5 51       LDA  $51
F109   20 70 F5    JSR  $F570
F10C   CA          DEX
F10D   D0 F8       BNE  $F107
F10F   A5 59       LDA  $59
F111   20 70 F5    JSR  $F570
F114   A5 58       LDA  $58
F116   20 70 F5    JSR  $F570
F119   A5 51       LDA  $51
F11B   09 40       ORA #$40
F11D   20 70 F5    JSR  $F570
F120   A9 70       LDA #$70
F122   20 70 F5    JSR  $F570
F125   A9 70       LDA #$70
F127   20 70 F5    JSR  $F570
F12A   A5 64       LDA  $64
F12C   8D 30 02    STA  $0230
F12F   A5 65       LDA  $65
F131   8D 31 02    STA  $0231
F134   A9 70       LDA #$70
F136   20 70 F5    JSR  $F570
F139   A5 64       LDA  $64
F13B   8D E5 02    STA  $02E5
F13E   A5 65       LDA  $65
F140   8D E6 02    STA  $02E6
F143   A0 01       LDY #$01
F145   AD 30 02    LDA  $0230
F148   91 68       STA ($68),Y
F14A   C8          INY
F14B   AD 31 02    LDA  $0231
F14E   91 68       STA ($68),Y
F150   A5 4C       LDA  $4C
F152   10 10       BPL  $F164
F154   8D EC 03    STA  $03EC
F157   20 94 EF    JSR  $EF94
F15A   AD EC 03    LDA  $03EC
F15D   A0 00       LDY #$00
```

```
F15F   8C EC 03    STY  $03EC
F162   A8          TAY
F163   60          RTS
F164   A5 2A       LDA  $2A
F166   29 20       AND #$20
F168   D0 0B       BNE  $F175
F16A   20 20 F4    JSR  $F420
F16D   8D 90 02    STA  $0290
F170   A5 52       LDA  $52
F172   8D 91 02    STA  $0291
F175   A9 22       LDA #$22
F177   0D 2F 02    ORA  $022F
F17A   8D 2F 02    STA  $022F
F17D   4C 0B F2    JMP  $F20B
F180   20 CA F6    JSR  $F6CA
(C) SCREEN GET BYTE
F183   20 8F F1    JSR  $F18F
F186   20 6A F7    JSR  $F76A
F189   20 0A F6    JSR  $F60A
F18C   4C 1E F2    JMP  $F21E
F18F   20 AC F5    JSR  $F5AC
(C) GET DATA UNDER CURSOR
F192   B1 64       LDA ($64),Y
F194   2D A0 02    AND  $02A0
F197   46 6F       LSR  $6F
F199   B0 03       BCS  $F19E
F19B   4A          LSR A
F19C   10 F9       BPL  $F197
F19E   8D FA 02    STA  $02FA
F1A1   C9 00       CMP #$00
F1A3   60          RTS
F1A4   8D FB 02    STA  $02FB
(C) SCREEN PUT BYTE
F1A7   C9 7D       CMP #$7D
F1A9   D0 06       BNE  $F1B1
F1AB   20 20 F4    JSR  $F420
F1AE   4C 0B F2    JMP  $F20B
F1B1   20 CA F6    JSR  $F6CA
(C) CHECK FOR END-OF-LINE
F1B4   AD FB 02    LDA  $02FB
F1B7   C9 9B       CMP #$9B
F1B9   D0 06       BNE  $F1C1
F1BB   20 61 F6    JSR  $F661
F1BE   4C 0B F2    JMP  $F20B
F1C1   20 CA F1    JSR  $F1CA
F1C4   20 0E F6    JSR  $F60E
F1C7   4C 0B F2    JMP  $F20B
F1CA   AD FF 02    LDA  $02FF
(C) PLOT POINT
F1CD   D0 FB       BNE  $F1CA
F1CF   A2 02       LDX #$02
F1D1   B5 54       LDA  $54,X
F1D3   95 5A       STA  $5A,X
F1D5   CA          DEX
F1D6   10 F9       BPL  $F1D1
```

APPENDIX E6:

```
F1D8  AD FB 02    LDA  $02FB
F1DB  A8          TAY
F1DC  2A          ROL A
F1DD  2A          ROL A
F1DE  2A          ROL A
F1DF  2A          ROL A
F1E0  29 03       AND #$03
F1E2  AA          TAX
F1E3  98          TYA
F1E4  29 9F       AND #$9F
F1E6  1D 49 FB    ORA  $FB49,X
F1E9  8D FA 02    STA  $02FA
(C) DISPLAY
F1EC  20 AC F5    JSR  $F5AC
F1EF  AD FA 02    LDA  $02FA
F1F2  46 6F       LSR  $6F
F1F4  B0 04       BCS  $F1FA
F1F6  0A          ASL A
F1F7  4C F2 F1    JMP  $F1F2
F1FA  2D A0 02    AND  $02A0
F1FD  85 50       STA  $50
F1FF  AD A0 02    LDA  $02A0
F202  49 FF       EOR #$FF
F204  31 64       AND ($64),Y
F206  05 50       ORA  $50
F208  91 64       STA ($64),Y
(C) SET EXIT CONDITIONS
F20A  60          RTS
F20B  20 8F F1    JSR  $F18F
F20E  85 5D       STA  $5D
F210  A6 57       LDX  $57
F212  D0 0A       BNE  $F21E
F214  AE F0 02    LDX  $02F0
F217  D0 05       BNE  $F21E
F219  49 80       EOR #$80
F21B  20 E9 F1    JSR  $F1E9
F21E  A4 4C       LDY  $4C
(C) SCREEN STATUS
F220  4C 26 F2    JMP  $F226
F223  4C FC C8    JMP  $C8FC
(C) EXECUTE SELF-TEST
F226  A9 01       LDA #$01
F228  85 4C       STA  $4C
F22A  AD FB 02    LDA  $02FB
F22D  60          RTS
(C) SCREEN EDITOR SPECIAL
F22E  2C 6E 02    BIT  $026E
(C) SCREEN EDITOR CLOSE
F231  10 EB       BPL  $F21E
F233  A9 40       LDA #$40
F235  8D 0E D4    STA  $D40E
F238  A9 00       LDA #$00
F23A  8D 6E 02    STA  $026E
F23D  A9 CE       LDA #$CE
F23F  8D 00 02    STA  $0200
```

```
F242  A9 C0       LDA #$C0
F244  8D 01 02    STA  $0201
F247  4C 94 EF    JMP  $EF94
F24A  20 62 F9    JSR  $F962
(C) EDITOR GET BYTE
F24D  20 BC F6    JSR  $F6BC
F250  A5 6B       LDA  $6B
F252  D0 34       BNE  $F288
F254  A5 54       LDA  $54
F256  85 6C       STA  $6C
F258  A5 55       LDA  $55
F25A  85 6D       STA  $6D
F25C  20 FD F2    JSR  $F2FD
F25F  84 4C       STY  $4C
F261  AD FB 02    LDA  $02FB
F264  C9 9B       CMP #$9B
F266  F0 12       BEQ  $F27A
F268  20 BE F2    JSR  $F2BE
F26B  20 62 F9    JSR  $F962
F26E  A5 63       LDA  $63
F270  C9 71       CMP #$71
F272  D0 03       BNE  $F277
F274  20 56 F5    JSR  $F556
F277  4C 5C F2    JMP  $F25C
F27A  20 18 F7    JSR  $F718
F27D  20 B1 F8    JSR  $F8B1
F280  A5 6C       LDA  $6C
F282  85 54       STA  $54
F284  A5 6D       LDA  $6D
F286  85 55       STA  $55
F288  A5 6B       LDA  $6B
F28A  F0 11       BEQ  $F29D
F28C  C6 6B       DEC  $6B
F28E  F0 0D       BEQ  $F29D
F290  A5 4C       LDA  $4C
F292  30 F8       BMI  $F28C
F294  20 80 F1    JSR  $F180
F297  8D FB 02    STA  $02FB
F29A  4C 62 F9    JMP  $F962
F29D  20 61 F6    JSR  $F661
F2A0  A9 9B       LDA #$9B
F2A2  8D FB 02    STA  $02FB
F2A5  20 0B F2    JSR  $F20B
F2A8  84 4C       STY  $4C
F2AA  4C 62 F9    JMP  $F962
F2AD  6C 64 00    JMP ($0064)
F2B0  8D FB 02    STA  $02FB
(C) EDITOR PUT BYTE
F2B3  20 62 F9    JSR  $F962
F2B6  20 BC F6    JSR  $F6BC
F2B9  A9 00       LDA #$00
F2BB  8D E8 03    STA  $03E8
F2BE  20 18 F7    JSR  $F718
(C) PROCESS CHARACTER
F2C1  20 3C F9    JSR  $F93C
```

```
F2C4   F0 09      BEQ  $F2CF
F2C6   0E A2 02   ASL  $02A2
F2C9   20 B4 F1   JSR  $F1B4
F2CC   4C 62 F9   JMP  $F962
F2CF   AD FE 02   LDA  $02FE
F2D2   0D A2 02   ORA  $02A2
F2D5   D0 EF      BNE  $F2C6
F2D7   0E A2 02   ASL  $02A2
F2DA   E8         INX
F2DB   AD E8 03   LDA  $03E8
F2DE   F0 05      BEQ  $F2E5
F2E0   8A         TXA
F2E1   18         CLC
F2E2   69 2D      ADC #$2D
F2E4   AA         TAX
F2E5   BD 0D FB   LDA  $FB0D,X
F2E8   85 64      STA  $64
F2EA   BD 0E FB   LDA  $FB0E,X
F2ED   85 65      STA  $65
F2EF   20 AD F2   JSR  $F2AD
F2F2   20 0B F2   JSR  $F20B
F2F5   4C 62 F9   JMP  $F962
F2F8   A9 FF      LDA #$FF
(C) IGNORE CHARACTER
F2FA   8D FC 02   STA  $02FC
(C) AND PERFORM
F2FD   A9 00      LDA #$00
(C) KEYBOARD GET BYTE
F2FF   8D E8 03   STA  $03E8
F302   A5 2A      LDA  $2A
F304   4A         LSR A
F305   B0 6F      BCS  $F376
F307   A9 80      LDA #$80
F309   A6 11      LDX  $11
F30B   F0 65      BEQ  $F372
F30D   AD FC 02   LDA  $02FC
F310   C9 FF      CMP #$FF
F312   F0 E9      BEQ  $F2FD
F314   85 7C      STA  $7C
F316   A2 FF      LDX #$FF
F318   8E FC 02   STX  $02FC
F31B   AE DB 02   LDX  $02DB
F31E   D0 03      BNE  $F323
F320   20 83 F9   JSR  $F983
F323   A8         TAY
F324   C0 C0      CPY #$C0
F326   B0 D0      BCS  $F2F8
F328   B1 79      LDA ($79),Y
F32A   8D FB 02   STA  $02FB
F32D   AA         TAX
F32E   30 03      BMI  $F333
F330   4C B4 F3   JMP  $F3B4
F333   C9 80      CMP #$80
F335   F0 C1      BEQ  $F2F8
F337   C9 81      CMP #$81
F339   D0 0A      BNE  $F345
F33B   AD B6 02   LDA  $02B6
F33E   49 80      EOR #$80
F340   8D B6 02   STA  $02B6
F343   B0 B3      BCS  $F2F8
F345   C9 82      CMP #$82
F347   D0 0C      BNE  $F355
F349   AD BE 02   LDA  $02BE
F34C   F0 0B      BEQ  $F359
F34E   A9 00      LDA #$00
F350   8D BE 02   STA  $02BE
F353   F0 A3      BEQ  $F2F8
F355   C9 83      CMP #$83
F357   D0 07      BNE  $F360
F359   A9 40      LDA #$40
F35B   8D BE 02   STA  $02BE
F35E   D0 98      BNE  $F2F8
F360   C9 84      CMP #$84
F362   D0 08      BNE  $F36C
F364   A9 80      LDA #$80
F366   8D BE 02   STA  $02BE
F369   4C F8 F2   JMP  $F2F8
F36C   C9 85      CMP #$85
F36E   D0 0B      BNE  $F37B
F370   A9 88      LDA #$88
F372   85 4C      STA  $4C
F374   85 11      STA  $11
F376   A9 9B      LDA #$9B
F378   4C DA F3   JMP  $F3DA
F37B   C9 89      CMP #$89
F37D   D0 10      BNE  $F38F
F37F   AD DB 02   LDA  $02DB
F382   49 FF      EOR #$FF
F384   8D DB 02   STA  $02DB
F387   D0 03      BNE  $F38C
F389   20 83 F9   JSR  $F983
F38C   4C F8 F2   JMP  $F2F8
F38F   C9 8E      CMP #$8E
F391   B0 12      BCS  $F3A5
F393   C9 8A      CMP #$8A
F395   90 F5      BCC  $F38C
F397   E9 8A      SBC #$8A
F399   06 7C      ASL  $7C
F39B   10 02      BPL  $F39F
F39D   09 04      ORA #$04
F39F   A8         TAY
F3A0   B1 60      LDA ($60),Y
F3A2   4C 2A F3   JMP  $F32A
F3A5   C9 92      CMP #$92
F3A7   B0 0B      BCS  $F3B4
F3A9   C9 8E      CMP #$8E
F3AB   90 DF      BCC  $F38C
F3AD   E9 72      SBC #$72
F3AF   EE E8 03   INC  $03E8
F3B2   D0 26      BNE  $F3DA
```

APPENDIX E6:

```
F3B4  A5 7C       LDA  $7C
F3B6  C9 40       CMP #$40
F3B8  B0 15       BCS  $F3CF
F3BA  AD FB 02    LDA  $02FB
F3BD  C9 61       CMP #$61
F3BF  90 0E       BCC  $F3CF
F3C1  C9 7B       CMP #$7B
F3C3  B0 0A       BCS  $F3CF
F3C5  AD BE 02    LDA  $02BE
F3C8  F0 05       BEQ  $F3CF
F3CA  05 7C       ORA  $7C
F3CC  4C 23 F3    JMP  $F323
F3CF  20 3C F9    JSR  $F93C
F3D2  F0 09       BEQ  $F3DD
F3D4  AD FB 02    LDA  $02FB
F3D7  4D B6 02    EOR  $02B6
F3DA  8D FB 02    STA  $02FB
F3DD  4C 1E F2    JMP  $F21E
F3E0  A9 80       LDA #$80
(C) ESCAPE CHARACTER HANDLER
F3E2  8D A2 02    STA  $02A2
F3E5  60          RTS
F3E6  C6 54       DEC  $54
(C) MOVE CURSOR UP
F3E8  10 06       BPL  $F3F0
F3EA  AE BF 02    LDX  $02BF
F3ED  CA          DEX
F3EE  86 54       STX  $54
F3F0  4C 0C F9    JMP  $F90C
F3F3  E6 54       INC  $54
(C) MOVE CURSOR DOWN
F3F5  A5 54       LDA  $54
F3F7  CD BF 02    CMP  $02BF
F3FA  90 F4       BCC  $F3F0
F3FC  A2 00       LDX #$00
F3FE  F0 EE       BEQ  $F3EE
F400  C6 55       DEC  $55
(C) MOVE CURSOR LEFT
F402  A5 55       LDA  $55
F404  30 04       BMI  $F40A
F406  C5 52       CMP  $52
F408  B0 04       BCS  $F40E
F40A  A5 53       LDA  $53
(C) CURSOR TO RIGHT MARGIN
F40C  85 55       STA  $55
(C) SET CURSOR COLUMN
F40E  4C 8E F8    JMP  $F88E
F411  E6 55       INC  $55
(C) MOVE CURSOR POINT
F413  A5 55       LDA  $55
F415  C5 53       CMP  $53
F417  90 F5       BCC  $F40E
F419  F0 F3       BEQ  $F40E
F41B  A5 52       LDA  $52
(C) CURSOR TO LEFT MARGIN
F41D  4C 0C F4    JMP  $F40C
F420  20 A6 F9    JSR  $F9A6
(C) CLEAR SCREEN
F423  A4 64       LDY  $64
F425  A9 00       LDA #$00
F427  85 64       STA  $64
F429  91 64       STA ($64),Y
F42B  C8          INY
F42C  D0 FB       BNE  $F429
F42E  E6 65       INC  $65
F430  A6 65       LDX  $65
F432  E4 6A       CPX  $6A
F434  90 F3       BCC  $F429
F436  A9 FF       LDA #$FF
F438  99 B2 02    STA  $02B2,Y
F43B  C8          INY
F43C  C0 04       CPY #$04
F43E  90 F8       BCC  $F438
F440  20 97 F9    JSR  $F997
(C) MOVE CURSOR HOME
F443  85 63       STA  $63
F445  85 6D       STA  $6D
F447  A9 00       LDA #$00
F449  85 54       STA  $54
F44B  85 56       STA  $56
F44D  85 6C       STA  $6C
F44F  60          RTS
F450  A5 63       LDA  $63
F452  C5 52       CMP  $52
F454  F0 21       BEQ  $F477
F456  A5 55       LDA  $55
F458  C5 52       CMP  $52
F45A  D0 03       BNE  $F45F
F45C  20 23 F9    JSR  $F923
F45F  20 00 F4    JSR  $F400
F462  A5 55       LDA  $55
F464  C5 53       CMP  $53
F466  D0 07       BNE  $F46F
F468  A5 54       LDA  $54
F46A  F0 03       BEQ  $F46F
F46C  20 E6 F3    JSR  $F3E6
F46F  A9 20       LDA #$20
F471  8D FB 02    STA  $02FB
F474  20 CA F1    JSR  $F1CA
F477  4C 8E F8    JMP  $F88E
F47A  20 11 F4    JSR  $F411
(C) TAB CHARACTER HANDLER
F47D  A5 55       LDA  $55
F47F  C5 52       CMP  $52
F481  D0 08       BNE  $F48B
F483  20 65 F6    JSR  $F665
F486  20 58 F7    JSR  $F758
F489  B0 07       BCS  $F492
F48B  A5 63       LDA  $63
F48D  20 5D F7    JSR  $F75D
```

```
F490   90 E8      BCC  $F47A
F492   4C 8E F8   JMP  $F88E
F495   A5 63      LDA  $63
(C) SET TAB
F497   4C 3E F7   JMP  $F73E
F49A   A5 63      LDA  $63
(C) CLEAR TAB
F49C   4C 4A F7   JMP  $F74A
F49F   20 4C F9   JSR  $F94C
(C) INSERT CHARACTER
F4A2   20 8F F1   JSR  $F18F
F4A5   85 7D      STA  $7D
F4A7   A9 00      LDA  #$00
F4A9   8D BB 02   STA  $02BB
F4AC   20 E9 F1   JSR  $F1E9
F4AF   A5 63      LDA  $63
F4B1   48         PHA
F4B2   20 12 F6   JSR  $F612
F4B5   68         PLA
F4B6   C5 63      CMP  $63
F4B8   B0 0C      BCS  $F4C6
F4BA   A5 7D      LDA  $7D
F4BC   48         PHA
F4BD   20 8F F1   JSR  $F18F
F4C0   85 7D      STA  $7D
F4C2   68         PLA
F4C3   4C AC F4   JMP  $F4AC
F4C6   20 57 F9   JSR  $F957
F4C9   CE BB 02   DEC  $02BB
F4CC   30 04      BMI  $F4D2
F4CE   C6 54      DEC  $54
F4D0   D0 F7      BNE  $F4C9
F4D2   4C 8E F8   JMP  $F88E
F4D5   20 4C F9   JSR  $F94C
(C) DELETE CHARACTER
F4D8   20 AC F5   JSR  $F5AC
F4DB   A5 64      LDA  $64
F4DD   85 68      STA  $68
F4DF   A5 65      LDA  $65
F4E1   85 69      STA  $69
F4E3   A5 63      LDA  $63
F4E5   48         PHA
F4E6   20 0A F6   JSR  $F60A
F4E9   68         PLA
F4EA   C5 63      CMP  $63
F4EC   B0 10      BCS  $F4FE
F4EE   A5 54      LDA  $54
F4F0   CD BF 02   CMP  $02BF
F4F3   B0 09      BCS  $F4FE
F4F5   20 8F F1   JSR  $F18F
F4F8   A0 00      LDY  #$00
F4FA   91 68      STA  ($68),Y
F4FC   F0 DA      BEQ  $F4D8
F4FE   A0 00      LDY  #$00
F500   98         TYA
F501   91 68      STA  ($68),Y
F503   20 18 F9   JSR  $F918
F506   20 57 F9   JSR  $F957
F509   4C 8E F8   JMP  $F88E
F50C   38         SEC
(C) INSERT LINE
F50D   20 C2 F7   JSR  $F7C2
F510   A5 52      LDA  $52
F512   85 55      STA  $55
F514   20 AC F5   JSR  $F5AC
F517   20 8E F7   JSR  $F78E
F51A   20 E2 F7   JSR  $F7E2
F51D   4C 8E F8   JMP  $F88E
F520   20 8E F8   JSR  $F88E
(C) DELETE LINE
F523   A4 51      LDY  $51
F525   84 54      STY  $54
F527   A4 54      LDY  $54
F529   98         TYA
F52A   38         SEC
F52B   20 5B F7   JSR  $F75B
F52E   08         PHP
F52F   98         TYA
F530   18         CLC
F531   69 78      ADC  #$78
F533   28         PLP
F534   20 3C F7   JSR  $F73C
F537   C8         INY
F538   C0 18      CPY  #$18
F53A   D0 ED      BNE  $F529
F53C   AD B4 02   LDA  $02B4
F53F   09 01      ORA  #$01
F541   8D B4 02   STA  $02B4
F544   A9 00      LDA  #$00
F546   85 55      STA  $55
F548   20 AC F5   JSR  $F5AC
F54B   20 2A F8   JSR  $F82A
F54E   20 58 F7   JSR  $F758
F551   90 D4      BCC  $F527
F553   4C 1B F4   JMP  $F41B
F556   A0 20      LDY  #$20
(C) SOUND BELL
F558   20 83 F9   JSR  $F983
F55B   88         DEY
F55C   10 FA      BPL  $F558
F55E   60         RTS
F55F   20 40 F4   JSR  $F440
(C) CURSOR TO BOTTOM-LEFT
F562   4C E6 F3   JMP  $F3E6
F565   A9 02      LDA  #$02
(C) DOUBLE-BYTE DOUBLE-DEC
F567   D0 11      BNE  $F57A
F569   AC 6E 02   LDY  $026E
(C) STORE DATA FOR FINE
```

APPENDIX E6:

```
F56C  F0 02       BEQ  $F570
(C) SCROLLING
F56E  09 20       ORA #$20
F570  A4 4C       LDY  $4C
F572  30 2B       BMI  $F59F
F574  A0 00       LDY #$00
F576  91 64       STA ($64),Y
F578  A9 01       LDA #$01
(C) DOUBLE-BYTE SINGLE-DEC
F57A  8D 9E 02    STA  $029E
F57D  A5 4C       LDA  $4C
F57F  30 1E       BMI  $F59F
F581  A5 64       LDA  $64
F583  38          SEC
F584  ED 9E 02    SBC  $029E
F587  85 64       STA  $64
F589  B0 02       BCS  $F58D
F58B  C6 65       DEC  $65
F58D  A5 0F       LDA  $0F
F58F  C5 65       CMP  $65
F591  90 0C       BCC  $F59F
F593  D0 06       BNE  $F59B
F595  A5 0E       LDA  $0E
F597  C5 64       CMP  $64
F599  90 04       BCC  $F59F
F59B  A9 93       LDA #$93
F59D  85 4C       STA  $4C
F59F  60          RTS
F5A0  A9 02       LDA #$02
(C) SET SCROLLING DL ENTRY
F5A2  20 70 F5    JSR  $F570
F5A5  A9 A2       LDA #$A2
F5A7  20 70 F5    JSR  $F570
F5AA  CA          DEX
F5AB  60          RTS
F5AC  A2 01       LDX #$01
(C) CONVERT CURSOR ROW/
F5AE  86 66       STX  $66
(C) COLUMN TO ADDRESS
F5B0  CA          DEX
F5B1  86 65       STX  $65
F5B3  A5 54       LDA  $54
F5B5  0A          ASL A
F5B6  26 65       ROL  $65
F5B8  0A          ASL A
F5B9  26 65       ROL  $65
F5BB  65 54       ADC  $54
F5BD  85 64       STA  $64
F5BF  90 02       BCC  $F5C3
F5C1  E6 65       INC  $65
F5C3  A4 57       LDY  $57
F5C5  BE 6D EE    LDX  $EE6D,Y
F5C8  06 64       ASL  $64
F5CA  26 65       ROL  $65
F5CC  CA          DEX
F5CD  D0 F9       BNE  $F5C8
F5CF  A5 56       LDA  $56
F5D1  4A          LSR A
F5D2  A5 55       LDA  $55
F5D4  BE 9D EE    LDX  $EE9D,Y
F5D7  F0 06       BEQ  $F5DF
F5D9  6A          ROR A
F5DA  06 66       ASL  $66
F5DC  CA          DEX
F5DD  D0 FA       BNE  $F5D9
F5DF  65 64       ADC  $64
F5E1  90 02       BCC  $F5E5
F5E3  E6 65       INC  $65
F5E5  18          CLC
F5E6  65 58       ADC  $58
F5E8  85 64       STA  $64
F5EA  85 5E       STA  $5E
F5EC  A5 65       LDA  $65
F5EE  65 59       ADC  $59
F5F0  85 65       STA  $65
F5F2  85 5F       STA  $5F
F5F4  BE 9D EE    LDX  $EE9D,Y
F5F7  BD 04 FB    LDA  $FB04,X
F5FA  25 55       AND  $55
F5FC  65 66       ADC  $66
F5FE  A8          TAY
F5FF  B9 AC EE    LDA  $EEAC,Y
F602  8D A0 02    STA  $02A0
F605  85 6F       STA  $6F
F607  A0 00       LDY #$00
F609  60          RTS
F60A  A9 00       LDA #$00
(C) ADVANCE CURSOR
F60C  F0 02       BEQ  $F610
F60E  A9 9B       LDA #$9B
F610  85 7D       STA  $7D
F612  E6 63       INC  $63
F614  E6 55       INC  $55
F616  D0 02       BNE  $F61A
F618  E6 56       INC  $56
F61A  A5 55       LDA  $55
F61C  A6 57       LDX  $57
F61E  DD 7D EE    CMP  $EE7D,X
F621  F0 0A       BEQ  $F62D
F623  E0 00       CPX #$00
F625  D0 E2       BNE  $F609
F627  C5 53       CMP  $53
F629  F0 DE       BEQ  $F609
F62B  90 DC       BCC  $F609
F62D  E0 08       CPX #$08
F62F  D0 04       BNE  $F635
F631  A5 56       LDA  $56
F633  F0 D4       BEQ  $F609
F635  A5 57       LDA  $57
F637  D0 2C       BNE  $F665
```

```
F639   A5 63       LDA  $63
F63B   C9 51       CMP #$51
F63D   90 0A       BCC  $F649
F63F   A5 7D       LDA  $7D
F641   F0 22       BEQ  $F665
F643   20 61 F6    JSR  $F661
F646   4C AB F6    JMP  $F6AB
F649   20 65 F6    JSR  $F665
F64C   A5 54       LDA  $54
F64E   18          CLC
F64F   69 78       ADC #$78
F651   20 5D F7    JSR  $F75D
F654   90 08       BCC  $F65E
F656   A5 7D       LDA  $7D
F658   F0 04       BEQ  $F65E
F65A   18          CLC
F65B   20 0D F5    JSR  $F50D
F65E   4C 8E F8    JMP  $F88E
F661   A9 9B       LDA #$9B
(C) RETURN WITH SCROLLING
F663   85 7D       STA  $7D
F665   20 97 F9    JSR  $F997
(C) RETURN
F668   A9 00       LDA #$00
F66A   85 56       STA  $56
F66C   E6 54       INC  $54
F66E   A6 57       LDX  $57
F670   A0 18       LDY #$18
F672   24 7B       BIT  $7B
F674   10 05       BPL  $F67B
F676   A0 04       LDY #$04
F678   98          TYA
F679   D0 03       BNE  $F67E
F67B   BD 8D EE    LDA  $EE8D,X
F67E   C5 54       CMP  $54
F680   D0 29       BNE  $F6AB
F682   8C 9D 02    STY  $029D
F685   8A          TXA
F686   D0 23       BNE  $F6AB
F688   A5 7D       LDA  $7D
F68A   F0 1F       BEQ  $F6AB
F68C   C9 9B       CMP #$9B
F68E   F0 01       BEQ  $F691
F690   18          CLC
F691   20 F7 F7    JSR  $F7F7
F694   EE BB 02    INC  $02BB
F697   C6 6C       DEC  $6C
F699   10 02       BPL  $F69D
F69B   E6 6C       INC  $6C
F69D   CE 9D 02    DEC  $029D
F6A0   AD B2 02    LDA  $02B2
F6A3   38          SEC
F6A4   10 EB       BPL  $F691
F6A6   AD 9D 02    LDA  $029D
F6A9   85 54       STA  $54
F6AB   4C 8E F8    JMP  $F88E
F6AE   38          SEC
(C) SUBTRACT END POINT
F6AF   B5 70       LDA  $70,X
F6B1   E5 74       SBC  $74
F6B3   95 70       STA  $70,X
F6B5   B5 71       LDA  $71,X
F6B7   E5 75       SBC  $75
F6B9   95 71       STA  $71,X
F6BB   60          RTS
F6BC   AD BF 02    LDA  $02BF
(C) CHECK CURSOR RANGE
F6BF   C9 04       CMP #$04
F6C1   F0 07       BEQ  $F6CA
F6C3   A5 57       LDA  $57
F6C5   F0 03       BEQ  $F6CA
F6C7   20 94 EF    JSR  $EF94
F6CA   A9 27       LDA #$27
F6CC   C5 53       CMP  $53
F6CE   B0 02       BCS  $F6D2
F6D0   85 53       STA  $53
F6D2   A6 57       LDX  $57
F6D4   BD 8D EE    LDA  $EE8D,X
F6D7   C5 54       CMP  $54
F6D9   90 2A       BCC  $F705
F6DB   F0 28       BEQ  $F705
F6DD   E0 08       CPX #$08
F6DF   D0 0A       BNE  $F6EB
F6E1   A5 56       LDA  $56
F6E3   F0 13       BEQ  $F6F8
F6E5   C9 01       CMP #$01
F6E7   D0 1C       BNE  $F705
F6E9   F0 04       BEQ  $F6EF
F6EB   A5 56       LDA  $56
F6ED   D0 16       BNE  $F705
F6EF   BD 7D EE    LDA  $EE7D,X
F6F2   C5 55       CMP  $55
F6F4   90 0F       BCC  $F705
F6F6   F0 0D       BEQ  $F705
F6F8   A9 01       LDA #$01
F6FA   85 4C       STA  $4C
F6FC   A9 80       LDA #$80
F6FE   A6 11       LDX  $11
F700   85 11       STA  $11
F702   F0 06       BEQ  $F70A
F704   60          RTS
F705   20 40 F4    JSR  $F440
F708   A9 8D       LDA #$8D
F70A   85 4C       STA  $4C
F70C   68          PLA
F70D   68          PLA
F70E   A5 7B       LDA  $7B
F710   10 03       BPL  $F715
F712   4C 62 F9    JMP  $F962
F715   4C 1E F2    JMP  $F21E
```

APPENDIX E6:

```
F718  A0 00     LDY #$00
(C) RESTORE OLD CURSOR DATA
F71A  A5 5F     LDA  $5F
F71C  F0 04     BEQ  $F722
F71E  A5 5D     LDA  $5D
F720  91 5E     STA ($5E),Y
F722  60        RTS
F723  48        PHA
(C) E:/S: BITMAP ROUTINES
F724  29 07     AND #$07
F726  AA        TAX
F727  BD B4 EE  LDA  $EEB4,X
F72A  85 6E     STA  $6E
F72C  68        PLA
F72D  4A        LSR A
F72E  4A        LSR A
F72F  4A        LSR A
F730  AA        TAX
F731  60        RTS
F732  2E B4 02  ROL  $02B4
F735  2E B3 02  ROL  $02B3
F738  2E B2 02  ROL  $02B2
F73B  60        RTS
F73C  90 0C     BCC  $F74A
F73E  20 23 F7  JSR  $F723
F741  BD A3 02  LDA  $02A3,X
F744  05 6E     ORA  $6E
F746  9D A3 02  STA  $02A3,X
F749  60        RTS
F74A  20 23 F7  JSR  $F723
F74D  A5 6E     LDA  $6E
F74F  49 FF     EOR #$FF
F751  3D A3 02  AND  $02A3,X
F754  9D A3 02  STA  $02A3,X
F757  60        RTS
F758  A5 54     LDA  $54
F75A  18        CLC
F75B  69 78     ADC #$78
F75D  20 23 F7  JSR  $F723
F760  18        CLC
F761  BD A3 02  LDA  $02A3,X
F764  25 6E     AND  $6E
F766  F0 01     BEQ  $F769
F768  38        SEC
F769  60        RTS
F76A  AD FA 02  LDA  $02FA
F76D  A4 57     LDY  $57
F76F  C0 0E     CPY #$0E
F771  B0 17     BCS  $F78A
F773  C0 0C     CPY #$0C
F775  B0 04     BCS  $F77B
F777  C0 03     CPY #$03
F779  B0 0F     BCS  $F78A
F77B  2A        ROL A
F77C  2A        ROL A
F77D  2A        ROL A
F77E  2A        ROL A
F77F  29 03     AND #$03
F781  AA        TAX
F782  AD FA 02  LDA  $02FA
F785  29 9F     AND #$9F
F787  1D 4D FB  ORA  $FB4D,X
F78A  8D FB 02  STA  $02FB
F78D  60        RTS
F78E  A6 6A     LDX  $6A
F790  CA        DEX
F791  86 69     STX  $69
F793  86 67     STX  $67
F795  A9 B0     LDA #$B0
F797  85 68     STA  $68
F799  A9 D8     LDA #$D8
F79B  85 66     STA  $66
F79D  A6 54     LDX  $54
F79F  E8        INX
F7A0  EC BF 02  CPX  $02BF
F7A3  F0 E8     BEQ  $F78D
F7A5  A0 27     LDY #$27
F7A7  B1 68     LDA ($68),Y
F7A9  91 66     STA ($66),Y
F7AB  88        DEY
F7AC  10 F9     BPL  $F7A7
F7AE  38        SEC
F7AF  A5 68     LDA  $68
F7B1  85 66     STA  $66
F7B3  E9 28     SBC #$28
F7B5  85 68     STA  $68
F7B7  A5 69     LDA  $69
F7B9  85 67     STA  $67
F7BB  E9 00     SBC #$00
F7BD  85 69     STA  $69
F7BF  4C 9F F7  JMP  $F79F
F7C2  08        PHP
F7C3  A0 16     LDY #$16
F7C5  98        TYA
F7C6  20 5A F7  JSR  $F75A
F7C9  08        PHP
F7CA  98        TYA
F7CB  18        CLC
F7CC  69 79     ADC #$79
F7CE  28        PLP
F7CF  20 3C F7  JSR  $F73C
F7D2  88        DEY
F7D3  30 04     BMI  $F7D9
F7D5  C4 54     CPY  $54
F7D7  B0 EC     BCS  $F7C5
F7D9  A5 54     LDA  $54
F7DB  18        CLC
F7DC  69 78     ADC #$78
F7DE  28        PLP
F7DF  4C 3C F7  JMP  $F73C
```

```
F7E2   A5 52       LDA  $52
F7E4   85 55       STA  $55
F7E6   20 AC F5    JSR  $F5AC
F7E9   38          SEC
F7EA   A5 53       LDA  $53
F7EC   E5 52       SBC  $52
F7EE   A8          TAY
F7EF   A9 00       LDA #$00
F7F1   91 64       STA ($64),Y
F7F3   88          DEY
F7F4   10 FB       BPL  $F7F1
F7F6   60          RTS
F7F7   20 32 F7    JSR  $F732
(C) SCREEN SCROLL
F7FA   AD 6E 02    LDA  $026E
F7FD   F0 28       BEQ  $F827
F7FF   AD 6C 02    LDA  $026C
F802   D0 FB       BNE  $F7FF
F804   A9 08       LDA #$08
F806   8D 6C 02    STA  $026C
F809   AD 6C 02    LDA  $026C
F80C   C9 01       CMP #$01
F80E   D0 F9       BNE  $F809
F810   AD 0B D4    LDA  $D40B
F813   C9 40       CMP #$40
F815   B0 F9       BCS  $F810
F817   A2 0D       LDX #$0D
F819   AD BF 02    LDA  $02BF
F81C   C9 04       CMP #$04
F81E   D0 02       BNE  $F822
F820   A2 70       LDX #$70
F822   EC 0B D4    CPX  $D40B
F825   B0 FB       BCS  $F822
F827   20 A6 F9    JSR  $F9A6
F82A   A5 64       LDA  $64
F82C   A6 65       LDX  $65
F82E   E8          INX
F82F   E4 6A       CPX  $6A
F831   F0 06       BEQ  $F839
F833   38          SEC
F834   E9 10       SBC #$10
F836   4C 2E F8    JMP  $F82E
F839   69 27       ADC #$27
F83B   D0 0A       BNE  $F847
F83D   A6 65       LDX  $65
F83F   E8          INX
F840   E4 6A       CPX  $6A
F842   F0 38       BEQ  $F87C
F844   18          CLC
F845   69 10       ADC #$10
F847   A8          TAY
F848   85 7E       STA  $7E
F84A   38          SEC
F84B   A5 64       LDA  $64
F84D   E5 7E       SBC  $7E
```

```
F84F   85 64       STA  $64
F851   B0 02       BCS  $F855
F853   C6 65       DEC  $65
F855   A5 64       LDA  $64
F857   18          CLC
F858   69 28       ADC #$28
F85A   85 7E       STA  $7E
F85C   A5 65       LDA  $65
F85E   69 00       ADC #$00
F860   85 7F       STA  $7F
F862   B1 7E       LDA ($7E),Y
F864   91 64       STA ($64),Y
F866   C8          INY
F867   D0 F9       BNE  $F862
F869   A0 10       LDY #$10
F86B   A5 64       LDA  $64
F86D   C9 D8       CMP #$D8
F86F   F0 0B       BEQ  $F87C
F871   18          CLC
F872   69 F0       ADC #$F0
F874   85 64       STA  $64
F876   90 DD       BCC  $F855
F878   E6 65       INC  $65
F87A   D0 D9       BNE  $F855
F87C   A6 6A       LDX  $6A
F87E   CA          DEX
F87F   86 7F       STX  $7F
F881   A2 D8       LDX #$D8
F883   86 7E       STX  $7E
F885   A9 00       LDA #$00
F887   A0 27       LDY #$27
F889   91 7E       STA ($7E),Y
F88B   88          DEY
F88C   10 FB       BPL  $F889
F88E   A9 00       LDA #$00
F890   85 63       STA  $63
F892   A5 54       LDA  $54
F894   85 51       STA  $51
F896   A5 51       LDA  $51
F898   20 5A F7    JSR  $F75A
F89B   B0 0C       BCS  $F8A9
F89D   A5 63       LDA  $63
F89F   18          CLC
F8A0   69 28       ADC #$28
F8A2   85 63       STA  $63
F8A4   C6 51       DEC  $51
F8A6   4C 96 F8    JMP  $F896
F8A9   18          CLC
F8AA   A5 63       LDA  $63
F8AC   65 55       ADC  $55
F8AE   85 63       STA  $63
F8B0   60 -        RTS
F8B1   20 4C F9    JSR  $F94C
(C) COMPUTE BUFFER COUNT
F8B4   A5 63       LDA  $63
```

```
F8B6   48          PHA
F8B7   A5 6C       LDA   $6C
F8B9   85 54       STA   $54
F8BB   A5 6D       LDA   $6D
F8BD   85 55       STA   $55
F8BF   A9 01       LDA  #$01
F8C1   85 6B       STA   $6B
F8C3   A2 17       LDX  #$17
F8C5   A5 7B       LDA   $7B
F8C7   10 02       BPL   $F8CB
F8C9   A2 03       LDX  #$03
F8CB   E4 54       CPX   $54
F8CD   D0 0B       BNE   $F8DA
F8CF   A5 55       LDA   $55
F8D1   C5 53       CMP   $53
F8D3   D0 05       BNE   $F8DA
F8D5   E6 6B       INC   $6B
F8D7   4C EA F8    JMP   $F8EA
F8DA   20 0A F6    JSR   $F60A
F8DD   E6 6B       INC   $6B
F8DF   A5 63       LDA   $63
F8E1   C5 52       CMP   $52
F8E3   D0 DE       BNE   $F8C3
F8E5   C6 54       DEC   $54
F8E7   20 00 F4    JSR   $F400
F8EA   20 8F F1    JSR   $F18F
F8ED   D0 17       BNE   $F906
F8EF   C6 6B       DEC   $6B
F8F1   A5 63       LDA   $63
F8F3   C5 52       CMP   $52
F8F5   F0 0F       BEQ   $F906
F8F7   20 00 F4    JSR   $F400
F8FA   A5 55       LDA   $55
F8FC   C5 53       CMP   $53
F8FE   D0 02       BNE   $F902
F900   C6 54       DEC   $54
F902   A5 6B       LDA   $6B
F904   D0 E4       BNE   $F8EA
F906   68          PLA
F907   85 63       STA   $63
F909   4C 57 F9    JMP   $F957
F90C   20 8E F8    JSR   $F88E
F90F   A5 51       LDA   $51
F911   85 6C       STA   $6C
F913   A5 52       LDA   $52
F915   85 6D       STA   $6D
F917   60          RTS
F918   A5 63       LDA   $63
(C) DELETE LINE
F91A   C5 52       CMP   $52
F91C   D0 02       BNE   $F920
F91E   C6 54       DEC   $54
F920   20 8E F8    JSR   $F88E
F923   A5 63       LDA   $63
F925   C5 52       CMP   $52
```

```
F927   F0 EE       BEQ   $F917
F929   20 AC F5    JSR   $F5AC
F92C   A5 53       LDA   $53
F92E   38          SEC
F92F   E5 52       SBC   $52
F931   A8          TAY
F932   B1 64       LDA ($64),Y
F934   D0 E1       BNE   $F917
F936   88          DEY
F937   10 F9       BPL   $F932
F939   4C 27 F5    JMP   $F527
F93C   A2 2D       LDX  #$2D
(C) CHECK FOR CONTROL
F93E   BD 0D FB    LDA   $FB0D,X
F941   CD FB 02    CMP   $02FB
F944   F0 05       BEQ   $F94B
F946   CA          DEX
F947   CA          DEX
F948   CA          DEX
F949   10 F3       BPL   $F93E
F94B   60          RTS
F94C   A2 02       LDX  #$02
(C) SAVE ROW & COLUMN
F94E   B5 54       LDA   $54,X
F950   9D B8 02    STA   $02B8,X
F953   CA          DEX
F954   10 F8       BPL   $F94E
F956   60          RTS
F957   A2 02       LDX  #$02
F959   BD B8 02    LDA   $02B8,X
F95C   95 54       STA   $54,X
F95E   CA          DEX
F95F   10 F8       BPL   $F959
F961   60          RTS
F962   AD BF 02    LDA   $02BF
(C) SWAP CURSOR WITH
F965   C9 18       CMP  #$18
(C) REGULAR CURSOR POSITION
F967   F0 17       BEQ   $F980
F969   A2 0B       LDX  #$0B
F96B   B5 54       LDA   $54,X
F96D   48          PHA
F96E   BD 90 02    LDA   $0290,X
F971   95 54       STA   $54,X
F973   68          PLA
F974   9D 90 02    STA   $0290,X
F977   CA          DEX
F978   10 F1       BPL   $F96B
F97A   A5 7B       LDA   $7B
F97C   49 FF       EOR  #$FF
F97E   85 7B       STA   $7B
F980   4C 1E F2    JMP   $F21E
F983   A2 7E       LDX  #$7E
(C) SOUND KEY CLICK
F985   48          PHA
```

APPENDIX E6:

```
F986   8E 1F D0    STX  $D01F
F989   AD 0B D4    LDA  $D40B
F98C   CD 0B D4    CMP  $D40B
F98F   F0 FB       BEQ  $F98C
F991   CA          DEX
F992   CA          DEX
F993   10 F1       BPL  $F986
F995   68          PLA
F996   60          RTS
F997   A9 00       LDA #$00
(C) CURSOR TO LEFT EDGE
F999   A6 7B       LDX  $7B
F99B   D0 04       BNE  $F9A1
F99D   A6 57       LDX  $57
F99F   D0 02       BNE  $F9A3
F9A1   A5 52       LDA  $52
F9A3   85 55       STA  $55
F9A5   60          RTS
F9A6   A5 58       LDA  $58
(C) SET MEMORY SCAN
F9A8   85 64       STA  $64
(C) COUNTER ADDRESS
F9AA   A5 59       LDA  $59
F9AC   85 65       STA  $65
F9AE   60          RTS
F9AF   A2 00       LDX #$00
(C) SCREEN XIO COMMAND
F9B1   A5 22       LDA  $22
F9B3   C9 11       CMP #$11
F9B5   F0 08       BEQ  $F9BF
F9B7   C9 12       CMP #$12
F9B9   F0 03       BEQ  $F9BE
F9BB   A0 84       LDY #$84
F9BD   60          RTS
F9BE   E8          INX
F9BF   8E B7 02    STX  $02B7
F9C2   A5 54       LDA  $54
F9C4   8D F5 02    STA  $02F5
F9C7   A5 55       LDA  $55
F9C9   8D F6 02    STA  $02F6
F9CC   A5 56       LDA  $56
F9CE   8D F7 02    STA  $02F7
F9D1   A9 01       LDA #$01
F9D3   8D F8 02    STA  $02F8
F9D6   8D F9 02    STA  $02F9
F9D9   38          SEC
F9DA   AD F5 02    LDA  $02F5
F9DD   E5 5A       SBC  $5A
F9DF   85 76       STA  $76
F9E1   B0 0E       BCS  $F9F1
F9E3   A9 FF       LDA #$FF
F9E5   8D F8 02    STA  $02F8
F9E8   A5 76       LDA  $76
F9EA   49 FF       EOR #$FF
F9EC   18          CLC
F9ED   69 01       ADC #$01
F9EF   85 76       STA  $76
F9F1   38          SEC
F9F2   AD F6 02    LDA  $02F6
F9F5   E5 5B       SBC  $5B
F9F7   85 77       STA  $77
F9F9   AD F7 02    LDA  $02F7
F9FC   E5 5C       SBC  $5C
F9FE   85 78       STA  $78
FA00   B0 17       BCS  $FA19
FA02   A9 FF       LDA #$FF
FA04   8D F9 02    STA  $02F9
FA07   A5 77       LDA  $77
FA09   49 FF       EOR #$FF
FA0B   85 77       STA  $77
FA0D   A5 78       LDA  $78
FA0F   49 FF       EOR #$FF
FA11   85 78       STA  $78
FA13   E6 77       INC  $77
FA15   D0 02       BNE  $FA19
FA17   E6 78       INC  $78
FA19   A2 02       LDX #$02
FA1B   A0 00       LDY #$00
FA1D   84 73       STY  $73
FA1F   98          TYA
FA20   95 70       STA  $70,X
FA22   B5 5A       LDA  $5A,X
FA24   95 54       STA  $54,X
FA26   CA          DEX
FA27   10 F6       BPL  $FA1F
FA29   A5 77       LDA  $77
FA2B   E8          INX
FA2C   A8          TAY
FA2D   A5 78       LDA  $78
FA2F   85 7F       STA  $7F
FA31   85 75       STA  $75
FA33   D0 0B       BNE  $FA40
FA35   A5 77       LDA  $77
FA37   C5 76       CMP  $76
FA39   B0 05       BCS  $FA40
FA3B   A5 76       LDA  $76
FA3D   A2 02       LDX #$02
FA3F   A8          TAY
FA40   98          TYA
FA41   85 7E       STA  $7E
FA43   85 74       STA  $74
FA45   48          PHA
FA46   A5 75       LDA  $75
FA48   4A          LSR A
FA49   68          PLA
FA4A   6A          ROR A
FA4B   95 70       STA  $70,X
FA4D   A5 7E       LDA  $7E
FA4F   05 7F       ORA  $7F
FA51   D0 03       BNE  $FA56
```

APPENDIX E6:

```
FA53  4C 01 FB   JMP  $FB01
FA56  18         CLC
FA57  A5 70      LDA  $70
FA59  65 76      ADC  $76
FA5B  85 70      STA  $70
FA5D  90 02      BCC  $FA61
FA5F  E6 71      INC  $71
FA61  A5 71      LDA  $71
FA63  C5 75      CMP  $75
FA65  90 15      BCC  $FA7C
FA67  D0 06      BNE  $FA6F
FA69  A5 70      LDA  $70
FA6B  C5 74      CMP  $74
FA6D  90 0D      BCC  $FA7C
FA6F  18         CLC
FA70  A5 54      LDA  $54
FA72  6D F8 02   ADC  $02F8
FA75  85 54      STA  $54
FA77  A2 00      LDX #$00
FA79  20 AE F6   JSR  $F6AE
FA7C  18         CLC
FA7D  A5 72      LDA  $72
FA7F  65 77      ADC  $77
FA81  85 72      STA  $72
FA83  A5 73      LDA  $73
FA85  65 78      ADC  $78
FA87  85 73      STA  $73
FA89  C5 75      CMP  $75
FA8B  90 28      BCC  $FAB5
FA8D  D0 06      BNE  $FA95
FA8F  A5 72      LDA  $72
FA91  C5 74      CMP  $74
FA93  90 20      BCC  $FAB5
FA95  2C F9 02   BIT  $02F9
FA98  10 10      BPL  $FAAA
FA9A  C6 55      DEC  $55
FA9C  A5 55      LDA  $55
FA9E  C9 FF      CMP #$FF
FAA0  D0 0E      BNE  $FAB0
FAA2  A5 56      LDA  $56
FAA4  F0 0A      BEQ  $FAB0
FAA6  C6 56      DEC  $56
FAA8  10 06      BPL  $FAB0
FAAA  E6 55      INC  $55
FAAC  D0 02      BNE  $FAB0
FAAE  E6 56      INC  $56
FAB0  A2 02      LDX #$02
FAB2  20 AE F6   JSR  $F6AE
FAB5  20 CA F6   JSR  $F6CA
FAB8  20 CA F1   JSR  $F1CA
FABB  AD B7 02   LDA  $02B7
FABE  F0 2F      BEQ  $FAEF
FAC0  20 4C F9   JSR  $F94C
FAC3  AD FB 02   LDA  $02FB
FAC6  8D BC 02   STA  $02BC
```

```
FAC9  A5 54      LDA  $54
FACB  48         PHA
FACC  20 12 F6   JSR  $F612
FACF  68         PLA
FAD0  85 54      STA  $54
FAD2  20 CA F6   JSR  $F6CA
FAD5  20 8F F1   JSR  $F18F
FAD8  D0 0C      BNE  $FAE6
FADA  AD FD 02   LDA  $02FD
FADD  8D FB 02   STA  $02FB
FAE0  20 CA F1   JSR  $F1CA
FAE3  4C C9 FA   JMP  $FAC9
FAE6  AD BC 02   LDA  $02BC
FAE9  8D FB 02   STA  $02FB
FAEC  20 57 F9   JSR  $F957
FAEF  38         SEC
FAF0  A5 7E      LDA  $7E
FAF2  E9 01      SBC #$01
FAF4  85 7E      STA  $7E
FAF6  A5 7F      LDA  $7F
FAF8  E9 00      SBC #$00
FAFA  85 7F      STA  $7F
FAFC  30 03      BMI  $FB01
FAFE  4C 4D FA   JMP  $FA4D
FB01  4C 1E F2   JMP  $F21E

FB04  00 01 03 07
(C) BIT MASKS
FB08  28 CA 94 46 00
(C) SCREEN COLOURS

FB0D  1B E0 F3
FB10  1C E6 F3
FB13  1D F3 F3
FB16  1E 00 F4
FB19  1F 11 F4
FB1C  7D 20 F4
FB1F  7E 50 F4
FB22  7F 7A F4
FB25  9B 61 F6
FB28  9C 20 F5
FB2B  9D 0C F5
FB2E  9E 9A F4
FB31  9F 95 F4
FB34  FD 56 F5
FB37  FE D5 F4
FB3A  FF 9F F4

FB3D  1C 40 F4
FB40  1D 5F F5
FB43  1E 1B F4
FB46  1F 0A F4
```

APPENDIX E6:

```
FB49  40 00 20 60
(C) ASCII - INTERNAL
(C) CONVERSION CONSTANTS
FB4D  20 40 00 60
(C) VICE VERSA

FB51  6C 6A 3B 8A 8B 6B 2B 2A
KEYBOOARD DEFINITION
FB59  6F 80 70 75 9B 69 2D 3D
TABLE
FB61  76 80 63 8C 8D 62 78 7A
FB69  34 80 33 36 1B 35 32 31
FB71  2C 20 2E 6E 80 6D 2F 81
FB79  72 80 65 79 7F 74 77 71
FB81  39 80 30 37 7E 38 3C 3E
FB89  66 68 64 80 82 67 73 61
FB91  4C 4A 3A 8A 8B 4B 5C 5E
FB99  4F 80 50 55 9B 49 5F 7C
FBA1  56 80 43 8C 8D 42 58 5A
FBA9  24 80 23 26 1B 25 22 21
FBB1  5B 20 5D 4E 80 4D 3F 81
FBB9  52 80 45 59 9F 54 57 51
FBC1  28 80 29 27 9C 40 7D 9D
FBC9  46 48 44 80 83 47 53 41
FBD1  0C 0A 7B 80 80 0B 1E 1F
FBD9  0F 80 10 15 9B 09 1C 1D
FBE1  16 80 03 89 80 02 18 1A
FBE9  80 80 85 80 1B 80 FD 80
FBF1  00 20 60 0E 80 0D 80 81
FBF9  12 80 05 19 9E 14 17 11
FC01  80 80 80 80 FE 80 7D FF
FC09  06 08 04 80 84 07 13 01

FC11  1C 1D 1E 1F 8E 8F 90 91
FUNCTION KEY DEFINITIONS

FC19  8A          TXA
KEYBOARD IRQ
FC1A  48          PHA
FC1B  98          TYA
FC1C  48          PHA
FC1D  AC 01 D3    LDY  $D301
FC20  AD 09 D2    LDA  $D209
FC23  CD F2 02    CMP  $02F2
FC26  D0 05       BNE  $FC2D
FC28  AE F1 02    LDX  $02F1
FC2B  D0 49       BNE  $FC76
FC2D  AE 6D 02    LDX  $026D
FC30  C9 83       CMP #$83
FC32  D0 13       BNE  $FC47
FC34  8A          TXA
FC35  49 FF       EOR #$FF
FC37  8D 6D 02    STA  $026D
FC3A  D0 05       BNE  $FC41
FC3C  98          TYA
FC3D  09 04       ORA #$04
FC3F  D0 03       BNE  $FC44
FC41  98          TYA
FC42  29 FB       AND #$FB
FC44  A8          TAY
FC45  B0 26       BCS  $FC6D
FC47  8A          TXA
FC48  D0 3D       BNE  $FC87
FC4A  AD 09 D2    LDA  $D209
FC4D  AA          TAX
FC4E  C9 9F       CMP #$9F
FC50  D0 0A       BNE  $FC5C
FC52  AD FF 02    LDA  $02FF
FC55  49 FF       EOR #$FF
FC57  8D FF 02    STA  $02FF
FC5A  B0 11       BCS  $FC6D
FC5C  29 3F       AND #$3F
FC5E  C9 11       CMP #$11
FC60  D0 2E       BNE  $FC90
FC62  8E DC 02    STX  $02DC
FC65  F0 06       BEQ  $FC6D
FC67  8E FC 02    STX  $02FC
FC6A  8E F2 02    STX  $02F2
FC6D  A9 03       LDA #$03
FC6F  8D F1 02    STA  $02F1
FC72  A9 00       LDA #$00
FC74  85 4D       STA  $4D
FC76  AD D9 02    LDA  $02D9
FC79  8D 2B 02    STA  $022B
FC7C  AD 2F 02    LDA  $022F
FC7F  D0 06       BNE  $FC87
FC81  AD DD 02    LDA  $02DD
FC84  8D 2F 02    STA  $022F
FC87  8C 01 D3    STY  $D301
FC8A  68          PLA
FC8B  A8          TAY
FC8C  68          PLA
FC8D  AA          TAX
FC8E  68          PLA
FC8F  40          RTI
FC90  E0 84       CPX #$84
FC92  F0 21       BEQ  $FCB5
FC94  E0 94       CPX #$94
FC96  D0 CF       BNE  $FC67
FC98  AD F4 02    LDA  $02F4
FC9B  AE 6B 02    LDX  $026B
FC9E  8D 6B 02    STA  $026B
FCA1  8E F4 02    STX  $02F4
FCA4  E0 CC       CPX #$CC
FCA6  F0 06       BEQ  $FCAE
FCA8  98          TYA
FCA9  09 08       ORA #$08
FCAB  A8          TAY
FCAC  D0 BF       BNE  $FC6D
FCAE  98          TYA
```

```
FCAF   29 F7      AND #$F7
FCB1   A8         TAY
FCB2   4C 6D FC   JMP  $FC6D
FCB5   AD 2F 02   LDA  $022F
FCB8   F0 CD      BEQ  $FC87
FCBA   8D DD 02   STA  $02DD
FCBD   A9 00      LDA #$00
FCBF   8D 2F 02   STA  $022F
FCC2   F0 C3      BEQ  $FC87
FCC4   48         PHA
(C) FINE-SCROLL DLI
FCC5   AD C6 02   LDA  $02C6
FCC8   4D 4F 00   EOR  $004F
FCCB   2D 4E 00   AND  $004E
FCCE   8D 0A D4   STA  $D40A
FCD1   8D 17 D0   STA  $D017
FCD4   68         PLA
FCD5   40         RTI
FCD6   00
FCD7   00
FCD8   4C 83 F9   JMP  $F983
(C) CASSETTE INITIALIZE
FCDB   A9 CC      LDA #$CC
FCDD   8D EE 02   STA  $02EE
FCE0   A9 05      LDA #$05
FCE2   8D EF 02   STA  $02EF
FCE5   60         RTS
FCE6   A5 2B      LDA  $2B
FCE8   85 3E      STA  $3E
FCEA   A5 2A      LDA  $2A
FCEC   29 0C      AND #$0C
FCEE   C9 04      CMP #$04
FCF0   F0 05      BEQ  $FCF7
FCF2   C9 08      CMP #$08
FCF4   F0 3E      BEQ  $FD34
FCF6   60         RTS
FCF7   A9 00      LDA #$00
FCF9   8D 89 02   STA  $0289
FCFC   85 3F      STA  $3F
FCFE   A9 01      LDA #$01
FD00   20 FC FD   JSR  $FDFC
FD03   30 29      BMI  $FD2E
FD05   A9 34      LDA #$34
FD07   8D 02 D3   STA  $D302
FD0A   A6 62      LDX  $62
FD0C   BC 93 FE   LDY  $FE93,X
FD0F   BD 91 FE   LDA  $FE91,X
FD12   AA         TAX
FD13   A9 03      LDA #$03
FD15   8D 2A 02   STA  $022A
FD18   20 5C E4   JSR  $E45C
FD1B   AD 2A 02   LDA  $022A
FD1E   D0 FB      BNE  $FD1B
FD20   A9 80      LDA #$80
FD22   85 3D      STA  $3D
FD24   8D 8A 02   STA  $028A
FD27   4C 77 FD   JMP  $FD77
FD2A   A0 80      LDY #$80
FD2C   C6 11      DEC  $11
FD2E   A9 00      LDA #$00
FD30   8D 89 02   STA  $0289
FD33   60         RTS
FD34   A9 80      LDA #$80
FD36   8D 89 02   STA  $0289
FD39   A9 02      LDA #$02
FD3B   20 FC FD   JSR  $FDFC
FD3E   30 EE      BMI  $FD2E
FD40   A9 CC      LDA #$CC
FD42   8D 04 D2   STA  $D204
FD45   A9 05      LDA #$05
FD47   8D 06 D2   STA  $D206
FD4A   A9 60      LDA #$60
FD4C   8D 00 03   STA  $0300
FD4F   20 68 E4   JSR  $E468
FD52   A9 34      LDA #$34
FD54   8D 02 D3   STA  $D302
FD57   A6 62      LDX  $62
FD59   BC 8F FE   LDY  $FE8F,X
FD5C   BD 8D FE   LDA  $FE8D,X
FD5F   AA         TAX
FD60   A9 03      LDA #$03
FD62   20 5C E4   JSR  $E45C
FD65   A9 FF      LDA #$FF
FD67   8D 2A 02   STA  $022A
FD6A   A5 11      LDA  $11
FD6C   F0 BC      BEQ  $FD2A
FD6E   AD 2A 02   LDA  $022A
FD71   D0 F7      BNE  $FD6A
FD73   A9 00      LDA #$00
FD75   85 3D      STA  $3D
FD77   A0 01      LDY #$01
FD79   60         RTS
FD7A   A5 3F      LDA  $3F
FD7C   30 33      BMI  $FDB1
FD7E   A6 3D      LDX  $3D
FD80   EC 8A 02   CPX  $028A
FD83   F0 08      BEQ  $FD8D
FD85   BD 00 04   LDA  $0400,X
FD88   E6 3D      INC  $3D
FD8A   A0 01      LDY #$01
FD8C   60         RTS
FD8D   A9 52      LDA #$52
FD8F   20 3F FE   JSR  $FE3F
FD92   98         TYA
FD93   30 F7      BMI  $FD8C
FD95   A9 00      LDA #$00
FD97   85 3D      STA  $3D
FD99   A2 80      LDX #$80
FD9B   AD FF 03   LDA  $03FF
FD9E   C9 FE      CMP #$FE
```

```
FDA0   F0 0D       BEQ  $FDAF       FE13   8D 1F D0    STA  $D01F
FDA2   C9 FA       CMP #$FA         FE16   A0 F0       LDY #$F0
FDA4   D0 03       BNE  $FDA9       FE18   88          DEY
FDA6   AE 7F 04    LDX  $047F       FE19   D0 FD       BNE  $FE18
FDA9   8E 8A 02    STX  $028A       FE1B   E4 14       CPX  $14
FDAC   4C 7A FD    JMP  $FD7A       FE1D   D0 E8       BNE  $FE07
FDAF   C6 3F       DEC  $3F         FE1F   C6 40       DEC  $40
FDB1   A0 88       LDY #$88         FE21   F0 0E       BEQ  $FE31
FDB3   60          RTS              FE23   8A          TXA
FDB4   A6 3D       LDX  $3D         FE24   18          CLC
FDB6   9D 00 04    STA  $0400,X     FE25   A6 62       LDX  $62
FDB9   E6 3D       INC  $3D         FE27   7D 97 FE    ADC  $FE97,X
FDBB   A0 01       LDY #$01         FE2A   AA          TAX
FDBD   E0 7F       CPX #$7F         FE2B   E4 14       CPX  $14
FDBF   F0 01       BEQ  $FDC2       FE2D   D0 FC       BNE  $FE2B
FDC1   60          RTS              FE2F   F0 CD       BEQ  $FDFE
FDC2   A9 FC       LDA #$FC         FE31   20 36 FE    JSR  $FE36
FDC4   20 7C FE    JSR  $FE7C       FE34   98          TYA
FDC7   A9 00       LDA #$00         FE35   60          RTS
FDC9   85 3D       STA  $3D         FE36   AD 25 E4    LDA  $E425
FDCB   60          RTS              FE39   48          PHA
FDCC   A0 01       LDY #$01         FE3A   AD 24 E4    LDA  $E424
FDCE   60          RTS              FE3D   48          PHA
FDCF   AD 89 02    LDA  $0289       FE3E   60          RTS
FDD2   30 08       BMI  $FDDC       FE3F   8D 02 03    STA  $0302
FDD4   A0 01       LDY #$01         FE42   A9 00       LDA #$00
FDD6   A9 3C       LDA #$3C         FE44   8D 09 03    STA  $0309
FDD8   8D 02 D3    STA  $D302       FE47   A9 83       LDA #$83
FDDB   60          RTS              FE49   8D 08 03    STA  $0308
FDDC   A6 3D       LDX  $3D         FE4C   A9 03       LDA #$03
FDDE   F0 0A       BEQ  $FDEA       FE4E   8D 05 03    STA  $0305
FDE0   8E 7F 04    STX  $047F       FE51   A9 FD       LDA #$FD
FDE3   A9 FA       LDA #$FA         FE53   8D 04 03    STA  $0304
FDE5   20 7C FE    JSR  $FE7C       FE56   A9 60       LDA #$60
FDE8   30 EC       BMI  $FDD6       FE58   8D 00 03    STA  $0300
FDEA   A2 7F       LDX #$7F         FE5B   A9 00       LDA #$00
FDEC   A9 00       LDA #$00         FE5D   8D 01 03    STA  $0301
FDEE   9D 00 04    STA  $0400,X     FE60   A9 23       LDA #$23
FDF1   CA          DEX              FE62   8D 06 03    STA  $0306
FDF2   10 FA       BPL  $FDEE       FE65   AD 02 03    LDA  $0302
FDF4   A9 FE       LDA #$FE         FE68   A0 40       LDY #$40
FDF6   20 7C FE    JSR  $FE7C       FE6A   C9 52       CMP #$52
FDF9   4C D6 FD    JMP  $FDD6       FE6C   F0 02       BEQ  $FE70
FDFC   85 40       STA  $40         FE6E   A0 80       LDY #$80
FDFE   A5 14       LDA  $14         FE70   8C 03 03    STY  $0303
FE00   18          CLC              FE73   A5 3E       LDA  $3E
FE01   A6 62       LDX  $62         FE75   8D 0B 03    STA  $030B
FE03   7D 95 FE    ADC  $FE95,X     FE78   20 59 E4    JSR  $E459
FE06   AA          TAX              FE7B   60          RTS
FE07   A9 FF       LDA #$FF         FE7C   8D FF 03    STA  $03FF
FE09   8D 1F D0    STA  $D01F       FE7F   A9 55       LDA #$55
FE0C   A9 00       LDA #$00         FE81   8D FD 03    STA  $03FD
FE0E   A0 F0       LDY #$F0         FE84   8D FE 03    STA  $03FE
FE10   88          DEY              FE87   A9 57       LDA #$57
FE11   D0 FD       BNE  $FE10       FE89   20 3F FE    JSR  $FE3F
```

```
FE8C   60               RTS

FE8D   04 03 80 C0 02 01 40 E0
FE95   1E 19 0A 08

FE99   A9 1E            LDA #$1E
(C) PRINTER INITIALIZE
FE9B   8D 14 03         STA  $0314
FE9E   60               RTS

FE9F   EA 02 C0 03

FEA3   A9 04            LDA #$04
FEA5   8D DF 02         STA  $02DF
FEA8   AE 9F FE         LDX  $FE9F
FEAB   AC A0 FE         LDY  $FEA0
FEAE   A9 53            LDA #$53
FEB0   8D 02 03         STA  $0302
FEB3   8D 0A 03         STA  $030A
FEB6   20 14 FF         JSR  $FF14
FEB9   20 59 E4         JSR  $E459
FEBC   30 03            BMI  $FEC1
FEBE   20 44 FF         JSR  $FF44
FEC1   60               RTS
FEC2   20 A3 FE         JSR  $FEA3
(C) PRINTER OPEN
FEC5   A9 00            LDA #$00
FEC7   8D DE 02         STA  $02DE
FECA   60               RTS
FECB   48               PHA
(C) PRINTER PUT BYTE
FECC   BD 41 03         LDA  $0341,X
FECF   85 21            STA  $21
FED1   20 4B FF         JSR  $FF4B
FED4   AE DE 02         LDX  $02DE
FED7   68               PLA
FED8   9D C0 03         STA  $03C0,X
FEDB   E8               INX
FEDC   EC DF 02         CPX  $02DF
FEDF   F0 15            BEQ  $FEF6
FEE1   8E DE 02         STX  $02DE
FEE4   C9 9B            CMP #$9B
FEE6   F0 03            BEQ  $FEEB
FEE8   A0 01            LDY #$01
FEEA   60               RTS
FEEB   A9 20            LDA #$20
(C) SPACE PRINTER BUFFER
FEED   9D C0 03         STA  $03C0,X
FEF0   E8               INX
FEF1   EC DF 02         CPX  $02DF
FEF4   D0 F7            BNE  $FEED
FEF6   A9 00            LDA #$00
(C) PRINTER PUT
FEF8   8D DE 02         STA  $02DE
FEFB   AE A1 FE         LDX  $FEA1
FEFE   AC A2 FE         LDY  $FEA2
FF01   20 14 FF         JSR  $FF14
FF04   4C 59 E4         JMP  $E459
FF07   20 4B FF         JSR  $FF4B
(C) PRINTER CLOSE
FF0A   A9 9B            LDA #$9B
FF0C   AE DE 02         LDX  $02DE
FF0F   D0 DC            BNE  $FEED
FF11   A0 01            LDY #$01
FF13   60               RTS
FF14   8E 04 03         STX  $0304
(C) SETUP PRINTER DCB
FF17   8C 05 03         STY  $0305
FF1A   A9 40            LDA #$40
FF1C   8D 00 03         STA  $0300
FF1F   A5 21            LDA  $21
FF21   8D 01 03         STA  $0301
FF24   A9 80            LDA #$80
FF26   AE 02 03         LDX  $0302
FF29   E0 53            CPX #$53
FF2B   D0 02            BNE  $FF2F
FF2D   A9 40            LDA #$40
FF2F   8D 03 03         STA  $0303
FF32   AD DF 02         LDA  $02DF
FF35   8D 08 03         STA  $0308
FF38   A9 00            LDA #$00
FF3A   8D 09 03         STA  $0309
FF3D   AD 14 03         LDA  $0314
FF40   8D 06 03         STA  $0306
FF43   60               RTS
FF44   AD EC 02         LDA  $02EC
(C) PRINTER STATUS TIMEOUT
FF47   8D 14 03         STA  $0314
FF4A   60               RTS
FF4B   A0 57            LDY #$57
(C) PRINT MODE
FF4D   A5 2B            LDA  $2B
FF4F   C9 4E            CMP #$4E
FF51   D0 04            BNE  $FF57
FF53   A2 28            LDX #$28
FF55   D0 0E            BNE  $FF65
FF57   C9 44            CMP #$44
FF59   D0 04            BNE  $FF5F
FF5B   A2 14            LDX #$14
FF5D   D0 06            BNE  $FF65
FF5F   C9 53            CMP #$53
FF61   D0 0C            BNE  $FF6F
FF63   A2 1D            LDX #$1D
FF65   8E DF 02         STX  $02DF
FF68   8C 02 03         STY  $0302
FF6B   8D 0A 03         STA  $030A
FF6E   60               RTS
FF6F   A9 4E            LDA #$4E
FF71   D0 DC            BNE  $FF4F
```

APPENDIX E6:

```
FF73   A2 00          LDX #$00
(C) 1ST CHECKSUM VERIFY
FF75   86 8B          STX  $8B
FF77   86 8C          STX  $8C
FF79   20 A9 FF       JSR  $FFA9
FF7C   E0 0C          CPX #$0C
FF7E   D0 F9          BNE  $FF79
FF80   AD 00 C0       LDA  $C000
FF83   AE 01 C0       LDX  $C001
FF86   C5 8B          CMP  $8B
FF88   D0 06          BNE  $FF90
FF8A   E4 8C          CPX  $8C
FF8C   D0 02          BNE  $FF90
FF8E   18             CLC
FF8F   60             RTS
FF90   38             SEC
FF91   60             RTS
FF92   A2 00          LDX #$00
(C) 2ND CHECKSUM VERIFY
FF94   86 8B          STX  $8B
FF96   86 8C          STX  $8C
FF98   A2 0C          LDX #$0C
FF9A   20 A9 FF       JSR  $FFA9
FF9D   20 A9 FF       JSR  $FFA9
FFA0   AD F8 FF       LDA  $FFF8
FFA3   AE F9 FF       LDX  $FFF9
FFA6   4C 86 FF       JMP  $FF86
FFA9   A0 00          LDY #$00
FFAB   BD D7 FF       LDA  $FFD7,X
FFAE   99 9E 00       STA  $009E,Y
FFB1   E8             INX
FFB2   C8             INY
FFB3   C0 04          CPY #$04
FFB5   D0 F4          BNE  $FFAB
FFB7   A0 00          LDY #$00
FFB9   18             CLC
FFBA   B1 9E          LDA ($9E),Y
FFBC   65 8B          ADC  $8B
FFBE   85 8B          STA  $8B
FFC0   90 02          BCC  $FFC4
FFC2   E6 8C          INC  $8C
FFC4   E6 9E          INC  $9E
FFC6   D0 02          BNE  $FFCA
FFC8   E6 9F          INC  $9F
FFCA   A5 9E          LDA  $9E
FFCC   C5 A0          CMP  $A0
FFCE   D0 E9          BNE  $FFB9
FFD0   A5 9F          LDA  $9F
FFD2   C5 A1          CMP  $A1
FFD4   D0 E3          BNE  $FFB9
FFD6   60             RTS

FFD7   02 C0 00 D0 00 50 00 58
FFDF   00 D8 00 E0 00 E0 F8 FF
FFE7   FA FF 00 00 00 00 00

FFEE   10 05 83 02 42 42 00 00
(C) CHECKSUM & ID
FFE6   01 02 8C 6C

FFFA   18 C0
(C) NMI   VECTOR
FFFC   AA C2
(C) RESET    "
FFFE   2C C0
(C) IRQ      "
```

APPENDIX E6:

Well, that's the 14K Operating System Source listing of the Atari XL/XE. Of course, there's no reason why you couldn't change and improve the OS now, I'll leave it in your competent hands.

Before I finish this Appendix, you might be pleased to find out that you could (if wanted), turn your Atari into a different 8-bit machine such as a ZX81, BBC, Vic 20, Oric, Spectrum, Dragon or Commodore 64.

What would I want to do this for, I hear you say? Well, it does offer some potentials!

You see, if all those unreleased games won't come to the Atari, then why not take the Atari to the games!? By re-writing the entire OS we can achieve just this. The Commodore 64, Vic 20 and BBC are the easier systems to imitate, because they use the 6502 CPU. I'm not sure about Dragon and Oric, but the ZX81 and Spectrum use the Z80 CPU. ZX81 conversion should be easy, but Spectrum conversion does bring difficulty because it uses the Z80' faster processing power to graphics advantage, by achieving up to 8 colours horizontally on an equivalent of Atari' Graphics 8 resolution.

I think it would be good if a group of people could get together on this subject to create the necessary OS and hardware porting equipment, and should any capable person be taking this seriously, then get in touch with me.

APPENDIX F

APPENDIX F1:

THE HARDWARE CHIPS.

Inside your trustworthy Atari classic there is quite a lot of power, by power I mean that it is capable of achieving excellent results in a wide and varied field of subjects. Whether you are word-processing, programming, on the BBS or utilizing the computer for a specific subject, the Atari classic is without hype, a very affective tool.

This power is all available due to the Hardware chips installed underneath the shell. In our Atari, there is the 6502 Central Processing Unit (CPU), 4 I/O chips, the Operating System (OS) ROM, expandable RAM and several MSI (Medium Scale Integration) chips for address decoding and databus buffering.

The CPU isn't the best of its kind, far from it. It wasn't bad in its day. Nowadays, on its own it isn't a scratch on latest RISK processors, but when it's used in conjunction with the special 4 I/O chips in the Atari, the odds differ. The OS is 14K of controlling program which basically converts the computer from a machine into a home-computer. It's the permanently residing program that helps interface the user with his hardware and software. RAM expansion in the XL is usually 64K (64*1024 bytes) which generally means that you could have an average of 1092 names and phone-numbers stored in your computer at once. The 130XE offers 128K, twice as much. In addition to this, you can have memory expansion. The amount to which you can expand is really determinate on the Electronics brain behind the creation. To date, a 1Meg (1 Million bytes) expansion is possible. Of course, in addition to onboard RAM, you have floppy disks, elephant disks and even hard-disk storage which can keep tremendous amounts of stored RAM, accessible within seconds.

The 4 I/O chips have been named; ANTIC, GTIA, POKEY and PIA. These are the main chips responsible for interfacing the computer itself to any device connected to any of the ports, which includes the creation of sound and vision.

All of these 4 I/O chips are what is known as Large Scale Intergration (LSI) and they occupy the memory range 53248 - 55295; $D000 - $D7FF. Overleaf is a short description about which chip does what.

APPENDIX F1:

CHIP	ADDR.	FUNCTIONS
ANTIC	$D400	DMA (Direct Memory Access) NMI (Non-Maskable Interrupts) Vertical & Horizontal fine-scrolling Light-Pen position registers TV Vertical line counter WSYNC (Wait for horizontal SYNC)
GTIA	$D000	Playfield priority control Colour and Luminance imaging Player/Missile Graphics (PMGs) Graphics registers Size control Horizontal position Playfield collision detection Switches and triggers (misc. I/O functions)
POKEY	$D200	Keyboard scan and control Serial communication port (bi-directional) POT scan (4 POT digitization) Audio generation (4 channels) System Timers IRQ (maskable interrupt requests) to/from peripherals Random number generation
PIA	$D300	Controller jacks (joysticks) I/O Peripheral control and interrupt lines IRQ (maskable) control to/from peripherals

You may see that the ANTIC chip is majoritly in control of graphics, although GTIA actually interfaces with the TV. POKEY is mainly for sound, although along with PIA, they control IRQ's which are for device control. GTIA also supports PMG's, which are the proto version of hardware sprites/bobs. NMI's and DMA are mainly concerned with the screen display. Although NMI's are non-maskable, meaning that they can't be disabled, in the Atari - they can! At least 2 out of 3. The reason behind this is that they are truly non-maskable to the CPU, but ANTIC is a very special Atari-only chip which is a CPU in its own right. It has its own instruction-set and it is this chip that masks the NMI's.

Anyway, without any more information to supply on the interior of the Atari, I think that's it! The only thing I haven't covered is the question of the GTIA. Is it really Georges Television Interface Adapter?

APPENDIX F2:

1050 SPECIFICATIONS:

The 1050 drive uses the 6507 Microprocessor, formats in either of single or dual densities, uses 5 1/4" disks each of which can hold a maximum of 260K uncondensed. Operating temperature is between 75.6-129.6F within the altitute 0-9842.5 feet. Besides, who on earth would take it up that high! Transfer rate is 19200 BAUD.

DENSITY-

DUAL:	SINGLE:	RAMDISK:	DESCRIPTION:
40	40	n/a	Tracks
26	18	n/a	Sectors/track
1040	720	512	Total sectors
1023	719	511	Sectors available to DOS
13	12	12	DOS overhead, sectors
1010	707	499	Sectors to user
128	128	128	Bytes/sector
3	3	3	Overhead bytes/sector

If you have a US-Doubler or similar chip fitted in your drive, then it offers true double-density, giving you 256 bytes per sector with a transfer rate of 70000 BAUD.
A good feature when using a double-density DOS, such as SUPERDOS with the US-Doubler fitted, is the high 128 bytes of each of the directory sectors 361 - 368. As they are not utilized, you can use them to give your files a clearer name. One such program that does do this is a program utility called PICODOS. It reads all the .COM files on a disk, and allows you to give a better description which is then displayed on the screen when booting the disk. If you allow 40 bytes per filename for its 'better name', then there is room for 25 new-names. But, whether it is likely that you'll fit 25 .COM files on 1 side of a disk is another story. Every file would have to be 40 or less (d/density) sectors long.

Protection of your disks is sometimes quite a tedious job. Normally, they should be kept in their sleeves away from dust and sunlight when unused. Bending should be avoided, they should be inserted in the drive the correct way only, no prodding the exposed oxide surfaces. Temperature ranges between 10-52C, and magnetic areas like TVs should be avoided. I have broken all these laws and disks have still worked! I leave them all over the place, including on top of the TV. I've put them in the drive the wrong way, exposed them to sunlight, dropped them in the rain carrying them up and down from friends houses. This last point does tend to do the worst damage. You need to give it about 2-3 days to dry off at room temperature. Don't put it near a Warmth Output System WOS (a fire), or the format warps and NO drive for ANY computer will understand which system the disk was FORMATted with!

APPENDIX F3:

PINOUTS.

Here's diagrams and reference information to all of the Atari' I/O jacks.

The games controller jack:

```
*****************************
 *                         *
  *  1   2   3   4   5   *
   *                   *
    *  6   7   8   9  *
     *              *
      *****************
```

```
PIN   FUNCTION

1     Joystick Forward
2        "      Back
3        "      Left
4        "      Right
5     B Potentionmeter (Paddle-1/3)
6     Joystick Trigger
7     +5Volts Output
8     Ground
9     A Potentionmeter (Paddle-0/2)
```

Monitor jack:

```
      3  *  1


    5        4

        2
```

```
PIN   FUNCTION

1     Composite Luminance
2     Ground
3     Audio Output
4     Composite Video
5     Composite Chroma
```

APPENDIX F3:

Serial I/O jack:

```
         **************************
        *                          *
       *   2   4   6   8   10   12  *
      *                              *
     *   1   3   5   7   9    11   13 *
    *                                  *
    ***************************************
```

PIN	FUNCTION
1	Clock Input
2	" Output
3	Data Input
4	Ground
5	Data Output
6	Ground
7	Command
8	Motor Control
9	Proceed
10	+5V/Ready
11	Audio Input
12	+12V
13	Interrupt

The Enhanced Cartridge Interface:

The Parallel Bus is on the next leaf, however, in the 130XE it is called the ECI. The only difference is the 14-pin addition explained here:

```
*******************
*                 *
*  A B C D E F G  *
*                 *
*  1 2 3 4 5 6 7  *
*                 *
*******************
```

PIN	FUNCTION		PIN	FUNCTION	
A	Reserved		1	External Device select?	EXSEL
B	Inter. Req. IRQ		2	System RESET	RST
C	Antic Halt sig.		3	Chip Select at D1xx	
D	Address Line	A13	4	Math Pack disable	MPD
E	Address Line	A14	5	External Audio Input	AUDIO
F	Address Line	A15	6	Present Cycle is Refresh	REF
H	Ground	GND	7	Second dc supply +5Volts	+5V

APPENDIX F3:

The ROM Cartridge Interface:

```
**************************************************
*                                                *
*  A  B  C  D  E  F  H  J  K  L  M  N  P  R  S   *
*                                                *
*  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15   *
*                                                *
**************************************************
```

Looking at the Cartridge slot from the reverse of your machine, the setup is above:

PIN	FUNCTION		PIN	FUNCTION	
A	ROM Present	RD4	1	Chip# \$8000-\$9FFF	S4
B	Ground	GND	2	Address Line	A3
C	Address Line	A4	3	Address Line	A2
D	Address Line	A5	4	Address Line	A1
E	Address Line	A6	5	Address Line	A0
F	Address Line	A7	6	Data Line	D4
H	Address Line	A8	7	Data Line	D5
J	Address Line	A9	8	Data Line	D2
K	Address Line	A12	9	Data Line	D1
L	Data Line	D3	10	Data Line	D0
M	Data Line	D7	11	Data Line	D6
N	Address Line	A11	12	Chip# \$A000-\$BFFF	S5
P	Address Line	A10	13	+5Volts DC	+5V
R	CPU Read/Write	R/W	14	ROM Present	RD5
S	System Clock	B2	15	ROM Bank Ctrl Sel.	CCTL

Well, that's it. The Atari 65XE doesn't have the PBI or ECI, so don't go buying one if you want this port!

APPENDIX F3:

The Parallel Bus Interface (PBI)

```
*****************************************************************************
*                                                                           *
*  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49  *
*                                                                           *
*                                                                           *
*  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50  *
*                                                                           *
*****************************************************************************
```

TOP		PIN	PIN	BOTTOM
Ground	GND	1	2	External Select
Address O/P	A0	3	4	A1
	A2	5	6	A3
	A4	7	8	A5
	A6	9	10	GND
	A7	11	12	A8
	A9	13	14	A10
	A11	15	16	A12
	A13	17	18	A14
	GND	19	20	A15
Data Lines	D0	21	22	D1
(Bi-Directional)	D2	23	24	D3
	D4	25	26	D5
	D6	27	28	D7
	GND	29	30	GND
Phase-2 CLK O/P		31	32	GND
Reserved	NC	33	34	RESET Output
Interr. Request	IRQ	35	36	Ready Input
	NC	37	38	External decoder Output
	NC	39	40	Refresh Output
Column Address O/P		41	42	GND
Mathpack disable I/P		43	44	Row Address strobe
	GND	45	46	Latch Read/Write out
(+5V dc?)	NC	47	48	NC (+5V dc?)
Audio Input		49	50	GND

APPENDIX F4:

PORT INPUT.

The voltages in the ports of the Atari classic ain't that significant, like most DC applications really. The joystick ports give +5 Volts on pin-7. The amperage is insignificant at around 50mAmps maximum. The potentionmeter pins 5 and 9 return a value of 228 for a full 5Volts on the line, while the lowest value of 0 is returned for the trigger voltage (almost 1Volt) being on the line (or is it the other way around?). It is actually possible to use these potentionmeter pins to input voice tracks from music tapes into the computer. You'll need to create your own lead and software. I'm not actually sure how to go about it as I haven't really looked into it. I don't even know how (if) any commercial packages do it this way, such as the REPLAY sound sampling system because I don't possess any. What I do know off hand, is that you'll need to either set bit-2 of SKCTL location 53775; $D20F or start and restart the pot scan via location $D20B. The pot scan should be read every 2-scan lines in fast mode, or prior to it being restarted. The lead would have to take about 1Volt plus the speaker voltage (music sound) into the potentionmeter port with a feed off back to the music system. Please don't take the facts I've given as accurate or even correct, because I'm really unsure. So don't go damaging things accidentally, this is merely my assumption, although I know that it is possible to put samples into the Atari this way. Anyway, pin-8 of the port is ground.

The Serial I/O port passes +5Volts dc/ready on pin-10 and +12Volts dc on pin-12. Ground lines are pins 4 and 6. Music can also be input from the cassette via pin-11. The Atari already offers an IRQ to obtain the bits from bit-4 of SKSTAT location 53775; $D20F, which are collected and placed in SERIN location 53773; $D20D. I did once try to input music from the cassette and play it back, but without success unfortunatly. The music I recorded turned out to be the background "noise", so I 'taped' the wrong track (the data-in) where I should have recorded the audio-in, I never had any success finding the audio-in bit, so I quit. Perhaps someone does know more on this subject. If so, it could be a good appendix addition to this book!?

On the next leaf is all the information I can supply:

POTGO location 53771; $D20B is POKEd to begin the POT scan sequence, the POT values should be read firstly. The write strobe then causes the following steps:

APPENDIX F4:

1. The Scan counter is cleared to 0,
2. The Capacitor dump transistors are turned off,
3. The Scan counter begins counting,
4. The counter value is captured in each of 4 POT (NOT 8) registers as each POT line crosses trigger voltage,
5. When the counter reaches 228, the capacitor dump transistors are switched on.

The PIA (6520/6820) gives TTL levels, 1 load for both input and output. The circuits are as follows:

PORT A:

```
------------¦
6520        ¦               ¦
           A¦----220R-------¦ Jack-0
        Port¦          ¦    ¦
------------¦         ---
                      ---
                       ¦  .001F
```

PORT-B:

```
------------¦     o+5
6520        ¦      4.7K     ¦
           B¦----220R-------¦ Jack-1
        Port¦          ¦    ¦
------------¦         ---
                      ---
                       ¦  .001F
```

Here's the "Trigger" Port circuit:

```
------------¦
GTIA Trig.  ¦               ¦
            ¦----220R-------¦ Jack
            ¦          ¦    ¦
------------¦         ---
                      ---
                       ¦  .001F
```

The parallel bus, the enhanced cartridge interface and indeed the above mentioned ports are fully described in the PINOUTS appendix.

APPENDIX G

APPENDIX G1:

OTHER PROGRAMS.

Obviously, the Atari' memory as it stands is a very large subject, but what's in it and where, when various programs are loaded. This subject in itself can comprise of many books as you can imagine. In addition to this, what uses can you put particular programs too? Anyway, without further ado, here's a few words about a couple of subjects.

A very quick tip is for the BOULDERDASH construction kit. Some of you will have noticed that there have been many screens designed for this game, and in particular, some of these caves and intermissions are 'different' in ways such as pertaining more than 1 exit etc.. Well, you too can achieve simple tasks such as this, the general idea is to design a cave so that you can just put 1 object inside it. You then load this cave file into a word-processor such as SPEEDSCRIPT or TEXTPRO. Hmmm. All the characters look the same apart from the 1 (the object you put in the screen). Hmmm, I wonder? I'll leave you to ponder over that one. There are a lot more tips on this one, a whole lot more!

If you file your program collection on MJ Hughes DATABASE program, and you posess either a Basic program called QUICK-VIEW Creator by someone called KRACTWERK, or 2 other programs called TOMO #2: MJDB VIEW and MJDB converter, then you can convert your program data to any of 2 types of quick reference bootable files. The 2 TOMO programs are a little harder to get hold of, but they can create a better view-file.

Bootable menu's including Multi-Boot XL and HOWFEN are very handy. If you want to write your own programs to convert from 1 menu to another, then where is the information you need? The only real way of obtaining the information is to HACK the menu etc., unless of course... Yeap! Unless of course I give you it.

The HOWFEN menu can hold a maximum of 20 files, the amount of files plus 1 presently on the disk is shown at byte 41 on sector 7. The name table and sector counts is in internal format beginning at byte 12 of sector 2. The screen is narrow width, so each file entry line is 32 spaces apart. The start sectors of each file begin at bytes 42 and 62 of sector 7 for LSBs and MSBs, respectively. The length of a file is found by subtracting the start sector of the file wanted with the start sector of the next file on the menu. If there isn't another file on the menu, then it doesn't matter, since the next free sector should be present which is the same principle.

Alike HOWFEN menu, Multi-Boot XL's start sectors are LSBs and MSBs and begin at bytes 10 and 20 of sector 48, respectively. This menu keeps file lengths beginning at byte 0 of the same sector. The name table begins at byte 30 of this same sector again. Since the file lengths are kept in single bytes, this menu can only retain files which have a maximum sector length of 255. There are several other menu's, but we won't cover them in this book.

Musical bars is a nice addition to programs, but how do you get the information that they need. If you own 1 of the serious music packages such as BlACK MAGIC COMPOSER (BMC), or the SOUNDMONITOR by Benjy, then you may be surprised to find that it is relatively easy to discover where in memory the programs keeps its frequency, distortion and volume controls before these values are loaded into the hardware channel registers. Benjy's SOUNDMONITOR is very explanatory, though, if you're unsure, all you have to do is to pack a music file and use the various Basic listings to start the music. Note, that just before you start the music, try changing the Display Lists DM pointer of the Graphics 0 screen. You can actually display the memory being altered and find out all the exact addresses by timing what you hear with what you see. Obviously, the more complex the tune the harder it is to compare, so try simple, slow tunes until you've found your bytes. BMC is based on this technique, although, you'll need to be familiar with machine-code to achieve any success. If you don't know M/C, then there is another method, but even this way, you'll need to possess a program called the FREEZER, and you need to get aquainted with the complexities of BMC and the Vertical Blank Interrupt.

Moving onto the subject of pictures, if you own the KOALA or ATARI touch tablet, then drawing pictures is made easier than having a joystick. A program called GRAPHIC ARTS DEPARTMENT has a good velocity mode for joystick users, but besides this, owning the touch tablet does make things easier. Your pictures probably come out best if you firstly draw them on paper, and then slip the paper under the plastic cover of the tablet. After tracing your own picture with the tablet pen, only a little touching up is needed for a picture that really shows your drawing ability. You can save these pictures to disk either using the SAVE option, or by pressing the 'greater-than' symbol on the 'insert' key. The first method saves in condensed mode, the latter in normal mode. If you wish to load the latter saved pictures, then use the 'less-than' symbol on the 'clear' key. The file-name of this file is always the same name, PICTURE, so be careful not to overwrite old ones - rename them.

Converting DOS saved pictures to your own machine-code bootable programs is another subject. There are many utilities in the public domain that do this, like a very good picture converter created by someone no longer on the 8-bit Atari (?) known as the MOCKINGBIRD. But, this is not the only use for such a utility. You can also convert ANY DOS file to boot-sectors so long as that file doesn't exceed 62 sectors. It's all experimentation, and here I leave you in ponderment for all those utilities - what else can they do!??

APPENDIX G2:

ATARI SUPPORT.

This is a list of addresses that I felt it necessary to include in this book. The addresses are mainly taken from the rear of Page-6 magazine, or otherwise known as NEW Atari User.

Software Infinity
642 East Waring Avenue
State College, PA 16801
Good PD selection and are now marketing commercial games overseas.

DataQue Software
PO Box 134
Ontario, OH 44862
Turbo-816 16-bit upgrade board. Transkey hardware for using IBM keyboards on the 8-bit, and more.

B&C ComputerVisions
2730 Scott Boulevard
Santa Clara, CA 95050
Tel. (408) 986 9960
Huge selection of software and hardware items. Also some commercial games unavailable elsewhere.

Sagamore Software
2104 Arapahoe Dr
Lafayette, IN 47905
Good PD/shareware selection with extensive documentation.

Change In Heat
12 Bella Vista Place
Iowa City, Iowa 52245
Independent programmer has produced 2 excellent commercial quality games for the 8-bit.

Bresnik Software
555 Ware Street
Mansfield, MA 02048
Another independent producing good educational software.

UltraBasic
10 East 10th Street
Bloomsburg
PA 17815
8-bit specialty software

NERDS Software
18 Wendy Drive
Farmingville, NY 11738
Printshop related software

IB Computers
9244 SW Beaverton Hills
Hwy, Valley Plaza
Shopping Centre
Beaverton, Oregon 97005
Tel. (503) 297 8425

BellCom
PO Box 1043
Peterborough, Ontario
Canada K9J7A5
The largest PD/shareware selection.

Compsult
PO Box 5160
San Luis Obispo
CA 93403-5160
Closeout items galore

No Frills software
800 East 23rd Street
Kearney, NE 68847
Closeouts & Printshop graphics.

Miles Better Software
219/221 Cannock Road
Chadsmoor, Cannock
Staffordshire WS11 2DD
Masses of Software for 8-bit and 16-bit.

Bacmun Software
1671 East 16th Street
Suite 629, Brooklyn
NY 11229
PD theme disks.

Alpha Systems
1012 Skyland Drive
Macedonia, OH 44056
Utility Software and
Hardware.

American Technavision
15338 Inverness Street
San Leandro, CA 94579
Tel. (510) 352 5639
Large selection of
commercial software at
closeout prices and
hardware replacements.

Best Electronics
2021 The Alameda,
Suite 290, San Jose
CA 95126
Tel. (408) 243 6950
Known as THE Atari
hardware store. If these
don't have the part you
need, nobody does.

BRE Software
Markets a new 8-bit game
352 West Bedford Avenue
Suite 104, Fresno
CA 93711
PD/Shareware.

C&T ComputerActive
PO Box 893
Clinton, OK 73601

Phantoms Atari 8-bit
Box 331, Levisa Road
Mouthcard KY 41548

Newell Industries
PO Box 253
Wylit, TX 75098
Tel. (214) 442 6612

Innovative Concepts
31172 Shawn Drive
Warren MI 48093
Tel. (313) 293 0730
Accessories, hardware
PD software.

CSS
PO Box 17660
Rochester NY 14617
Tel. (716) 429 5639
Specialty hardware
and 8-bit repairing.

San Jose Computers
640 Blossom Hill Road
San Jose CA 95123
Tel (408) 995 5080
New and reconditioned
hardware and software

East Hartford Computers
202 Robert Street
East Hartford CT 06108
Discontinued software
for all computers.

Aerion Software
PO Box 1222
Riverdale Station
NY 10471-1222

Toad Computers
556 Baltimore Annapolis Blvd
Severna Park, Maryland 21146
Tel. (301) 544 6943
Software and reconditioned
hardware.

Gralin International
11 Shilito Road
Poole, Dorset BH12 2BN
Hardware and Software
including ICD products

APPENDIX G2:

TWAUG
PO Box 8
Wallsend, Tyne'n'Wear
NE28 6DQ
Regular newsletter with disk, also hardware repairs.

NOSAUG
Stuart Murray
71 Walker Road
Torry, Aberdeen AB1 3DL
Regular Futura disk also on tape.

Atari Classics
170 Sproul Road/Rt.352
Frazer PA 19355-1958
A recently formed magazine by dedicated 8-bit users.

Current Notes
122 North Johnson Road
Sterling VA 22170
A top quality 8-bit and 16-bit magazine.

Micro Discount
265 Chester Road
Streetly, W. Midlands
Tel. (021) 353 5730
Large selection of commercial software, also hardware replacements.

Tiger Developments
26 Menziers Avenue
Walmer, Kent CT14 7QZ
Commercial 8-bit software

Atari Interface
3487 Braeburn Circle
Ann Arbor MI 48108
Tel. (313) 973 8825
8-bit and 16-bit magazine with disk, with input from groups all over.

NEW Atari User (Page-6)
PO Box 54
Stafford ST16 1DR
A professionally produced magazine for 8-bit and 16-bit with large PD on 8-bit and 16-bit, including commercial software.

ANG Software
Puttershoeks estraat 63a
3114 PK Schiedam
Holland
Tel. (0)10 4735987
Parts for the Pokey stereo upgrade as well as MegaMag, I believe.

There are still many more existing sources on the Atari 8-bit, of which seem a little less known. There is an excellent German games company called KE-Soft, as well as many other magazine and newsletters still alive. There is Moje Polish magazine, the New Aladdin, Phoenix (risen from the ashes), even a good quality free disk called The Grim Reaper by John E.

There is also a compilation of British demos which is in the making at the same time of this book. I haven't worked out what to do with this disk at the moment, but by the time that your reading this, it will have been sorted out. Perhaps, it may be obtainable through TWAUG, who knows as yet?

APPENDIX G3:

GLOSSARY:

ANTIC, GTIA, PIA, POKEY: Special Atari chips controlling the XL/XE's graphics, colour and screen resolution, controller jacks and sound, repectively. Located in ROM at 53248 - 54783. ANTIC also processes the NMI's and POKEY processes the IRQ's.

ATTRACT MODE: The feature included in the Operating System to protect the TV from burn-out.

BACKGROUND: The area of the screen for typing in Graphics 0, Memory display etc..

BCD: Binary Coded Decimal, see the LOGIC appendice for a full explanation.

BORDER: That area of the screen which surrounds the Background, normally black on Graphics 0.

BIT, BYTE: A BIT is the smallest size division of memory in the computer. It is so small that it can only retain a status of being off or on, 0 or 1, low or high. 4 BITS are a NYBBLE, while 8 BITS, or 2 NYBBLES, form 1 BYTE. Every memory location within the Atari XL/XE's are 1 BYTE in size, hence, the name: Atari 8-BIT computers. This means that every memory location can have a value within the range 0 - 255.

CHARACTER GRAPHICS: The technique of using redefined character sets, usually in Graphics 12 or 13 to create graphical display.

CHARACTER SET: The term used for a particular set of characters in a particular order. See location 756 and 57344.

CIO: Central Input/Output routines located in ROM. Controls the I/O Control Block (IOCB) operations. In brief, CIO handles the data I/O through the device driver/s (or device handlers), then passes control to those drivers. It's a single interface with which to access all peripherals in a device-independent manner (i.e. uniformed handling of data with no regard to which device is being accessed). As an example; writing data to a disk file is treated in an identical manner as writing data to the screen; commas insert blanks between elements and both commas and semi-colons surpress the End of Line (EOL) character.

COLDSTART: The term used which simply means to turn the computer off and on.

COLOUR CLOCK: The smallest unit of horizontal distance across a scan-line. See the TIMINGS appendix.

APPENDIX G3:

CTIA: The elder chip to the GTIA.

CYCLE STEALING: A process carried out by ANTIC in order to create the screen display.

DCB: Device Control Block, used by SIO.

DISPLAY LIST: This is the set of ANTIC instructions detailing the whereabouts of the screen memory and also in which way it is to be displayed.

DISPLAY LIST INTERRUPT: A DLI is, usually, a very short machine language routine that is executed during a Horizontal Blank on the TV frame.

DOS: Disk Operating System. The software loaded from disk file DOS.SYS that controls all disk I/O. If you are using DOS 1 or 3, then chuck it in the bin and get DOS 2.5.

DMA: Direct Memory Access. The process of the Antic chip in order to obtain data from memory without the use of the CPU.

DUP: Disk Utilities Package. The software loaded from the disk file DUP.SYS that handles all of the DOS menu functions.

EOL: An End Of Line character having the code $9B (The RETURN key).

FMS/DFMS: Disk File Management System portion of DOS; a dedicated device driver that controls all I/O operations for device "D:".

FONT: See CHARACTER SET.

FP: Floating Point mathematical package in ROM.

FUNCTION: A Basic instruction which returns a value back to the program.

HORIZONTAL BLANK: The time period from when the TV electron-beam is switched off at the right edge of the screen, to when it is switched back on at the left edge of the screen, 1 scan-line lower.

IMMEDIATE MODE: A Basic line input without the use of a line number.

I/O: Input/Output.

IOCB: Input/Output Control Block. Area of RAM (locations 832 - 959) used by CIO to define operations to devices such as the disk drive (D:), printer (P:), screen display (S:), screen editor (E:), keyboard (K:), cassette recorder (C:) and RS232 (R:). ZIOCB is the Page-0 IOCB.

IRQ: Interrupt ReQuest used for the serial port communication, peripheral devices, timing and keyboard input. IRQ's are processed by the POKEY chip.

LSB: The Lowest Significant Byte, or Bit. See the LSBs/MSBs appendice.

MODE LINE: A particular amount of scan-lines depending on the Graphics mode in use. Graphics mode 0 has 8 scan-lines per mode-line.

MSB: The Most Significant Byte, or Bit. See the LSBs/MSBs appendice.

NMI: Non-Maskable Interrupt; used for video display and Hard RESET. NMI's are processed by the ANTIC chip.

OS: Operating System. The resident system that runs the Atari. The OS is 14K and resides at 49152 - 53247 and 55296 - 65535.

PIA: The Peripheral Interface Adapter chip which interfaces the 6502 CPU with external devices. It also interfaces the joystick ports.

PIXEL: The smallest 2 dimensional unit of a Graphics mode. In Graphics 15, the pixel is 1:1; 1 colour clock in width and 1 scan-line in depth.

PMG, PM Graphics: Player/Missile Graphics. Players and Missiles are special moveable, user-defined, coloured screen objects otherwise known as Hardware sprites or bobs. They are often used for games, animation or various other special-FX. PMG's are also unique in that you can establish the manner (priority) in which they interact with the rest of the screen display as well as each other.

RAM: Random Access Memory. All memory from location 0 - 49151, which is used for storage, programs, buffers, cartridges, DOS, IOCB's, shadow registers and the registers for the special Atari chips. Random Access means you can get to and from these locations at random, and not that they store information randomly!

ROM: Read Only Memory. Locations 49152 - 65535 is the ROM. ROM is also used to describe cartridge memory which cannot be user altered, even the ROM Basic package which is switched in when enabled. You cannot alter ROM, except for various locations of the Hardware memory found in the D-block.

APPENDIX G3:

SCANLINE: A horizontal distance of 228 colour clocks. See the TIMINGS appendice.

SECTOR: This is a 128 byte area on a disk.

SHADOW REGISTERS: Used to monitor the contents of write-only hardware registers.

SIO: Serial Input/Output routines located in ROM. Controls serial operations including the 850 interface (R:) and cassette recorder (C:). Briefly, SIO controls the Atari's peripherals as per request placed in its DCB by the proper device driver. It is also accessed by FMS for data transfer.

TEXT WINDOW: This is the 4 lines of Graphics 0 which appear at the bottom of the screen after a call such as Graphics 1.

VERTICAL BLANK: This is the interval between the time the TV electron-beam turns off after reaching the bottom right corner of the screen and returning to the top left corner and turns back on again. See VBI. There are 2 VBLANK stages; stage-1 is every 50th of a second, while a stage-2 VBLANK can be any relation to a 50th of a second divisable by 2; ie. 25th of a second, 12\th of a second etc., depending when you set and clear CRITIC at location 66.

VBI: Vertical Blank Interrupt. A VBI is a machine-language program of limited 'time' that is processed during the Vertical Blank interval.

WARMSTART: The term which simply means to press the Reset key.

ZERO PAGE: This is memory in the range 0 - 255; $00 - $FF, which can be accessed by just an LSB.

APPENDIX G4:

PROGRAM LISTINGS.

This is the very last appendix of the book, and is also the very last one to be compiled. Here you'll find just a few useful listings and programming techniques you might not have seen before.

I'm going to kick off with a program to help your graphics angles. Triangles, squares, circles and even elipses are all user creatable of course, but how do you create some of these more complex ones? Well, with Turbo Basic, circles and simple elipses can be achieved with the CIRCLE command, but if you haven't got this Basic, why not try my first program:

```
10 GRAPHICS 15+16
20 COLOR 1
30 DEG
40 FOR I=0 TO 540 STEP 3
50 C=(COS(I-(I/3))*50)+80
60 S=(SIN(I+(I/3))*50)+96
70 PLOT C,S
80 NEXT I
90 GOTO 90
```

Not bad eh!? If you want the Basic circle, then remove the "-(I/3)" and "+(I/3)" strings in lines 50 and 60. The size of circle/eclipse is achieved with the value 50 on both the sine and cosine curves. Co-ordinate 80,96 is the dead centre of the curve. Now, if you're after more complex eclipses, or different shapes like octagons, then you'll need to fiddle around with the string arguments (explained above). These parameters are the secrets. Oh, you might find changing the DEG command to the RAD command quite interesting too!?

Keeping on the subject of circulism, the program on the next page stores the sine and cosine values of a circle into an array, which are then used as X,Y positions for text.

The text that is plotted is reversely typed into T$, but you'll have to have a copy of Turbo Basic (TB from now on) to run this program because I've used TBs TEXT command, which happens to be a very useful command. The purpose that I've put it to is just an example as you'll realize with the speed of Basic, but there's no reason why the program can't display the text in steps of 2 or more. You will have problems with a trail of bits being left behind, but it can be overcome. One method is by redefining the character-set!

APPENDIX G4:

```
100 GRAPHICS 8
110 POKE 710,0
120 DIM C(360),S(360),T$(30)
130 COLOR 1
140 DEG
150 FOR I=0 TO 360 STEP 1
160 C=(COS(I)*50)+80
170 S=(SIN(I)*50)+96
180 C(I)=C:S(I)=S
190 PLOT C,S
200 NEXT I
210 T$=" EREHT IH"
220 G=15:I=300
230 I=I+(I[360 AND J=1)-((I]359)*360)
240 J=J+(J[9)-((J]8)*8)
244 N=I+J*G:N=N-360*(N]360)
250 TEXT C(N),S(N),T$(J,J)
280 GOTO 230
```

Well, as you'll know if you've typed the listing in, in it's present form it is very slow, but I leave you to work something out with it.

In addition to the use of the program, you might not have come across lines like 230, 240 and 244. Believe it or not, lines 230 and 240 are a boolean style nested FOR/NEXT loop. I goes from 0 to 360, and J goes from 1 to 9. Both in steps of 1 (in this case). Line 244 shows variable N, which is used to extract the X and Y co-ordinates for each character in T$. For more information on this Boolean style programming, and its reasons, consult my relating appendice.

Again, here's a listing with some Boolean expressions, but this time they're used to guide a rolling square around the graphics mode 8 screen.

```
100 GRAPHICS 8:POKE 710,0
110 X=0:Y=0
120 I=I+(I[30)-((I=30)*29)
130 V=0:GOSUB 180:V=1:GOSUB 180
140 X=X+(X[100 AND Y=0)
142 Y=Y+(X=100)
144 X=X-(Y=100)
146 Y=Y-(Y]0 AND X=0)
160 GOTO 120
170 --
180 COLOR NOT V
190 PLOT X+30+V-I,Y
200 DRAWTO X,Y+I-V
210 DRAWTO X+I-V,Y+30
220 DRAWTO X+30,Y+30+V-I
230 DRAWTO X+30+V-I,Y
240 RETURN
```

APPENDIX G4:

Advanced programmers, and they know who they are! Might like to use the previous program for other reasons. One that comes straight to mind is creating the characters for a large font out of the moire effect created here. Who's bold enough to go for it!?

How about plain and simple tidyness of a program display. One such method is to add borders to a wide screen. For example;

```
10 POKE 559,35
12 POKE 53256,3:POKE 53257,3
14 POKE 53261,255:POKE 53262,255
16 POKE 704,0:POKE 705,0
18 POKE 53248,24:POKE 53249,203
```

That's not the only use, though, it could be used deceptively to make some people think that you can place graphics 0 text on the border! Hmmm, I wonder if I've given the secret away?

Another aspect of program improvement is the special effects department. Here's a simple one to wet your appetite:

```
100 GRAPHICS 0
110 POKE 82,0:POKE 710,0
120 FOR I=0 TO 80:? "HMMM  ";
130 NEXT I
140 DL=PEEK(560)+256*PEEK(561)
150 AFFECT=240
160 --
170 Z=112:V1=0:V2=16:Q1=0:Q2=-255
180 GOSUB AFFECT
190 --
200 Z=0:V1=16:V2=0:Q1=+255:Q2=112
210 GOSUB AFFECT
220 GOTO 170
230 --
240 FOR I=6 TO 28 STEP 2
250 POKE DL+I,Z
260 NEXT I
270 Z=Z+V1-V2
280 FOR D=0 TO 49:NEXT D
290 IF Z]=Q1 OR Z[=Q2 THEN 240
300 RETURN
```

There is so much that you can do. You may also notice I tried to keep the listing short, since I've utilized the same FOR/NEXT loop to expand and reduce the graphics 0 display.

APPENDIX G4:

Again, not only from the special effects department but also from Page-6 magazine issue 41. Here's a very famous Atari effect:

```
12 GRAPHICS 0
14 POKE 710,0:POKE 623,1
16 POKE 53256,0:POKE 53261,1
18 FOR I=0 TO 33
20   READ D:POKE 1536+I,D
22 NEXT I
24 DATA 72,162,216,189,0,129,56,253,0
26 DATA 130,157,0,129,141,10,212,141
28 DATA 0,208,42,41,240,9,15,141,18
30 DATA 208,202,224,0,208,227,104,64
32 ------------------------------
34 DL=PEEK(560)+256*PEEK(561)
36 POKE 512,0:POKE 513,6:POKE DL,128
40 FOR I=0 TO 255
42   POKE 33024+I,PEEK(53770)
44   POKE 33280+I,INT(RND(0)*3)+1
46   POKE 33792+I,1
48   ? 255-I
50 NEXT I
52 POKE 54286,192
54 LIST
```

The original version of this program was done by Edward Brooksbank, but I've made some modifications and come up with this version above. You'll find that you can type Basic commands in, but it's best if you avoid this because the method in which the display is created is very time consuming! I would have written a fast method, but I think I was held back by a slight case of bone-idleness! Perhaps next time.

Away from special effects now and into the bits of the bytes, or the shapes of the character-set. This following program will take a single key input and return you with the making of that character you pressed.

The key you press is initially read from location 764. This value found in variable RAW is known as the hardware key-matrix value and is of no particular order. The ascii equivalent of this character is found via the use of the DFT and RAW variables, and ends up in variable K. But, since the program is meant to print out the bits of the character, we must still convert it to its internal code value. We do this on line 20. On reaching line 20, K is the ascii code, but on processing line 20, K becomes the internal code.

```
10 GRAPHICS 0:POKE 752,1
12 DFT=PEEK(121)+256*PEEK(122)
14 POKE 764,255
16 RAW=PEEK(764):GOTO 16+(RAW[255)*2
18 K=PEEK(DFT+RAW)
20 K=K+(K[32)*64-((K]31)-(K]95))*32
22 ADDR=52224+K*8
24 FOR I=0 TO 7
26 V=PEEK(ADDR+I)
28 ?
30 B=128
32 IF V-B]=0 THEN 38
34 ? "-";
36 GOTO 42
38 V=V-B
40 ? "*";
42 B=INT(B/2)
44 IF B]0 THEN 32
46 NEXT I
48 ?
50 GOTO 14
```

As you can see, line 20 is the boolean technique for the ascii to internal character code conversion. It might be useful to remember these boolean expressions, since they not only take up less program space, they execute faster and become inter-dependable on only 1 program line. You do not need to initiate variables used in boolean expressions because if the expression is done good enough, it will initiate itself. You can prove what I'm trying to explain with a program listed earlier in this appendix.

The disk directory is another task. Try this one:

```
10 OPEN #1,7,0,"D:*.*"
20 TRAP 60
30 GET #1,B
40 ? CHR$(B);
50 GOTO 30
60 TRAP 40000
70 CLOSE #1
```

Steering clear of TRAP, why not try:

```
10 DIM A$(20)
20 OPEN #1,7,0,"D:*.*"
30 INPUT #1;A$
40 ? A$
50 IF A$(1,3)["000" THEN 30
60 CLOSE #1
```

APPENDIX G4:

And now for something completely different. Here's a useful program for those of you who like the game Boulderdash and only have a 1 drive system:

```
110 REM *- BOULDERDASH SCREEN
112 REM *- COPIER AND ORGANISER
114 REM
116 REM *- ANDREW C. THOMPSON
118 REM *- ORIGINAL VERSION
120 REM *- FEB'91
122 REM
124 REM *- MODIFIED FAST VERSION
126 REM *- MAR'92
128 REM
130 REM
132 REM *- INIT
134 REM
136 DIM D$(19),E$(15),CIO$(7),L$(82)
138 DIM G$(640),F$(40*13+8),S$(1)
140 REM
142 REM *- INSTRUCTIONS
144 REM
146 GRAPHICS 0
148 LIST 110,126
150 ?
152 ? "This will copy any Boulderdash"
154 ? "game and its screens in"
156 ? "one disk pass."
158 ?
160 ? "The destination copy will be"
162 ? "best organized so as to reduce"
164 ? "wear and tear on the drive-head"
166 ? "when the files are loaded."
168 REM
170 REM *- SOURCE
172 REM
174 I=1
176 GOTO 178+4*(I]0)
178 ?
180 ? "NO GAMES ON THIS DISK"
182 ?
184 ? "INSERT YOUR BOULDERDASH"
186 ? "GAME DISK"
188 ?
190 ? "PRESS RETURN";
192 KEY=155
194 GOSUB 468
196 GOSUB 484
198 GOTO 178+28*(I]0)
200 REM
202 REM *- GET GAME NAME
204 REM
206 D$="D1:"
208 ?
210 ? "GAME NAME- ";D$;
212 INPUT #16;E$
214 D$(4)=E$
216 REM
218 REM *- CIO CALL ROUTINE
220 REM
221 DATA 104,162,16,32,86,228,96
222 FOR I=0 TO 6
223 READ D:POKE ADR(CIO$)+I,D:NEXT I
224 REM
226 REM PLA
228 REM LDX #$10
230 REM JSR $E456
232 REM RTS
234 REM
236 REM
238 REM *- GET GAME-FILES
240 REM
242 A=ADR(G$)
244 AUX1=4
246 ICCOM=7
248 L1=252
250 L2=252
252 CLINE=254:GOTO 510
254 L$(1)=CHR$(PEEK(856))
256 L$(2)=CHR$(PEEK(857))
258 REM
260 REM *- FIND AMOUNT OF SCREENS
262 REM
264 I=8
266 S=0
268 GOTO 270+2*(PEEK(A+I)[]46)
270 S=S+1
272 I=I+13
274 GOTO 268+14*(I]40*13+8-1)
276 REM
278 REM *- ZERO SCREENS CHECK
280 REM
282 ?
284 GOTO 286+12*(S]0)
286 ? "THERE ARE NO SCREENS IN THIS"
288 ? "GAME FILE!"
290 STOP
292 REM
294 REM *- GET SCREEN NAME
296 REM
298 B=A
```

```
300 A=ADR(S$)
302 C=0
304 F$=""
306 D$="D1:"
308 I=0
310 GOTO 312+40*(C]S-1)
312 W=PEEK(B+C*13+I)
314 GOTO 316+2*(W=32)
316 D$(LEN(D$)+1)=CHR$(W)
318 I=I+1
320 GOTO 312+10*(I]11)
322 F$(C*12+1)=D$(4)
324 F$(LEN(F$)+1)="          "
326 ? ,D$(4)
328 REM
330 REM *- GET SCREEN
332 REM
334 CLINE=336:GOTO 510
336 L$(LEN(L$)+1)=CHR$(PEEK(856))
338 L$(LEN(L$)+1)=CHR$(PEEK(857))
340 A=A+505
342 C=C+1
344 GOTO 306
346 REM
348 REM *- DESTINATION
350 REM
352 ?
354 ? "INSERT YOUR DESTINATION"
356 ? "DOS-FORMAT DISK"
358 ?
360 ? "PRESS RETURN"
362 K=155
364 GOSUB 468
366 REM
368 REM *- CONFIRM
370 REM
372 ? "CONFIRM! WRITE? ]";
374 KEY=89
376 GOSUB 468
378 ?
380 REM
382 REM *- PUT GAME-FILE
384 REM
386 D$="D1:"
388 D$(4)=E$
390 A=ADR(G$)
392 AUX1=8
394 ICCOM=11
396 L1=ASC(L$(1))
398 L2=ASC(L$(2))
400 CLINE=408:GOTO 510
402 REM
404 REM *- PUT SCREENS
406 REM
408 ?
410 A=ADR(S$)
412 C=0
414 D$="D1:"
416 D$(4)=F$(C*12+1,C*12+1+11)
418 ? ,D$(4)
420 L1=ASC(L$(C*2+3))
422 L2=ASC(L$(C*2+4))
424 CLINE=426:GOTO 510
426 A=A+505
428 C=C+1
430 GOTO 414+24*(C]S-1)
432 REM
434 REM *- WRITE AGAIN?
436 REM
438 ?
440 ? E$;" HAS BEEN COPIED"
442 ? "WRITE AGAIN? ]"
444 KEY=89
446 GET #3,K
448 IF K=89 THEN GOTO 352
450 RUN
452 REM
454 REM *- SUBROUTINES
456 REM *- -----------
458 REM
460 REM
462 REM
464 REM *- GET KEY
466 REM
468 CLOSE #3
470 OPEN #3,4,0,"K:"
472 GET #3,K
474 GOTO 472+4*(KEY=K)
476 RETURN
478 REM
480 REM *- DIRECTORY
482 REM
484 CLOSE #2
486 OPEN #2,7,0,"D:*.GAM"
488 I=0
490 ?
492 INPUT #2;D$
494 GOTO 496+6*(D$(1,3)]="000")
496 I=I+1
498 ? ,D$
500 GOTO 492
502 RETURN
504 REM
506 REM *- CIO EXECUTE
508 REM
510 HI=INT(A/256)
512 LO=A-(HI*256)
514 CLOSE #1
516 OPEN #1,AUX1,0,D$
518 POKE 850,ICCOM
520 POKE 852,LO
```

```
522 POKE 853,HI
524 POKE 856,L1
526 POKE 857,L2
528 X=USR(ADR(CIO$))
530 CLOSE #1
532 GOTO CLINE
```

If you've made any screens with the Boulderdash Construction Kit, then this program will make a copy of them screens in 1 disk pass. Just RUN the program up, type in the name of the game file you want to copy and leave it to read in all the files. When you insert a DOS formatted destination disk, the program will then write all those screens in 1 go.

Music is an essential addition to programs as well as pictures and graphics affects, here's a relatively straightforward assembly program that I originally received from a friend (Hiya Phil) several months back. Try it:

```
100             ;ASSEMBLER MUSIC
160 ;
164 ;
170             RTCLOK = $12
180             TIMER  = $CB
190             AUDF1  = $D200
200             AUDC1  = $D201
210             AUDF2  = $D202
220             AUDC2  = $D203
230             AUDF3  = $D204
240             AUDC3  = $D205
250             AUDCTL = $D208
260             SKCTL  = $D20F
270 ;
280             *=$4000        ;START ADR
290 ;
300             LDA #0         ;INIT
310             STA AUDCTL
320             LDA #3
330             STA SKCTL
340             LDA #168
350             STA AUDC1
360             STA AUDC2
370             STA AUDC3
380 ;
430             LDX #0         ;LOAD
450 NEXTNOTE    LDA CHAN1,X    ;NOTES
460             STA AUDF1
470             LDA CHAN2,X
480             STA AUDF2
490             LDA CHAN3,X
500             STA AUDF3
510             LDA #8         ;SET
```

```
520             STA TIMER     ;TIMER
530             LDA #0
540             STA TIMER+1
550             JSR DELAY     ;WAIT
560             INX
570             LDA CHAN1,X   ;LOAD
580             STA AUDF1     ;NOTES
590             LDA CHAN2,X
600             STA AUDF2
610             LDA CHAN3,X
620             STA AUDF3
630             LDA #5        ;SET
640             STA TIMER     ;TIMER
650             JSR DELAY     ;WAIT
660             LDA #0
670             STA AUDF1     ;CLEAR
680             STA AUDF2     ;CHANNELS
690             STA AUDF3
700             LDA #3        ;SET
710             STA TIMER     ;TIMER
720             JSR DELAY     ;WAIT
730             INX
740             CPX #26       ;FINISHED?
750             BNE NEXTNOTE  ;NOPE
760             BRK           ;YEAP
770 ;
790 DELAY       LDA #0
800             STA RTCLOK+1
810             STA RTCLOK+2
830 LOOP1       LDA RTCLOK+1
840             CMP TIMER+1
850             BNE TOOP1
870 LOOP2       LDA RTCLOK+2
880             CMP TIMER
890             BNE LOOP2
900             RTS
910 ;
930 CHAN1        .BYTE 60,53,60,64,68,68,47,47
940              .BYTE 45,45,60,60,68,68,60,60
950              .BYTE 72,72,60,60,81,81
960              .BYTE 60,60,91,91
980 CHAN2        .BYTE 0,0,0,0,0,0,60,60,60,60,0
990              .BYTE 0,0,0,0,0,121,121,136,136
1000             .BYTE 144,144,162,162,182,182
1020 CHAN3       .BYTE 0,0,0,0,0,0,68,68,72,72
1030             .BYTE 0,0,0,0,0,0,0,0
1040             .BYTE 0,0,0,0,0,0,0,0
1050             END
```

It's not bad is it. The listing is fairly well remarked, including the numbers in the data (.BYTE). Try changing some of the values and see how good you can compose.

APPENDIX G4:

Have you ever wondered how to load machine-code files from Basic? Well, one method would be with the following program:

```
10 CLOSE #1
20 OPEN #1,4,0,"D:FILENAME.EXT"
30 X=USR(5576)
```

This is usually very effective, but it doesn't always work. Some machine-code files are very awkward and they just won't load. Well, there is a solution to this problem, and you'll find it looks something like:

```
10 DATA 162,16,169,3,157,66,3,169,1,157,65,3,169,4,157,74,3
20 DATA 169,33,157,68,3,169,6,157,69,3,32,86,228,76,200,21
30 DATA 68,49,58,69,71,79,46,67,79,77,155,-1
40 FOR I=0 TO 43:READ D:POKE 1536+I,D
50 NEXT I:POKE 5446,0:POKE 5450,6
60 POKE 1016,1:X=USR(58484)
```

In fact, believe it or not, the 2nd program does exactly what the 1st program does. The only difference is that the 2nd program disables Basic, which is the reason why some machine-code files wouldn't previously load, they need the space that Basic normally occupys!

The file that is OPENed for loading off the disk is all in ascii codes on line 30. The present line translates to: D1:EGO.COM (CR). The (CR) is a carriage-return character, code 155.

Here's a handy program for users of MJ and DW's DATABASE program. It will printout your data to a 1029.

```
110 REM **   "DATABASE" FILE
114 REM **   1029 PRINTOUT UTILITY
118 REM **   ANDREW C. THOMPSON
122 REM **   APR'92
126 REM
140 GRAPHICS 0:LIST 110,134
144 ? :?
146 ? "Insert DATABASE program"
148 ? "in drive #1 and prepare 1029."
150 ?
152 ? "Program will print during"
154 ? "input of data file."
156 ?
158 DIM I$(23),P$(76)
160 ? "Press RETURN";
162 INPUT #16;P$
164 CLOSE #1: OPEN #1,4,0,"D:PROGDAT.DAT"
170 C=0
172 INPUT #1;I$:REM 2*ascii-00 RIDDER
```

```
174 TRAP 220
176 P$="
"
178 INPUT #1;I$
180 P$(1)=I$(1,3):P$(5)=I$(4)
182 INPUT #1;I$
184 P$(27)=I$(1,3):P$(31)=I$(4)
186 INPUT #1;I$
188 P$(53)=I$(1,3):P$(57)=I$(4)
200 LPRINT P$
202 C=C+1
204 IF C/63=INT(C/63) THEN LPRINT :LPRINT :LPRINT
210 IF PEEK(53279)=3 THEN 300
214 GOTO 176
220 LPRINT P$
230 CLOSE #1
232 ? :? "The DATABASE file is printed."
236 END
300 IF PEEK(53279)[]5 THEN 300
302 GOTO 214
```

OK then, here's perhaps a very useful listing. It will convert a Revision-B ROM into a Revision-C one.

```
100 REM ** 64K+ XL/XE REV. B(UGS)
102 REM ** TO REV. C BASIC CONVERTER.
104 REM ** MATT RATCLIFF  4.5.85
106 -----------------------------
120 RESTORE
130 DIM A$(10)
140 GRAPHICS 0
150 ? "Prepare DOS disk for the"
160 ? "destination AUTORUN.SYS file."
162 ?
164 ? "Press RETURN";
170 TRAP 300
180 INPUT #16;A$
190 OPEN #1,8,0,"D:AUTORUN.SYS"
200 READ A
210 IF A[0 THEN 240
220 PUT #1,A
230 GOTO 200
240 CLOSE #1
250 ?
260 ? "** ALL DONE **"
270 ? "Don't forget to save this file"
280 ? "for backup also!"
290 GOTO 320
300 ? "ERROR-  ";PEEK(195); "AT LINE ";
310 ? PEEK(186)+256*PEEK(187)
320 STOP
330 -----------------------------
```

```
400 DATA 255,255,0,6,130,6
402 DATA 169,0,133,2,169,6,133,3
404 DATA 173,250,3,240,1,96,169,0
406 DATA 133,206,169,160,133,217,160,0
408 DATA 173,1,211,41,253,141,1,211
410 DATA 177,216,72,173,1,211,9,2
412 DATA 141,1,211,104,145,216,230,216
414 DATA 208,228,230,217,165,217
416 DATA 201,192,208,220,162,0,169,12
418 DATA 133,218,160,0,189,95,6,133,216
420 DATA 232,189,95,6,133,217,232
422 DATA 189,95,6,145,216,232,198
424 DATA 218,208,232,165,9,9,2,133,9,96
426 DATA 223,168,234,224,168,240,225
428 DATA 168,17,226,168,234,41
430 DATA 187,0,243,191,0,244
432 DATA 191,0,245,191,0,246
434 DATA 191,0,247,191,0,248
436 DATA 191,0,249,191,0
438 DATA 226,2,227,2,0,6,-1
```

C at last! No more bugs for you now thanks to Matt Ratcliff' program.

A database program is next, this RKM Filer #3 will retain a wide range of data, whatever you want to throw at it. The original version was written for my brother to file his heavy metal records. Here's the latest and last version I care to make of it. Use it for what you will:

```
110 REM ** RKM FILER-#3
120 REM ** PROGRAMMED 1991 BY
130 REM ** ANDREW C. THOMPSON
140 REM ** THE MODIFIED VERSION
150 REM ** ON THE BASIS OF
160 REM ** RKM FILER-#2 VER.2
170 REM ** PROGRAMMED EARLY 1985
180 REM ** RKM FROM PAGE-6 ISS.8
190 REM
200 REM ** INITIALISE
202 REM
210 GRAPHICS 0
220 POKE 82,1:POKE 83,38
230 POKE 709,12:POKE 710,132
240 POKE 729,48:POKE 730,3
250 POKE 731,0
260 REM
270 REM ** VARS
280 DIM E$(102),P$(102)
290 REM
300 REM ** OPEN GLOBAL CHANNELS
310 OPEN #1,4,0,"K:"
320 REM
330 REM ** DISPLAY MAIN-MENU
```

APPENDIX G4:

```
340 ? CHR$(125)
350 ? "RKM FILER-#3, APRIL 1991"
360 ? "THE MODIFIED VERSION OF"
370 ? "RKM FILER-#2, EARLY 1985"
380 ? :?
390 ? "(1) INPUT FROM DEVICE"
400 ? "(2) OUTPUT TO DEVICE"
410 ? "(3) DISPLAY ENTRIE/S"
420 ? "(4) ADD ENTRIE/S"
430 ? "(5) CHANGE ENTRIE/S"
440 ? :?
450 ? "THIS RKM FILER CAN BE USED"
460 ? "FOR STORING MISCALLANEOUS"
470 ? "DATA LIKE A RECORD LIST"
480 ?
490 ? "NOTE THAT WHEN ADDING ANY"
500 ? "ENTRIES, YOU CANNOT USE"
510 ? "THE COMMA, REFER TO USING"
520 ? "A SEMI-COLON (;) INSTEAD"
530 ? "FREE MEMORY =";FRE(0)
540 REM
550 REM ** GET KEY
560 KHI=5
570 GOSUB 750
580 ON K GOTO 870,1090,1340,1960,2250
590 GOTO 560
600 REM
610 REM ** PROMPT
620 ?
630 ? "READY APPROPRIATE MEDIA"
640 ? "PRESS START"
650 IF PEEK(53279)[]6 THEN 650
660 RETURN
670 REM
680 REM ** I/O MENU
690 ?
700 ? "(1) D1:RKMF3"
710 ? "(2) C1:RKMF3"
720 RETURN
730 REM
740 REM ** KEY LIMITER
750 GET #1,K
760 IF K=27 THEN 790
770 K=K-48
780 IF K[1 OR K]KHI THEN 750
790 RETURN
800 REM
810 REM ** SPECIAL-CHECK
820 K2=PEEK(764)
830 K2=NOT (K2=255)
840 RETURN
850 REM
860 REM ** LOAD RKM F#3
870 ? CHR$(125)
880 ? "(1) INPUT FROM DEVICE"
890 GOSUB 690
900 KHI=2
910 GOSUB 750
920 IF K=27 THEN 340
930 GOSUB 620
940 GOTO 940+(K*10)
950 RUN "D:RKMFIL3.SAV"
960 ? CHR$(125)
970 POSITION 1,16
980 ? "POKE 842,12:POKE 764,12"
990 ? :?
1000 ? "RUN"
1010 POSITION 1,0
1020 ? "PLEASE WAIT"
1030 ? "PROGRAM LOADING"
1040 POKE 764,12
1050 POKE 842,13
1060 CLOAD
1070 REM
1080 REM ** SAVE RKM F#3
1090 ? CHR$(125)
1100 ? "(2) OUTPUT TO DEVICE"
1110 GOSUB 690
1120 ? "(3) HARDCOPY"
1130 KHI=3
1140 GOSUB 750
1150 IF K=27 THEN 340
1160 GOSUB 620
1170 ON K GOTO 1180,1200,1220
1180 SAVE "D:RKMFIL3.SAV"
1190 GOTO 340
1200 CSAVE
1210 GOTO 340
1220 RESTORE 9970
1230 READ LI
1240 C=10000
1250 RESTORE C
1260 READ E$
1270 IF E$="*" THEN 1290:REM * = inverse
1280 LPRINT E$
1290 C=C+2
1300 IF C[=LI THEN 1250
1310 GOTO 340
1320 REM
1330 REM ** VIEWER
1340 ? CHR$(125)
1350 ? "(3) DISPLAY ENTRIE/S"
1360 ?
1370 ? "(1) DISPLAY ALL DATA"
1380 ? "(2) DISPLAY BY PREFIX"
1390 KHI=2
1400 GOSUB 750
1410 ON K GOTO 1550,1720
1420 GOTO 340
1430 REM
```

```
1440 REM ** DATA-CHECK
1450 RESTORE 9970
1460 READ LI
1470 IF LI]10000 THEN 1520
1480 ?
1490 ? "THERE ARE NO ENTRIES!"
1500 ? :?
1510 LI=0
1520 RETURN
1530 REM
1540 REM ** VIEW ALL
1550 GOSUB 1450
1560 IF LI THEN 1580
1570 GOTO 1350
1580 ?
1590 C=10000
1600 POKE 764,255
1610 RESTORE C
1620 READ E$
1630 ? E$
1640 GOSUB 820
1650 C=C+2
1660 GOTO 1670+(K2*10)
1670 IF C[LI THEN 1610
1680 ? :?
1690 GOTO 1350
1700 REM
1710 REM ** PREFIX
1720 GOSUB 1450
1730 IF LI THEN 1750
1740 GOTO 1350
1750 ?
1760 ? "[ESC] [ESC] [RETURN] = EXIT"
1770 ? "PLEASE TYPE CHARACTER PREFIX"
1780 INPUT P$
1790 IF P$="" THEN 1750
1800 IF P$="[ESC]" THEN 1340
1810 L=LEN(P$)
1820 RESTORE 9970
1830 READ LI
1840 C=10000
1850 RESTORE C
1860 READ E$
1870 IF LEN(E$)]=L THEN 1920
1880 C=C+2
1890 IF C[LI THEN 1850
1900 ? :?
1910 GOTO 1350
1920 IF P$=E$(1,L) THEN ? E$
1930 GOTO 1880
1940 REM
1950 REM ** ENTRY ADDITION
1960 ? CHR$(125)
1970 ? "(4) ADD ENTRIE/S"
1980 ?
1990 ? "[ESC] [ESC] [RETURN] = EXIT"
2000 ? "ENTER NEW ENTRY"
2010 INPUT P$
2020 IF P$="" THEN 1980
2030 IF P$="[ESC]" THEN 340
2040 IF FRE(0)-135 THEN 2110
2050 ?
2060 ? "OUT OF MEMORY!"
2070 ? "INITIATE ANOTHER FILE!"
2080 GOSUB 820
2090 IF NOT K2 THEN 2080
2100 GOTO 340
2110 RESTORE 9970
2120 READ LI
2130 ? CHR$(125)
2140 ?
2150 ? 9970;" DATA ";LI+2
2160 ? LI;" DATA ";P$
2170 ? "CONT"
2180 POSITION 0,0
2190 POKE 842,13
2200 STOP
2210 POKE 842,12
2220 GOTO 1980
2230 REM
2240 REM ** ENTRY CHANGES
2250 ? CHR$(125)
2260 ? "(5) CHANGE ENTRIE/S"
2270 ?
2280 ? "[ESC] [ESC] [RETURN] = EXIT"
2290 ? "CHANGE WHICH ENTRY"
2300 INPUT #16;P$
2310 IF P$="" THEN 2270
2320 IF P$="[ESC]" THEN 340
2330 L=LEN(P$)
2340 RESTORE 9970
2350 READ LI
2360 C=10000
2370 RESTORE C
2380 READ E$
2390 IF LEN(E$)]=L THEN 2430
2400 C=C+2
2410 IF C[LI THEN 2370
2420 GOTO 2270
2430 IF P$[]E$(1,L) THEN 2400
2440 ? :?
2450 ? E$
2460 ? "[ESC] [ESC] [RETURN] = EXIT"
2470 ? "CHANGE ABOVE ENTRY TO"
2480 INPUT #16;P$
2490 IF P$="" THEN P$="*":REM * = inverse
2500 IF P$="[ESC]" THEN 2270
2510 ? CHR$(125)
2520 ?
2530 ? C;" DATA ";P$
```

```
2540 ? "CONT"
2550 POSITION 0,0
2560 POKE 842,13
2570 STOP
2580 POKE 842,12
2590 ? :?
2600 GOTO 2270
9950 REM
9960 REM ** WRITE-POINTER
9970 DATA 10000
9980 REM
9990 REM ** FILE
10000 DATA DEFAULT
32767 STOP
```

This particular filer appends any inputted information at the end of the actual program listing. It does this with the use of what's known as the Return Key Mode (RKM) which I first read about in issue #8 of Page-6 magazine (the very 1st issue I bought too!). For more information about this RK mode, take a visit to page-80 in the MAP section of this book.

And finally, to end this appendix and indeed this programming reference book, I leave you with the disassembling program that I wrote to disassemble the Atari' Operating system.

```
100 GOTO 150
101 ------------------------------
102 HI=INT(B1/16):LO=B1-HI*16
104 ? #1;H$(HI+1,HI+1);H$(LO+1,LO+1);
106 RETURN
107 ------------------------------
108 HI=INT(B2/16):LO=B2-HI*16
110 ? #1;H$(HI+1,HI+1);H$(LO+1,LO+1);
112 RETURN
113 ------------------------------
114 HI=INT(LOC/4096):RLOC=LOC-HI*4096
116 M1=INT(RLOC/256):RLOC=RLOC-M1*256
118 M2=INT(RLOC/16):LO=RLOC-M2*16
122 ? #1;H$(HI+1,HI+1);H$(M1+1,M1+1);
124 ? #1;H$(M2+1,M2+1);H$(LO+1,LO+1);
126 RETURN
127 ------------------------------
128 HI=INT(MCI/16):LO=MCI-HI*16
130 ? #1;H$(HI+1,HI+1);H$(LO+1,LO+1);
132 RETURN
133 ------------------------------
134 HI=INT(NLOC/4096)
136 SLOC=NLOC-HI*4096
138 M1=INT(SLOC/256):SLOC=SLOC-M1*256
140 M2=INT(SLOC/16)
```

```
142 LO=SLOC-M2*16
144 ? #1;H$(HI+1,HI+1);H$(M1+1,M1+1);
146 ? #1;H$(M2+1,M2+1);H$(LO+1,LO+1);
148 RETURN
149 ------------------------------
150 DIM AI$(3),H$(16)
152 H$="0123456789ABCDEF"
154 OPEN #1,8,0,"S:"
156 POKE 752,0
157 ? "START ADDRESS [DEC] ";:INPUT S
158 IF S=-1 THEN CLOSE #1:STOP
160 ? "  END ADDRESS [DEC] ";:INPUT E
162 ?
163 POKE 752,1
164 FOR LOC=S TO E
166   MCI=PEEK(LOC)
168   RESTORE 626+MCI
170   READ MODE,AI$
172   GOSUB 114:? #1;"  ";
173   GOSUB 128:? #1;" ";
174   B1=PEEK(LOC+1)
176   B2=PEEK(LOC+2)
178   GOSUB 186+(MODE*10)
180 NEXT LOC
182 ? #1
183 GOTO 156
184 ------------------------------
186 GOSUB 102
187 ? #1;"        ";AI$;" #$";
188 GOSUB 102:? #1:LOC=LOC+1
190 RETURN
196 GOSUB 102:? #1;" ";:GOSUB 108
198 ? #1;"     ";AI$;"  $";:GOSUB 108
200 GOSUB 102:? #1:LOC=LOC+2
202 RETURN
206 GOSUB 102
207 ? #1;"        ";AI$;"  $";
208 GOSUB 102:? #1:LOC=LOC+1
210 RETURN
216 ? #1;"          ";AI$;" A"
218 RETURN
226 ? #1;"          ";AI$
228 RETURN
236 GOSUB 102
237 ? #1;"        ";AI$;" ($";
238 GOSUB 102:? #1;",X)":LOC=LOC+1
240 RETURN
246 GOSUB 102
247 ? #1;"        ";AI$;" ($";
248 GOSUB 102:? #1;"),Y":LOC=LOC+1
250 RETURN
256 GOSUB 102
257 ? #1;"        ";AI$;"  $";
258 GOSUB 102:? #1;",X":LOC=LOC+1
260 RETURN
```

```
266 GOSUB 102:? #1;" ";:GOSUB 108
268 ? #1;"      ";AI$;"  $";:GOSUB 108
270 GOSUB 102:? #1;",X":LOC=LOC+2
272 RETURN
276 GOSUB 102:? #1;" ";:GOSUB 108
278 ? #1;"      ";AI$;"  $";:GOSUB 108
280 GOSUB 102:? #1;",Y":LOC=LOC+2
282 RETURN
286 GOSUB 102
287 ? #1;"         ";AI$;"  $";
288 IF B1]127 THEN 291
289 NLOC=LOC+2+B1:GOSUB 134:? #1
290 GOTO 293
291 NLOC=LOC+1-(255-B1):GOSUB 134
292 ? #1
293 LOC=LOC+1
294 RETURN
296 GOSUB 102:? #1;" ";:GOSUB 108
298 ? #1;"      ";AI$;" ($";:GOSUB 108
300 GOSUB 102:? #1;")":LOC=LOC+2
302 RETURN
306 GOSUB 102
307 ? #1;"         ";AI$;"  $";
308 GOSUB 102:? #1;",Y":LOC=LOC+1
310 RETURN
616 ------------------------------
626 DATA 4,BRK
627 DATA 5,ORA
628 DATA 4,???
629 DATA 4,???
630 DATA 4,???
631 DATA 2,ORA
632 DATA 2,ASL
633 DATA 4,???
634 DATA 4,PHP
635 DATA 0,ORA
636 DATA 3,ASL
637 DATA 4,???
638 DATA 4,???
639 DATA 1,ORA
640 DATA 1,ASL
641 DATA 4,???
642 DATA 10,BPL
643 DATA 6,ORA
644 DATA 4,???
645 DATA 4,???
646 DATA 4,???
647 DATA 7,ORA
648 DATA 7,ASL
649 DATA 4,???
650 DATA 4,CLC
651 DATA 9,ORA
652 DATA 4,???
653 DATA 4,???
654 DATA 4,???
655 DATA 8,ORA
656 DATA 8,ASL
657 DATA 4,???
658 DATA 1,JSR
659 DATA 5,AND
660 DATA 4,???
661 DATA 4,???
662 DATA 2,BIT
663 DATA 2,AND
664 DATA 2,ROL
665 DATA 4,???
666 DATA 4,PLP
667 DATA 0,AND
668 DATA 3,ROL
669 DATA 4,???
670 DATA 1,BIT
671 DATA 1,AND
672 DATA 1,ROL
673 DATA 4,???
674 DATA 10,BMI
675 DATA 6,AND
676 DATA 4,???
677 DATA 4,???
678 DATA 4,???
679 DATA 7,AND
680 DATA 7,ROL
681 DATA 4,???
682 DATA 4,SEC
683 DATA 9,AND
```

APPENDIX G4:

```
684 DATA 4,???
685 DATA 4,???
686 DATA 4,???
687 DATA 8,AND
688 DATA 8,ROL
689 DATA 4,???
690 DATA 4,RTI
691 DATA 5,EOR
692 DATA 4,???
693 DATA 4,???
694 DATA 4,???
695 DATA 2,EOR
696 DATA 2,LSR
697 DATA 4,???
698 DATA 4,PHA
699 DATA 0,EOR
700 DATA 3,LSR
701 DATA 4,???
702 DATA 1,JMP
703 DATA 1,EOR
704 DATA 1,LSR
705 DATA 4,???
706 DATA 10,BVC
707 DATA 6,EOR
708 DATA 4,???
709 DATA 4,???
710 DATA 4,???
711 DATA 7,EOR
712 DATA 7,LSR
713 DATA 4,???
714 DATA 4,CLI
715 DATA 9,EOR
716 DATA 4,???
717 DATA 4,???
718 DATA 4,???
719 DATA 8,EOR
720 DATA 8,LSR
721 DATA 4,???
722 DATA 4,RTS
723 DATA 5,ADC
724 DATA 4,???
725 DATA 4,???
726 DATA 4,???
727 DATA 2,ADC
728 DATA 2,ROR
729 DATA 4,???
730 DATA 4,PLA
731 DATA 0,ADC
732 DATA 3,ROR
733 DATA 4,???
734 DATA 11,JMP
735 DATA 1,ADC
736 DATA 1,ROR
737 DATA 4,???
738 DATA 10,BVS
739 DATA 6,ADC
740 DATA 4,???
741 DATA 4,???
742 DATA 4,???
743 DATA 7,ADC
744 DATA 7,ROR
745 DATA 4,???
746 DATA 4,SEI
747 DATA 9,ADC
748 DATA 4,???
749 DATA 4,???
750 DATA 4,???
751 DATA 8,ADC
752 DATA 8,ROR
753 DATA 4,???
754 DATA 4,???
755 DATA 5,STA
756 DATA 4,???
757 DATA 4,???
758 DATA 2,STY
759 DATA 2,STA
760 DATA 2,STX
761 DATA 4,???
762 DATA 4,DEY
763 DATA 4,???
764 DATA 4,TXA
765 DATA 4,???
766 DATA 1,STY
767 DATA 1,STA
768 DATA 1,STX
769 DATA 4,???
770 DATA 10,BCC
771 DATA 6,STA
772 DATA 4,???
773 DATA 4,???
774 DATA 7,STY
775 DATA 7,STA
776 DATA 7,STX
777 DATA 4,???
778 DATA 4,TYA
779 DATA 9,STA
780 DATA 4,TXS
781 DATA 4,???
782 DATA 4,???
783 DATA 8,STA
784 DATA 4,???
785 DATA 4,???
786 DATA 0,LDY
787 DATA 5,LDA
788 DATA 0,LDX
789 DATA 4,???
790 DATA 2,LDY
791 DATA 2,LDA
792 DATA 2,LDX
793 DATA 4,???
```

```
794 DATA 4,TAY
795 DATA 0,LDA
796 DATA 4,TAX
797 DATA 4,???
798 DATA 1,LDY
799 DATA 1,LDA
800 DATA 1,LDX
801 DATA 4,???
802 DATA 10,BCS
803 DATA 6,LDA
804 DATA 4,???
805 DATA 4,???
806 DATA 7,LDY
807 DATA 7,LDA
808 DATA 7,LDX
809 DATA 4,???
810 DATA 4,CLV
811 DATA 9,LDA
812 DATA 4,TSX
813 DATA 4,???
814 DATA 8,LDY
815 DATA 8,LDA
816 DATA 9,LDX
817 DATA 4,???
818 DATA 0,CPY
819 DATA 5,CMP
820 DATA 4,???
821 DATA 4,???
822 DATA 2,CPY
823 DATA 2,CMP
824 DATA 2,DEC
825 DATA 4,???
826 DATA 4,INY
827 DATA 0,CMP
828 DATA 4,DEX
829 DATA 4,???
830 DATA 1,CPY
831 DATA 1,CMP
832 DATA 1,DEC
833 DATA 4,???
834 DATA 10,BNE
835 DATA 6,CMP
836 DATA 4,???
837 DATA 4,???
838 DATA 4,???
839 DATA 7,CMP
840 DATA 7,DEC
841 DATA 4,???
842 DATA 4,CLD
843 DATA 9,CMP
844 DATA 4,???
845 DATA 4,???
846 DATA 4,???
847 DATA 8,CMP
848 DATA 8,DEC
849 DATA 4,???
850 DATA 0,CPX
851 DATA 5,SBC
852 DATA 4,???
853 DATA 4,???
854 DATA 2,CPX
855 DATA 2,SBC
856 DATA 2,INC
857 DATA 4,???
858 DATA 4,INX
859 DATA 0,SBC
860 DATA 4,NOP
861 DATA 4,???
862 DATA 1,CPX
863 DATA 1,SBC
864 DATA 1,INC
865 DATA 4,???
866 DATA 10,BEQ
867 DATA 6,SBC
868 DATA 4,???
869 DATA 4,???
870 DATA 4,???
871 DATA 7,SBC
872 DATA 7,INC
873 DATA 4,???
874 DATA 4,SED
875 DATA 9,SBC
876 DATA 4,???
877 DATA 4,???
878 DATA 4,???
879 DATA 8,SBC
880 DATA 8,INC
881 DATA 4,???
```

APPENDIX G4:

In it's present form it will disassemble from memory to screen, but should you want to have your disassembly on another media such as the printer, or cassette then simply put the necessary alteration in line 154.

The DATA statements from lines 626 to 881 are the actual assembly instructions and their addressing mode. You'll notice that there are many ??? instructions. These are illegal codes, and if you wish to include the actual illegal instruction names and modes then you can gather the information from the machine-code appendix. Also, if you want to know what each instructions decimal code is then simply take 626 off the line number and voila, there you have it!

Well, that about brings this last of the last appendices to an end.

I very nearly forgot to include a particular program, demonstrating Graphics mode 0.5! I was just about to finish this appendix, and indeed the book (excluding an introduction) too.

Here it is folks, Graphics 0.5 with true descenders and some quirky positioned characters: see next page.

APPENDIX G4:

```
10 POKE 106,PEEK(106)-4
12 GRAPHICS 0
14 DL=PEEK(560)+256*PEEK(561)
16 POKE DL+3,64+3
18 FOR I=6 TO 24
20   POKE DL+I,3
22 NEXT I
24 FOR I=0 TO 2
26   POKE DL+25+I,PEEK(DL+29+I)
28 NEXT I
34 NSET=PEEK(106)
36 FOR I=0 TO 1023
38   POKE NSET*256+I,PEEK(57344+I)
40 NEXT I
42 POKE 756,NSET
44 FOR I=0 TO 12
46   READ CH
48   FOR J=0 TO 7
50     READ ROW
52     POKE NSET*256+CH*8+J,ROW
54   NEXT J
56 NEXT I
58 ? "abcdefghijklmnopqrstuvwxyz"
60 STOP
70 DATA 98,0,0,96,96,124,102,102,124
72 DATA 100,0,0,6,6,62,102,102,62
74 DATA 102,0,0,14,24,62,24,24,24
76 DATA 103,6,124,0,62,102,102,62,6
78 DATA 104,0,0,96,96,124,102,102,102
80 DATA 105,0,0,24,0,56,24,24,60
82 DATA 106,6,60,6,0,6,6,6,6
84 DATA 107,0,0,96,96,108,120,108,102
86 DATA 108,0,0,56,24,24,24,24,60
88 DATA 112,96,96,0,124,102,102,124,96
90 DATA 113,6,6,0,62,102,102,62,6
92 DATA 116,0,0,24,126,24,24,24,14
94 DATA 121,6,126,0,102,102,102,126,6
```

Don't forget now, you can also use this special mode with the international character-set to allow you more room for the umlauts and all that stuff above the characters.

Now then, is this really the end? I think so... Well, I reckon I'll use that last full-stop now, happy programming and good luck in the future to all my contacts and the rest of you Atari 8-BIT freaks.

COMPLETE & ESSENTIAL MAP

for the

XL / XE

BOOK CORRECTIONS

After reading through the book we have unfortunately found a few page references that do not correspond with the pages indicated in the book.

On page 15 in the paragraph under location 91,92 it indicates to refer to page 97, unfortunately it should read: "See page 85 of the map".

On page 140 in the first paragraph, under location 54272, it reads (Page-45) but it should read "Page-38".

In part two of the book on page 170 in the OPEN paragraph it reads: (See the table on page 96), this is another mistake, it should read "See the table on page 84".

These mistakes have occurred when the author's Master Copy was set up and re-printed as it is now. There were too many large gaps between the lines and some pages had only a few line on them, it would have pushed the cost up too high. Please notify TWAUG with any other errors found in the book, the page references above are the only ones I've found up to now.

The author wasn't able to print the "lesser than < and greater than > characters with his printer, in place he used the square brackets []. Again some of these characters were overlooked, you will find these square brackets in some of the BASIC program listings, mostly in the appendix pages. Please replace these square brackets [] with the lesser than and greater than <> characters, or the programmes wont run.

If we find further mistakes we will update this 'Book correction leaflet' and post it out to our customers. Please keep this leaflet clipped to your book.

TYNE&WEAR ATARI USER GROUP

T.W.A.U.G.

P.O.Box No.8, WALLSEND

Tyne & Wear NE28 6DQ

NEWS Publishers

TWAUG publications presents

THE

Atari XL/XE

Complete And Essential MAP

Including Probably The Most
Comprehensive Appendice
Selection Ever Produced

Written by
Andrew C. Thompson

This Book Contains Information
Never Released Anywhere Before
And Is Heavily Based And
Expanded On Mapping The Atari - Revised

TYNE&WEAR
ATARI
USER GROUP

T.W.A.U.G.
P.O.Box No.8
Wallsend
TYNE & WEAR
NE28 6DQ