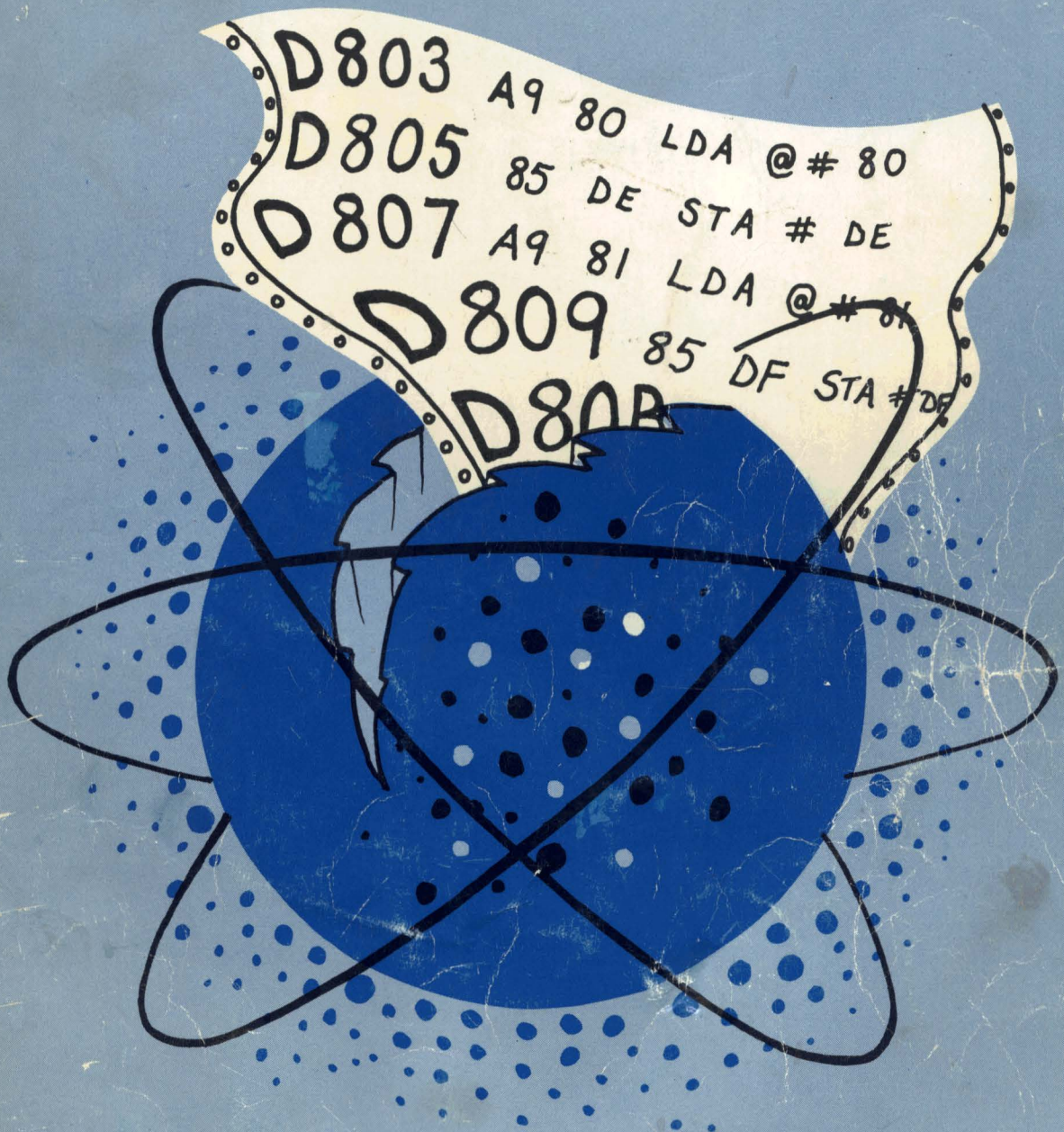
SPLITTING THE ATOM



THE ACORN RECOMMENDED ADVANCED USER MANUAL

SPLITTING THE ATOM

A MANUAL FOR INFORMED USERS

BY

J. R. STEVENSON and J. C. ROCKETT

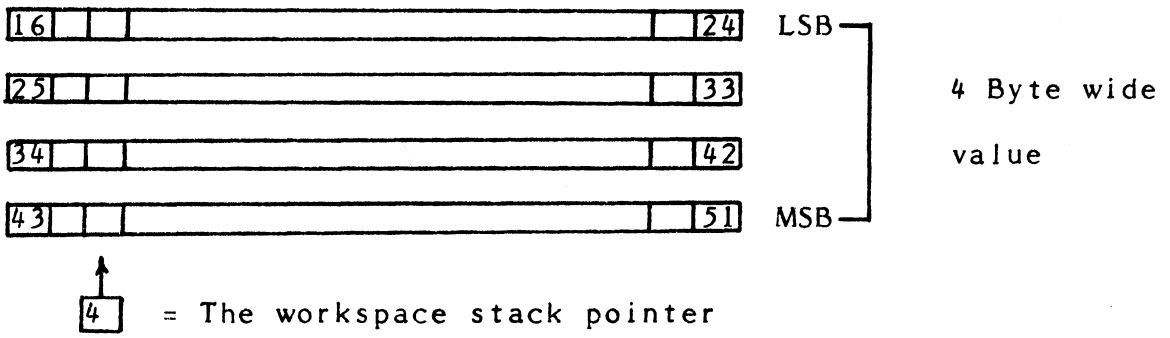
TABLE OF CONTENTS

CHAPTER 1
OPERATION OF THE WORKSPACE AND OTHER STACKS

1. The Workspace Stack

A four byte wide workspace stack is used by the ATOM to perform arithmetic functions and temporary storage of data being manipulated. This stack is best explained by comparison with the 6502 machine code stack, as the principle is very similar.

The page zero locations 16 through 51 inclusive are reserved for the workspace stack, but since the information being stored is up to four bytes wide (that is, a BASIC integer range of about + 2*10↑9) this area is split up into four parts:



Just as the 6502 uses a stack from 1FF thru 180 and points to the next free location in it by the stack pointer register S , the workspace stack also requires a pointer, and this is kept in location 4, as shown above.

In the case of the 6502 stack, the pushing and pulling of the numbers on the stack automatically changes S, the stack pointer, so that it points to the next free location. With the workspace stack the equivalent operation must be done by the software, by incrementing or decrementing the contents of 4 as needed.

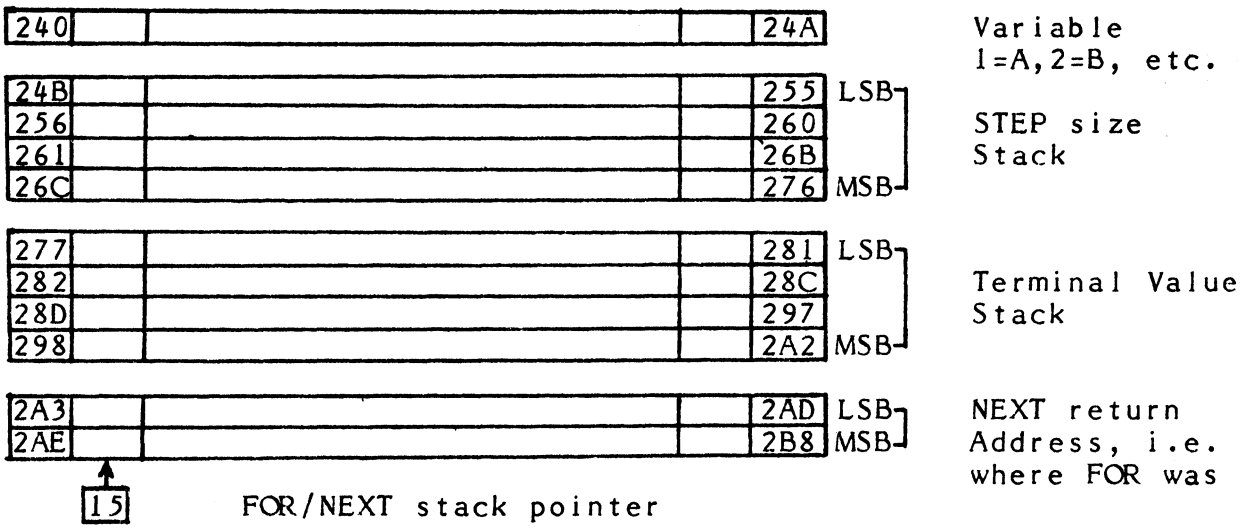
Many references are made in this book to routines which read or write values to the workspace stack, and may be used fairly freely by those writing machine code routines. One example is given below. It is extracted from the ATOM ROM at C99D, and is part of a routine to copy a random number in location 8 thru B to the workspace stack.

```
C99D    LDY @ 8
        LDX #4
        LDA #0001,Y
        STA #25,X
        LDA #0002,Y
        STA #34,X
        LDA #0003,Y
        STA #43,X
        LDA #0000,Y
        STA #16,X
        INX
        STX #4
```

Note how the X register is loaded from location 4 and then used as an offset to point at the current workspace stack values 16,X; 25,X; etc. . Note also that having pushed this data on the workspace stack, the w/s stack pointer is incremented by INX ; STX #4 . This is directly equivalent to the machine code instruction PHA (push value on stack and change stack pointer S) except that the routine achieves this on a 4 Byte wide basis.

Machine code writers invoking existing ROM routines such as this should pay careful attention to the w/s stack pointer at 4, and always ensure that it stays inside the limits 0 thru E .

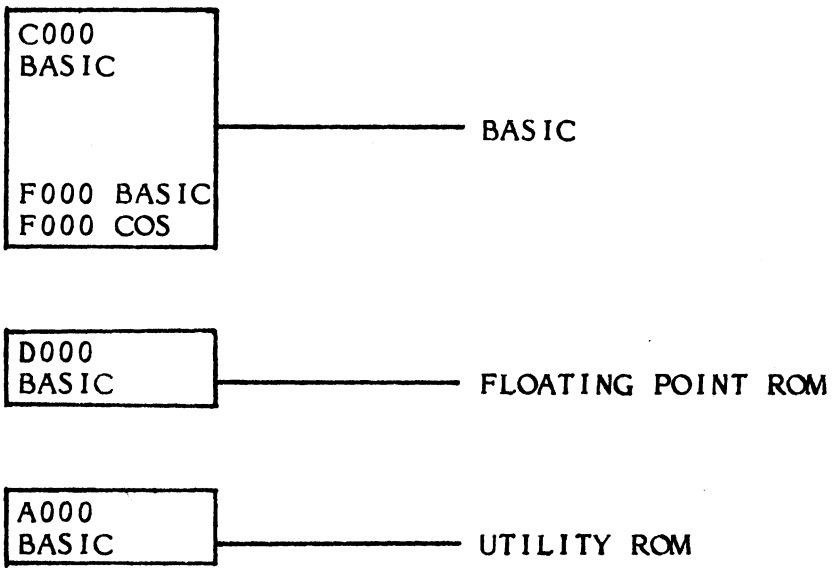
II. The FOR/NEXT Stacks



Each new FOR command increments the FOR/NEXT stack pointer to point at the data relevant to this loop, viz. , the location of the FOR, the terminal value, the STEP size,and the variable used.

A similar map can be drawn for DO/UNTIL and GOSUB/RETURN loops, though there are obvious differences. See Chapter 3 - RAM usage.

CHAPTER 2
THE STRUCTURE OF THE INTERPRETER



Programs are stored in memory as a series of strings, which in the expanded ATOM are normally begun at #2900. Address 2900 contains an 0D which means "start of program. Each line of the program consists of a two byte line number (stored as hex), followed by the actual ASCII code for what ever you typed in. At the end of each line is an 0D, and the end of program is marked by an FF (thus a program always ends in 0D FF). A program consisting only of 20 PRINT"HELLO";END would look like this if we did an ASCII Dump starting at 2900:

0D	00	14	P	R	I	N	T	"	H	E	L	L	O	"	;	E	N	D	0D	FF

P. &TOP would give 2915, since this is the next memory location after the FF at the end of the program.

Strings being interpreted, either in direct mode or as a program being run, are first checked by the C000 BASIC interpreter. If they are valid, a match with the word in the string is found in the ROM, and the appropriate routines are called for execution of the word.

If the C000 interpreter can't find a match for the string, then it passes control over to the F000 Basic interpreter. Again, valid matches are sought, and executed if one is found.

If the F000 interpreter can't resolve the string, then normally this would mean that an erroneous string is present, and an ERROR routine is called. However, before giving up all hope, a simple test is made which looks for the signature of a ROM at D000 (the FLT. PT. ROM), and if the ROM is present, then the string is passed over to it for interpretation.

By this means the ATOM can work with or without the FLT. PT. ROM installed, and when one is plugged in, the machine is able to detect that it is there.

Similarly, the FLT.PT. ROM contains a test that examines the UTILITY socket at A000, by testing the location A000 and A001 for 40 and BF respectively. If these are present, then interpretation is passed to the A000 ROM.

The COS commands are independent of the BASIC interpreter, and have their own interpreter at F8F0, accessed automatically by the leading asterisk (*) of all COS commands. The COS command interpreter is indirected by (OSCLI), which, since it is in RAM, allows user intervention, and so the possibility of adding extra words without the addition of a ROM. An illustration of this is given later by the HEX DUMP program.

Assume the following string is being interpreted:

P	R	I	N	T	A	;	P	R	I	N	T	B	OD

and that we are in the direct mode, so that this has been typed into the machine from the keyboard. The string is held in the direct mode input buffer at 100 onward. The keying of the carriage return (<CR>) puts an 0D at the end of the string as shown, and passes control over to the interpreter.

The interpreter uses a vector at 5,6 to point to the location of the string under scrutiny and so this vector is set to 100 from the direct mode, and a word match is sought. The interpreter works its way along the word by incrementing Y , so that (5),Y points to the character within the word being matched. Once the machine has resolved the entire command (PRINT in the case above) the vector (5) is consolidated by adding the Y register to it. Then Y is set to zero, so that in our case (5),Y is pointing at A in the PRINT A command. The interpreter goes on to find out what needs printing, but before execution checks that there is no rubbish behind the letter A, then executes the appropriate routines. Having executed the PRINT A, the vector at (5),Y is now pointing at the statement separator (semicolon), and the machine skips past this to execute the next command.

By this means the (5),Y pointer can range throughout the whole of the memory area. All the machine's BASIC interpreters use this vector, and before the value of Y has been spoiled by execution calls, its value is stored in ?3 .

CHAPTER 3
RAM USED BY THE OPERATING SYSTEM

ADDRESS		FUNCTION
00		Error number in BASIC
01-02		Line number in BASIC, 0 means Direct Mode (as MSB,LSB in Binary, not BCD)
03		BASIC text pointer offset
04		Workspace Stack pointer
05,06		BASIC text pointer:(5),3 points at character
07		COUNT value
08-0C		Random Number seed
0D,0E		TOP : points at top of BASIC text area
0F		Hexadecimal printer flag (positive=hex)
10,11		Pointer to BASIC error handler
12		BASIC text area MSB (page), normally #29
13		DO/UNTIL stack pointer
14		GOSUB/RETURN stack pointer
15		FOR/NEXT stack pointer
16-24	LSB	Integer Workspace stack
25-33		
34-42		
43-51	MSB	
23,24		DIM (free space) pointer
32,33		DATA pointer for DISATOM
52-6F		Arithmetic Workspace
70-7F		Floating Point Workspace (free if FP unused)
80-AF		FREE
B0-FF		COS workspace
C9		Title string of file to load from tape
DD		*FLOAD flag. Set if bit 7=1
DE,DF		Cursor position pointer (start of line)
E0		Horizontal cursor position 0-1F
E1		Cursor Mask, usually #80
E6		Page mode flag:neg.=OFF,else No. lines left
E7		Lock key flag. 0=inactive, #60=lock on
EA		noramlly 0. If not, then *NOMON engaged
FE		Character NOT sent thru VIA to printer
100-13F		Direct Mode input buffer
140-17F		BASIC input buffer and String operation area
180-1FF		Microprocessor Stack
200,201		NMIVEC -
202,203		BRKVEC C9D8
204,205		IRQVEC A000 , just RTI
206,207		COMVEC F8EF
208,209		WRCVEC FE52
20A,20B		RDCVEC FE94
20C,20D		LODVEC F96E
20E,20F		SAVVEC FAE5
210,211		RDRVEC C2AC , just BRK
212,213		STRVEC C2AC , just BRK
214,215		BGTVEC FBEE
216,217		BPTVEC FC7C
218,219		FNDVEC FC38

21A,21B		SHTVEC C278 , RTS(unless DOS present)
21C-23F		FREE
240-24A		Pointer to variable stack,FOR/NEXT,1=A,2=B,etc.
24B-255	LSB	
256-260		FOR/NEXT step size stack
261-26B		
26C-276	MSB	
277-281	LSB	FOR/NEXT terminal value stack
282-28C		
28D-297		
298-2A2	MSB	
2A3-2AD	LSB	FOR/NEXT return address stack
2AE-2B8	MSB	
2B9-2C3	LSB	DO/UNTIL return address stack
2C4-2CE	MSB	
2CF-2DC	LSB	GOSUB/RETURN return address stack
2DD-2EA	MSB	
2EB-305	LSB	Array pointer stack : 2EB,306= @@
306-320	MSB	2EC,307=AA etc.
321-33B	LSB	Simple Integer Variable stack
33C-356		321,33C,357,372 = @
357-371		322,33D,358,373 = A
372-38C	MSB	etc.
38D-3C0		Label address stack 38D,38E= [@];38F,390= [A] etc
3C1-3C4		Last plotted point (for line drawing)
3C5-3C9		Used by FPUT and FGET
3CA-3FC		FREE unless DOS used
3FD		Used by colour point plot
3FE-3FF		Point plot vector

2800- 2887 Floating point variables %@ to %Z .Each is 5 bytes wide, so 135 bytes used.

THE SIMPLE INTEGER VARIABLE STACK

Variable	LSB			MSB
@	321	33C	357	372
A	322	33D	358	373
B	323	33E	359	374
C	324	33F	35A	375
D	325	340	35B	376
E	326	341	35C	377
F	327	342	35D	378
G	328	343	35E	379
H	329	344	35F	37A
I	32A	345	360	37B
J	32B	346	361	37C
K	32C	347	362	37D
L	32D	348	363	37E
M	32E	349	364	37F
N	32F	34A	365	380
O	330	34B	366	381
P	331	34C	367	382
Q	332	34D	368	383
R	333	34E	369	384
S	334	34F	36A	385
T	335	350	36B	386
U	336	351	36C	387
V	337	352	36D	388
W	338	353	36E	389
X	339	354	36F	38A
Y	33A	355	370	38B
Z	33B	356	371	38C

THE ARRAY POINTER STACK

ARRAY POINTER	LSB	MSB
@@	2EB	306
AA	2EC	307
BB	2ED	308
CC	2EE	309
DD	2EF	30A
EE	2F0	30B
FF	2F1	30C
GG	2F2	20D
HH	2F3	30E
II	2F4	30F
JJ	2F5	310
KK	2F6	311
LL	2F7	312
MM	2F8	313
NN	2F9	314
OO	2FA	315
PP	2FB	316
QQ	2FC	317
RR	2FD	318
SS	2FE	319
TT	2FF	31A
UU	300	31B
VV	301	31C
WW	302	31D
XX	303	31E
YY	304	31F
ZZ	305	320

THE LABEL ADDRESS STACK

Label	Address LSB	MSB
A	38D	38E
B	38F	390
C	391	392
D	393	394
E	395	396
F	397	398
G	399	39A
H	39B	39C
I	39D	39E
J	39F	3A0
K	3A1	3A2
L	3A3	3A4
M	3A5	3A6
N	3A7	3A8
O	3A9	3AA
P	3AB	3AC
Q	3AD	3AE
R	3AF	3B0
S	3B1	3B2
T	3B3	3B4
U	3B5	3B6
V	3B7	3B8
W	3B9	3BA
X	3BB	3BC
Y	3BD	3BE
Z	3BF	3C0

CHAPTER 4
ADDRESSES OF ROUTINES

C000 to C22B : All this is Data for the Interpreter. The interpreter looks in this area for a match for the first letter of the word it is looking at. It then jumps in the table to an area containing all words beginning with that first letter, and looks at the second letter. It thus performs a Tree Search of the BASIC words stored in this area.

C22C to C278 : A Subroutine, the Function Interpreter. This area evaluates the Value of any arbitrarily complex function pointed to by (5),Y ,finds its value, then stores the results on the workspace stack (SEE C3C8).

C279 to C2AC : Looks up the "meaning" of commands. If there is no match in the Tree Table at C000 it hands over to those kept at F000, if not there then D000, if not there then A000, and if not there then error. The tree search is very quick and it seems that this is the original ACORN Interpreter. The later additions at F000 and elsewhere are total linear searches and slower.

C2AD : Executes the command NEW .This is available to you, but exits back to direct mode. Enter routine at C2B2.

C2B2 to C31A : Execution of the <BREAK> key comes to here from about FF94. It puts 0D FF into 2900,2901,sets @=8, then hands over to the CDOF Keyboard Input routines. This routine is entered at C2CF after a command execution, and at the end of a BASIC program.It carries on thus:
C2D5-set vector at (5) to =100
C2DC-set line number =0
C2E0-set BRK vector to C9D8
C2EA-set error pointer to C9E7
C2F2-set stack pointer to FF
C2F5-zero the temporary X and Y stores
C2FB-set nesting level of all GOSUB, FOR, DO loops to 0.
C301-set all labels to 0
C309-asks"is this a line number";C313-YES;C316-NO .
This area can be entered anywhere if there is a command in the Input buffer.

C31B to C333 : Executes the command THEN.

C325 to C333 : Executes the command LET.

C334 to C33E : Executes the command PRINT.

C33F to C3B1 : PRINT in Hexadecimal.Entry at C349 prints the workspace stack in HEX. See example, CHAPTER 6.

C3B2 to C3C7 : Executes the command LINK.

C3C8 to C3E4 : A Subroutine to evaluate an arbitrarily complex function pointed at by (5),Y and store the computed value on the workspace stack. On return the current value of the workspace stack pointer is where the answer is stored. The value is also copied to 52,53,54,55. On return the (5),Y pointer has been consolidated, i.e. (5),0 points at the last character in the string interpreted.

C3E5 to C3ED : Deal with assignments such as "X=..." .

C3EE to C405 : Deal with the command ! (quad-POKE).

C406 to C40D : Deal with the command ? (POKE).

C40F to C423 : Executes the cassette operating commands starting with *. The routine strips off the * and copies the remainder of the (5),Y string, up to a <CR>, into the direct mode input buffer at 100. A subroutine is then called which passes interpretation over to COS by JSR FFF7 (indirected by (OSCLI)).

C424 to C433 : Checks to see if Floating Point ROM is in. The lowest two bytes of the FP ROM are a signature (AA 55), and this routine tests for these values at D000 and D001, then returns with the carry clear if the ROM is not there. The routine is called from C550, where the machine is deciding whether to pass a string it can't understand to the interpreter contained in the floating point ROM, or to give up and signal an error.

C434 to C464 : The Interpreter "Pre-Test" subroutine whose effect is to take the character pointed to by 5,Y (where Y=?3) and if this character is an alphabetic it converts it to the number 1-26, then places it at 16,X (where X=?4), then ?4 is incremented. If the next character is non-alphabetic the carry is cleared before return (eg the command P.), but if the next character is alphabetic (eg the command LINK) then the carry flag is set before return. This routine therefore enables the machine to rapidly execute abbreviated commands, since it need not read the entire command.

C465 to C4DD : A valuable Subroutine to read a decimal string. It reads a string pointed to by (5),Y (where Y=?3) as ASCII decimal characters, and converts the decimal numeric value to a binary value, then stores it in the 16,X workspace stack (where X=?4). ?4 is incrememted so the workspace stack can continue. If the first non-space character is not a number, then BRK is executed. Spoils A,X, and Y registers.

C4E4 to C50B : A Subroutine used as the interpreters post-test. It checks that (5),Y (where Y=?3) is pointing at a carriage return or a semi-colon, or spaces leading thereto. If not, then executes BRK.
C4F6- consolidates (5) by (5)=(5)+Y and Y=1 .
C504-checks to see if the ESC key is depressed. If not then RTS, otherwise it jumps to direct mode and executes the escape code.

C50C to C546 : A Subroutine which copies a new line number to 1,2 and checks if the line is labelled. If there is a label this routine passes the current text-position pointer at (5),Y to the label store (LSB 38D,X MSB 38E,X).

C54A to C565 : Execution of a statement pointed at by 5,Y. It also checks for the Floating Point ROM, and if it is there this routine jumps indirectly to (D002). If not then it jumps to default handling. C55B is the best place to return to BASIC after a m/c routine, whether in direct or program mode.

C566 to C574 : Executes the IF command. C566 calls C70C, which is a truth test that puts a zero on the workspace stack (at 16,X where X=?4) if false.

C575 to C588 : Executes the REM command by incrementing (5),Y until a <CR> is encountered.

C589 to C607 : A Subroutine which prints the lowest level of the workspace stack (ie 16,25,34,43) as a signed decimal number in field size @ . A,X,Y are spoiled.

C608 to C62D : Data tables for the above routine.

C62E to C660 : A Subroutine which uses the vector at (58) to search through a BASIC program looking for a line number match, or for a line number greater then that recently inputted. The inputted line number is assumed to be on the 16,X workspace stack one level down from the workspace stack pointer(?4). The routine returns with (58),Y pointing at the character immediately after the matching line number, and the carry is clear. If the carry is set, then no line number match was found.

C661 to C688 : A Subroutine called by the C80B multiply routine.

C689 to C6D9 : A Subroutine as C661.

C70C to C713 : A Subroutine which is the truth test used by the IF and UNTIL commands. It evaluates an arbitrarily complex statement or equation [pointed at by (5),(?3)] and places zero on the workspace stack at 16,X if false.

C714 to C721 : The logical AND truth test (you use C70C).

C722 to C72B : The logical OR truth test (you use C70C).

C731 to C79C : String comparison test use by the above truth test

C79D to C7B6 : deals with adding together two adjacent 4-byte numbers on the workspace stack, viz.:

```
14       14       15
23   =   23   +   24
32       32       33
41 ,X    41 ,X    42 ,X
```

C7B7 to C7D2 : As above, but subtraction.

C7D3 to C7ED : As above, but bitwise logical OR.

C7EF to C80A : As above, but EOR.

C80B to C87A : Deals with multiplication.

C87B to C89B : Similar to C79D, but bitwise AND based on 16,X.

C8BC to C8DB : As for C3C8, but increments w/s pointer, and does not copy the result to 52,53,etc.

C8BC to C8DB : A Subroutine which deals with the minus sign. Entering at C8C4 negates the current slot on the workspace stack cf:

```
15        15
24   =0-  24
33        33
42 ,X     42,X
```

C8DC to C8F7 : A Subroutine to deal with variable assignments. Entering at C8E3 will copy any simple variable pointed at by Y (Y=1 is A,Y=2 is B etc.) to the current slot on the workspace stack (as given by ?4). See eg program at back. This is the opposite of CA2F.

C8F8 to C901 : Deals with numeric assignments.

C902 to C909 : Executes the ABS function. This can be used by pointing at the item you want ABSed with 5,Y. The result is placed on the workspace stack.

C90A to C943 : Deals with the # sign (HEX number sign).

C944 to C94B : Deals with ((leftbracket).

C94C to C95E : Deals with ? as a PEEK function.

C95F to C972 : Deals with ! as a quad-PEEK function.

C973 to C985 : A Subroutine that reads TOP value at vector (D,E) onto the current workspace stack, and increments the workspace stack pointer.

C97A to C985 : A Subroutine which reads the current COUNT value (?7) to the current slot of the workspace stack.

C986 to C9BC : A Subroutine to execute RND. It generates a new random number at 8 to C, copies it to the current slot of the workspace stack, and increments the workspace stack pointer (?4), which you MUST reset. This can be used by you to generate random numbers in a machine code program (see example, CHAPTER 6).

C9BD to C9D1 : Executes the LEN function.

C9D2 to C9D7 : Deals with the CH operator.

C9D8 to C9E6 : BRK handler. When the 6502 executes a BRK instruction it is directed here through the vector in 202,203, normally set by the operating system immediately before executing a Direct Mode command. Its effect is to point the BASIC interpreter text pointer at the vector 10,11, normally C9E7. Exits to direct mode.

C9E7 to CA23 : BASIC error handler. This is the BASIC statement executed whenever a BRK command is executed, normally meaning an error of some type. It says:

```
@ =1;P.$6$7'"ERROR "?0;
@ =8;IF ?1?2 P." LINE"! 1 & #FFFF;P.';E.
```

It uses ?0 as the error number and ! 1 & #FFFF as the line number. If the line number is zero this is inferred as a direct mode error, and no line number displayed. Usable by pointing 5,Y at C9E7, then JMP C55B.

CA24 to CA2B : Routine which calls the floating point ROM installation check at C424 and either Breaks if not installed, or jumps indirect (D004) if ROM is there.

CA2F to CA4B : A Subroutine, which copies the last value on the workspace stack to the integer variable pointed at by the Y register (Y=1 for A,Y=2 for B, etc.). The workspace stack pointer (?4) is decremented TWICE. This is the opposite of C8DC.

CA4C to CA4E : Subroutine, which increments the value of COUNT (location 7) and then prints the contents of the accumulator as an ASCII character.

CA51 to CACC : Execute LIST. The value of the X register must be 0 on entry, and the routine exits to direct mode.

CACD to CB56 : Execute NEXT. CAD0 checks the value of the FOR/NEXT stack pointer(?15) and causes BRK if 0, since this must mean no FOR/NEXT has been set.
CAE5- adds the STEP size to the variable.
CB16- checks if the control variable value has reached the final value.
CB45- moves the text pointer back to the statement after the corresponding FOR statement.

CB57 to CB80 : Execute FOR. CB5F sets the control variable equal to its first value.
CB65- checks that the FOR/NEXT stack pointer has not exceeded the allowable range.
CB6C- saves a default STEP value of 1 .

CB81 to CBA1 : Execute TO. CB89 saves the terminal value of the FOR control variable.

CBA2 to CBD1 : Execute STEP. CBAA saves the STEP size.
CBC3- saves the FOR/NEXT return address, and increments the FOR/NEXT stack pointer at 15.

CBD2 to CBEB : Execute "GOSUB". CBD8 tests the GOSUB stack pointer value (14) and yields an error if too many.
CBDE- saves the RETURN address, and increments the GOSUB stack pointer.

CBEC to CC04 : Executes RETURN. CBEF tests the GOSUB stack pointer (14), and if 0 gives the RETURN WITHOUT GOSUB error.
CBF5- pulls the return address from the data stack into the text pointer at 5.

CC05 to CC1C : Executes GOTO.

CC1F to CC80 : Subroutine, called by GOTO and GOSUB. It searches for an inputted line number or matching label. A successful search results in the line number being copied to location 1,2. If the label address is already known this is copied to 58,59. Otherwise the label is searched for and then stored in the label store as well as being copied to 58,59.

CC81 to CCD1 : Execute INPUT. CC8E is the entry point for a numeric variable INPUT, and CCB6 for a string variable. Both entries call the BASIC input routine at CD09 (q.v.); the inputted data is then copied or read from the string input buffer at 140 onwards (see e.g. prog. at back).

CCD2 to CCEF : Execute UNTIL. CCD2 calls the routine at C70C (the truth tester).
CCD5- checks for a zero value of the DO/UNTIL stack pointer at 13 .If zero, this is an UNTIL with no DO error.
CCE5- pulls the corresponding return address from data.

CCF0 to CD08 : Execute DO. CCF0 checks the value of the DO/UNTIL stack pointer at 13 for range, and causes an error if out of range (too many DO/UNTIL loops).
CCFA- saves the DO/UNTIL return address.

CD09 to CD58 : A very useful Subroutine, to execute inputs. Entry at CD09 prints a '?' on the screen and then waits for keypresses. Entries are stored in the string input buffer at 140 onwards, and full editing is allowed. The routine returns when <CR> key is pressed, with the Y register pointing at the last character inputted. Entry at CD0B prints the contents of the accumulator as an ASCII character (normally the > prompt sign), and then stores keypresses in the Direct Mode input buffer at 100 onwards. The value of COUNT (?7) is set to 0 on return (see e.g. program at back).

CD98 to CDBB : Execute END. This effectively sets TOP (?0D) and jumps to direct mode.
CD9B- set TOP=?12 (start of text area).
CDA5- using TOP as a vector ,find a carriage return followed by a negative number, indicating end of program.

CDBC to CDC8 : A Subroutine called by END which executes:
TOP=TOP+Y register;Y register=1 .

CDC9 to CE82 : Routine to enter a BASIC program line into the text area. On entry 16 and 25 contain the line number being entered.
CE3E- A RAM test to see if there is enough to enter it.

CE83 to CE92 : Continuation of the RUN command (see F141). It sets the text pointer at 5 equal to start of text (normally 2900) and then jumps to the interpreter at C55B .

CE93 to CEA0 : A Subroutine called by the "?" command at C406.

CEA1 to CEAD : A Subroutine which executes:
(58)=(58)+Y register; Y register=1

CEB1 to CEB5 : A Subroutine that checks for a dollar sign or quotes at the location pointed to by 5,(?3). If true, it returns with 5,(?3) pointed to the character after, if false, BRK.

CEBF to CEEC : A Subroutine. It copies a string in quotes pointed at by (5),Y into the string input buffer at 140 onwards. The quotation signs are removed. Enter at CEC2.

CEED to CEF9 : Execute LOAD command. CEF4 calls the 'Load a File' routine at FFE0. All this is well documented in the ATOM manual.

CEFA to CF09 : A Subroutine called by LOAD and SAVE. It reads the program title into the string input buffer at 140, sets the vector (54) equal to the start of the BASIC text area (normally 2900), and then returns.

CF0A to CF27 : Execute SAVE command.
CF0A- calls above subroutine to set (54)=start of text.
CF0D- sets (58)=start of text.
CF11- sets (5A)=TOP
CF19- sets (56)=RUN address of C2B2.
CF22- calls 'Save a File' routine at FFDD.

CF28 to CF5A : Various uninteresting subroutines used by GET and PUT- see routines that follow.

CF5B to CF65 : A Subroutine to execute BGET. It reads a value from tape/disc to the workspace stack LSB and sets the other bytes to zero.

CF66 to CF7A : A Subroutine to execute the GET command. It reads four bytes from tape/disc to the workspace stack.

CF8F to CF94 : Execute BPUT command.

CF95 to CFB3 : A Subroutine to execute PUT.

CFA6 to CFB3 : A Subroutine to execute FIN.

CFA7 to CFB3 : A Subroutine to execute FOUT.

CFC5 to CFE2 : Execute SPUT command.

CFE3 to CFFF : execute SGET command.

The above GET and PUT routines use 5,Y to point at the data after the command.

F000 ROUTINES

F000 to F02D : Command word table and action addresses. Includes PLOT,MOVE,DRAW,CLEAR,DIM,OLD,WAIT, and [.

F02E to F04A : An array pre-test, looks for two consecutive characters being the same, thus identifying an array.

F04B to F082 : Interpreter for the above command words. Jumps to the appropriate action addresses.

F08B to F0AD : A Subroutine called by F02E to pull the array start address from the table of array addresses (as LSB=2EB,Y and MSB=306,Y) and places it on the workspace stack.

F0AE to F140 : Executes DIM command as follows :
F0AE- Causes error 216 if in direct mode.
F0B9- Simple string dimension:set simple variable values (lower 2 bytes) to next free RAM space, and points DIM vector at(23)to the next available space.
F0D7- set up array dimensions. Sets the appropriate array variable pointer (see F08B), and points DIM vector to next available space.
F119- check that DIM vector has not exeeded avialable RAM, and cause error 30 if it has.
F131- take action on additional items separated by commas in the same DIM statement.

F141 to F14B : Executes the RUN command. Sets DIM vector at (23) equal to TOP, then jumps to CE83. This is the correct GO address for BASIC programs that use a DIM statement. CE86 may also be used if there are no DIM commands.

F14C to F154 : Executes the WAIT command (uses FE66).

F155 to F290 : Assembler data and look-up tables.

F291 to F29B : A Subroutine to fetch the next non-space character in the BASIC statement being interpreted. It uses 5,(?3) as a pointer, and returns with ?3 pointing at the first non-space character.

F2A1 to F375 : Executes the "[" command (start assembler).
F2A3- deals with "]"
F32E- deal with assembler labels.
F360- deal with assembler REMs (/).
F36B- deal with statement separator (;).

F376 to F37D : A Subroutine to print the contents of the accumulator as two hex characters followed by a space. Used by the assembler listing display.

F37E to F38D : Byte-printing routine called by F376 above.

F38E to F530 : Various routines used by the assembler.
F399- separate labels, separaters(;), and REMs (/).
F3F2- separate immediate(@),indirect (()), and accumulator mnemonics.
F454- act on immediate mode (@).
F462- act on indirect mode (()).
F49B- act on accumulator commands (e.g. ROL A).
F511- print "Out of Range".
F514- the string "Out of Range"

F531 to F541 : Carries out the OLD command.Exits to END at CD9B.

F542 to F641 : Carries out MOVE,DRAW,and PLOT commands.
F542- entry point for MOVE.
F546- entry point for DRAW.
F54E- entry point for PLOT.

F644 to F67A : Subroutines used by MOVE,DRAW, and PLOT.
F668- decrement the vector (5A),X .
F671- increment the vector (5A),X .
F678- point plot subroutine (JMP(3FE)). 3FE/3FF depends on the mode set by the CLEAR command (see below).

F67B to F6CE : Carries out the CLEAR command. This sets up the word at B000 for the CRT controller, and places the appropriate point plot routine address in 3FE/3FF.

F6C2 to F6CF : Carries out CLEAR 0 .

F6CF to F6E1 : Graphics mode control data, including appropriate clear mode and point plot routine addresses, and CRT controller words for B000 (port control from PIA).

F6E2 to F7C8 : Point PLOT subroutines use by MOVE,DRAW,PLOT.
It requires the X Co-ordinate in 5A,5B ; the Y Co-ordinate in 5C,5D ; 5E=0 clears point, 5E=1 sets the point and 5E=2 inverts the point. Entry points are:

MODE	ADDRESS
0	F6E2
1	F73B
2	F754
3	F76D
4	F7AA

F7C9 to F7D0 : Data used by point plot routines at F6E2 et.al. .

F7D1 to F7EB : A Subroutine that is very useful for printing from your own machine code program. When this routine is called, all bytes after the call are considered to be ASCII code, which is outputted to the screen. The routine will terminate back to your m/c program when it encounters a negative number (NOP is a good one).See example of use in CHAPTER 6.

F7EC to F817 : Subroutines to print the hex value of words (4 bytes), vectors (2 bytes) and single bytes. On return X is spoiled, but A and Y preserved.
F7EE-print in hex a word in order X+1,X,X+3,X+2.
F7F1-print in hex a vector (X+1,X).
F7FA-print byte in accumulator plus a space.
F802-print in hex the byte in the accumulator.

F818 to F84E : A Subroutine (use by *LOAD,*SAVE etc.), which copies a string enclosed in quotes in the 100 input buffer to the string area starting at 140. Y should point to the beginning of the input string. X,Y, and the accumlulator are spoiled.

F86C to F874 : Print "NAME" then BRK.

F875 to F87D : A Subroutine to fetch the next non-space character from the direct mode input buffer at 100,Y . On return, Y points to the character fetched.

F87E to F892 : A Subroutine which converts the value in the accumulator from a valid ASCII hexadecimal character to its hexadecimal value. If the contents of the accumulator was not a valid ASCII hex character the routine returns with the accumulator unchanged, and the carry flag set. Otherwise, the accumulator contains the true hex value and the carry flag is clear.

F893 to F8BD : A Subroutine which reads the ASCII hexadecimal value in the direct mode input buffer at 100,Y as a vector (two bytes or 4 characters) to the location pointed to by X on entry to the routine. e.g. :

Y=position of the 1st character in the buffer, lets say it points at the A of A147.
X= #80
After JSR F893, then 80,81=A147. If the first character in the buffer was invalid, then the zero flag is set on return.

F8BE to F8ED : Table of *COS reserved words and their action addresses.These are: CAT,LOAD,SAVE,RUN,MON,NOMON,FLOAD, and DOS.

F8EF to F925 : *COS interpreter subroutine called by OSCLI. It looks for a match between a word in the direct mode input buffer at 100,Y and the reserved words starting at F8BE. It jumps to the correct action address if a match is found.

F926 to F92E : Default routine for unknown *COS command, which prints "COM" and then ERROR 48.

F955 to F96D : Executes the *FLOAD and *LOAD commands. F955=*FLOAD , and F958=*LOAD. The routine exits via (20C) , the LODVEC, which is normally set to F96E.

F96E to F9A1 : A Subroutine which loads a file. This is normally called by JSR FFE0 (OSLOAD-pointed to by [20C]). X must point at zero page vectors as follows: 0,X 1,X=file name string ; 2,X 3,X=first data to be put here ; if bit 7 of 4,X is 0 the file's own start address is used.

F99A- print a series of spaces by INY until Y=0F, so up to 15 spaces can be printed (note-it's easier to use CA46 and monitor ?7).

F9A2 to FA07 : A Subroutine called by the F96E routine.

FA08 to FA18 : A Subroutine which increments a vector (2 bytes) in page zero pointed at by X (X,X+1),and each time does a CMP with the vector pointed at by X+2,X+3. It returns with the zero flag set if the vectors are equal,otherwise clear.

FA19 to FA1F : Executes the *MON and *NOMON commands.
FA19=*NOMON, and FA1A=*MON

FA20 to FA29 : Executes the *RUN command.

FA2A to FA64 : Executes the *CAT command.

FA65 to FA6A : A Subroutine that calls the routine at F893. If the data read by F893 was invalid then this routine prints "MON?" followed by a break.

FA76 to FA85 : A Subroutine to check that there is no rubbish after a valid * command. Only a carriage return or spaces leading to a carriage return are allowed. Otherwise it prints "MON?" followed by a break.

FA86 to FABA : Saves an unnamed file. Called by FAE5.

FABB to FAE4 : Executes the *SAVE command. This routine calls the operating system save-file routine pointed at by (20E), which normally contains FAE5.

FAE5 to FB3A : Save file routine normally called by OSSAVE routines. Enter with X pointing at a table of addresses in page zero as follows:

0,X 1,X	file name string
2,X 3,X	reload address
4,X 5,X	execution address
6,X 7,X	first byte to be saved
8,X 9,X	last byte+1 to be saved

FB3B to FB89 : Routines called by the save-file routine which commit the file to tape. Useful parts are :
FB7D- wait 2 seconds.
FB81- wait 0.5 seconds.
FB83- wait X/60 seconds.
FB8C- wait 0.1 seconds.
X=0 on return from these routines.

FBEE to FC2A : A Subroutine to get a byte from tape. This routine is indirected by (214), normally called by JSR OSBGET (FFD4), and is designed to act at 300 baud. The routine reads individual bytes from the tape and is called by the LOAD routines, and by BGET, SGET, etc.. The byte fetched is passed back in the accumulator, the X and Y registers are preserved. The accumulator value is also added to the check sum kept in location hex DC.

FC38 to FC7B : A Subroutine used by COS commands to write PLAY,RECORD, or REWIND TAPE, then wait for a key to be pressed before returning. Entry at FC38 with C=1 gives "RECORD TAPE", while C=0 gives "PLAY TAPE".Entry at FC40 gives "REWIND TAPE".
FC4F- message PLAY TAPE.
FC58- message RECORD TAPE.
FC63- message REWIND TAPE.
FC6D- message TAPE.
FC76- wait for keypress.

FC7C to FCBC : A Subroutine to put a byte to tape. This routine is indirected via (216), normally called by JSR OSBPUT (FFD1) , and operates at 300 baud.The routine is called by the SAVE and BPUT commands, and passes the value of the accumulator to tape. The X and Y registers are preserved. The accumulator is also added to the checksum total, kept in hex DC.
FC88- synchronise to 2.4 KHz edge.
FC92- output a logical 1.
FC9C- output a logical 0.

FCD8 to FCE9 : A Subroutine used by OSBPUT to synchronise the bits being output to 2.4 KHz. reference oscillator. Entry at FCD8 waits for the first occurence of a high-to-low transition on bit 7 of port C of the PIA (the 2.4 KHz reference). Entry at FCDA with the X register set to a number 0 to 7F counts that number of 2.4 KHz. transitions before returning. This can be used for timing since X=1 gives c. 400 microseconds, X=2 c. 800 usecs. , etc..

FCEA to FE51 : A Collection of subroutines associated with the print channnel OSWRCH, including execution of the control codes 0 thru 1F. Useful ones are given below.

FD0B- <CTRL> F (screen off).
FD11- <CTRL> U (screen on).
FD1A- <CTRL> G (bell).
FD1C- short bell.
FD40- move cursor to start of line without deletion.
FD44- invert character at current cursor position.
FD50- delete a character.
FD5C- backspace.
FD62- linefeed.
FD65- Invert character under the cursor. If the screen has previously been turned off(i.e. ?E0 < 0) then a CLEAR SCREEN is executed.
FD69- <CTRL> L (Clear,Home Up Left)
FD7D- <CTRL> ↑ (Home Up Left)
FD87- cursor up.
FD8D- <CTRL> N (Page Mode On).
FD92- <CTRL> O (Page Mode Off).

FDEC- Scroll-Screen Check, looks to see if the next character would cause a scroll, checks the page mode counter (?E6), and executes a scroll or waits for a keypress.
FE08- Scroll the Screen. Entry at FE0A with Y=40 will scroll all but the top line of the screen. Y=60 leaves the top two lines alone, etc..
FE22- delete all current line
FE24- blank Y+1 characters in current line.
FE26- fill Y+1 characters from current line onward with the character in the accummulator.
FE35- Check Next Cursor Position, called by Backspace and Delete to see if the cursor is at the beginning of a line or Home position.

FE52 to FE65 : Routine to print a character. This is indirected by (208), called by the OSWRCH at FFF4.
FE52- Pass character to VIA printer, and execute.
FE55- Print character on screen or execute any recognisible control codes. X and Y registers preserved.

FE66 to FE70 : A Subroutine to synchronise to CRT Field Flyback, used to write on the screen without generating noise. Can be used as a timer.
FE66- wait until the start of the next field flyback, even if already in flyback.
FE6B- return immediately if already in flyback, else wait until the next flyback. A,X,Y all preserved.

FE71 to FE93 : The Keyscan Subroutine called by OSRDCH (see below). Does not examine <CTRL>, <SHIFT>, <RPT>, or <BREAK>. It returns with the carry flag set if no key was pressed. If a key was pressed when this routine was called, the carry flag is cleared and the Y register holds the key pressed as its ASCII value minus hex 20.

FE94 to FECA : OSRDCH Subroutine. This routine waits for a key to be pressed and then returns with its ASCII value in the accumulator. Cursor and some other control codes are executed BEFORE returning.

FECB to FEFA : Data and Look-up tables for executing control codes.

FEFB to FF3E : A Subroutine called by OSWRCH to pass the value of the accumulator to the printer using the VIA. <CTRL> B and C enable or disable this routine respectively.
FF10- waits for handshake signal. (SEE Chapter 7).

FF3F to FF99 : RESET - the machine comes here after hitting <BREAK> or at switch-on, by picking up the reset address at FFFC (common to all 6502 microprocesssors)
FF3F- initialise page 2 vectors (204 and up).
FF4A- set stack pointer to FF.
FF53- set all array pointers to FFFF.
FF69- print message 'ACORN ATOM'
FF7C- test for RAM at 2900, and set text pointer to default values if appropriate.

FF9A to FFB1 : Data used by the RESET routine to initialise page two vectors.

FFB2 to FFBD : IRQ handler. Determines the kind of IRQ (true interupt or BRK), and executes it.

FFC0 to FFC6 : Executes BRK.

FFC7 to FFCA : Executes non-maskable interupt (NMI).

FFCB to FFF9 : Jump tables for major operating system routines.

ADDRESS	JUMP(x)	CODE	NORMAL VALUE
FFCB	021A	OSSHUT	C278
FFCE	0218	OSFIND	FC38
FFD1	0216	OSBPUT	FC7C
FFD4	0214	OSBGET	FBEE
FFD7	0212	RDRVEC	C2AC (BRK)
FFDA	0210	STRVEC	C2CA -"-
FFDD	020E	OSSAVE	FAE5
FFE0	020C	OSLOAD	F96E
FFE3	020A	OSRDCH	FE94
FFE6		OSECHO	FE94 THEN FE52
FFE9		OSASCI	0D CAUSES CR,LF
FFED		OSCRLF	CAUSES CR,LF
FFF4	0208	OSWRCH	FE52
FFF7		OSCLI	F8EF
FFFA		NMI	FFC7
FFFC		RESET	FF3F
FFFE		IRQ/BRK	FFB2

DISASSEMBLY OF C000 AND F000

```
C22C JSR #CF3E
C22F STY #0F
C231 LDX@#ED
C233 LDY #03
C235 DEY
C236 INY
C237 LDA(#05),Y
C239 CMP@#20
C23B BEQ #C236
C23D STY #5E
C23F STA #52
C241 INX
C242 LDA #BFFF,X
C245 BMI #C26B
C247 CMP #52
C249 BNE #C241
C24B LDA #C0EE,X
C24E TAX
C24F INX
C250 INY
C251 LDA #BFFF,X
C254 BMI #C26B
C256 CMP(#05),Y
C258 BEQ #C24F
C25A LDA(#05),Y
C25C CMP@#2E ..
C25E BEQ #C264
C260 LDY #5E
C262 BPL #C24B
C264 INX
C265 LDA #BFFF,X
C268 BPL #C264
C26A INY
C26B CMP@#FE
C26D BCS #C2AA
C26F STA #53
C271 LDA #C0EE,X
C274 BCC #C29F
C276 LDX #04
C278 RTS
C279 LDX@#0E
C27B LDY #03
C27D DEY
C27E INY
C27F LDA(#05),Y
C281 CMP@#20
C283 BEQ #C27E
C285 CMP #C1DD,X
C288 BEQ #C296
C28A STA #52
C28C INX
C28D LDA #C1DD,X
C290 BMI #C2A8
C292 CMP #52
C294 BNE #C28C
C296 LDA #C212,X
C299 STA #53
C29B LDA #C1F8,X
C29B LDA #C1F8,X
C29E INY
C29F STA #52
C2A1 STY #03
C2A3 LDX #04
C2A5 JMP(#0052)
C2A8 CMP@#FE
C2AA BEQ #C276
C2AC BRK
C2AD JSR #C4E4
C2B0 BNE #C2B6
C2B2 LDA@#29 .)
C2B4 STA #12
C2B6 LDA@#0D
C2B8 LDY #12
C2BA STY #0E
C2BC LDY@#00
C2BE STY #0D
C2C0 STA(#0D),Y
C2C2 LDA@#FF
C2C4 INY
C2C5 STA(#0D),Y
C2C7 INY
C2C8 STY #0D
C2CA LDA@#08
C2CC STA #0321
C2CF LDA@#3E .>
C2D1 CLD
C2D2 JSR #CD0F
C2D5 LDX@#01
C2D7 STX #06
C2D9 DEX
C2DA STX #05
C2DC STX #01
C2DE STX #02
C2E0 LDA@#D8
C2E2 STA #0202
C2E5 LDA@#C9
C2E7 STA #0203
C2EA LDA@#E7
C2EC STA #10
C2EE LDA@#C9
C2F0 STA #11
C2F2 LDX@#FF
C2F4 TXS
C2F5 LDA@#00
C2F7 STA #04
C2F9 STA #03
C2FB STA #15
C2FD STA #13
C2FF STA #14
C301 LDX@#34 .4
C303 STA #038C,X
C306 DEX
C307 BNE #C303
C309 JSR #C434
C30C BCS #C32F
C30E JSR #C46A
C30E JSR #C46A
C311 BCC #C316
C313 JMP #CDC9
C316 LDX@#7D .}
C318 JMP #C233
C31B JSR #C434
C31E BCS #C32F
C320 LDX@#7F
C322 JMP #C233
C325 JSR #C434
C328 BCS #C32F
C32A LDX@#10
C32C JMP #C27B
C32F LDX@#14
C331 JMP #C27B
C334 SEC
C335 ROR #0F
C337 JSR #C372
C33A LDX@#2E ..
C33C JMP #C233
C33F JSR #C78B
C342 JSR #C3CB
C345 LDA #0F
C347 BMI #C36A
C349 LDX@#00
C34B STX #27
C34D LDY@#00
C34F LDA #0052,Y
C352 PHA
C353 AND@#0F
C355 STA #45,X
C357 PLA
C358 LSR A
C359 LSR A
C35A LSR A
C35B LSR A
C35C INX
C35D STA #45,X
C35F INX
C360 INY
C361 CPY@#04
C363 BCC #C34F
C365 JSR #C5C8
C368 BMI #C337
C36A JSR #C589
C36D BMI #C337
C36F JSR #CD54
C372 LDX@#18
C374 JMP #C27B
C377 JSR #CA4C
C37A LDA(#05),Y
C37C INY
C37D CMP@#0D
C37F BEQ #C39D
C381 STY #03
C383 CMP@#22 ."
C385 BNE #C377
C387 LDA(#05),Y
C387 LDA(#05),Y
C389 CMP@#22 ."
C38B BNE #C372
C38D INY
C38E BCS #C377
C390 JSR #C78B
C393 JSR #C3CB
C396 ORA #54
C398 ORA #53
C39A BEQ #C3AA
C39C LDY@#00
C39E LDA(#52),Y
C3A0 CMP@#0D
C3A2 BEQ #C337
C3A4 JSR #CA4C
C3A7 INY
C3A8 BNE #C39E
C3AA LDA #52
C3AC JSR #CA4C
C3AF JMP #C337
C3B2 JSR #C3C8
C3B5 JSR #C4E4
C3B8 LDA #0322
C3BB LDX #0339
C3BE LDY #033A
C3C1 JSR #C2A5
C3C4 CLD
C3C5 JMP #C55B
C3C8 JSR #C8BC
C3CB LDY@#52 .R
C3CD DEX
C3CE STX #04
C3D0 LDA #16,X
C3D2 STA #0000,Y
C3D5 LDA #25,X
C3D7 STA #0001,Y
C3DA LDA #34,X
C3DC STA #0002,Y
C3DF LDA #43,X
C3E1 STA #0003,Y
C3E4 RTS
C3E5 JSR #C4E1
C3E8 JSR #CA2F
C3EB JMP #C55B
C3EE JSR #C8BC
C3F1 JSR #CE93
C3F4 LDA #26,X
C3F6 INY
C3F7 STA(#52),Y
C3F9 INY
C3FA LDA #35,X
C3FC STA(#52),Y
C3FE INY
C3FF LDA #44,X
C401 STA(#52),Y
C403 JMP #C55B
C406 JSR #C8BC
C409 JSR #CE93
```

```
C409 JSR #CE93
C40C JMP #C55B
C40F LDX@#00
C411 LDA(#05),Y
C413 STA #0100,X
C416 STY #03
C418 INY
C419 INX
C41A CMP@#0D
C41C BNE #C411
C41E JSR #FFF7
C421 JMP #C558
C424 LDA #D000
C427 CMP@#AA
C429 BNE #C463
C42B LSR A
C42C CMP #D001
C42F BNE #C463
C431 LDY #5E
C433 RTS
C434 LDY #03
C436 BPL #C43B
C438 INY
C439 STY #03
C43B LDA(#05),Y
C43D CMP@#20
C43F BEQ #C438
C441 CMP@#5B .[
C443 BCS #C463
C445 SBC@#3F .?
C447 BCC #C464
C449 LDX #04
C44B STA #16,X
C44D INY
C44E LDA(#05),Y
C450 CMP@#2E ..
C452 BEQ #C463
C454 CMP@#5B .[
C456 BCS #C45C
C458 CMP@#40 .@
C45A BCS #C463
C45C INX
C45D STX #04
C45F SEC
C460 STY #03
C462 RTS
C463 CLC
C464 RTS
C465 JSR #C434
C468 BCS #C425
C46A LDX@#00
C46C LDY #03
C46E STX #52
C470 STX #53
C472 STX #54
C474 STX #55
C476 DEY
C477 INY
C478 LDA(#05),Y
C478 LDA(#05),Y
C47A SEC
C47B SBC@#30 .0
C47D BMI #C4D3
C47F CMP@#0A
C481 BCS #C4D3
C483 LDX #53
C485 PHA
C486 LDA #55
C488 PHA
C489 LDA #54
C48B PHA
C48C LDA #52
C48E ASL A
C48F ROL #53
C491 ROL #54
C493 ROL #55
C495 BMI #C46B
C497 ASL A
C498 ROL #53
C49A ROL #54
C49C ROL #55
C49E BMI #C46B
C4A0 ADC #52
C4A2 STA #52
C4A4 TXA
C4A5 ADC #53
C4A7 STA #53
C4A9 PLA
C4AA ADC #54
C4AC STA #54
C4AE PLA
C4AF ADC #55
C4B1 ASL #52
C4B3 ROL #53
C4B5 ROL #54
C4B7 ROL A
C4B8 BMI #C46B
C4BA STA #55
C4BC PLA
C4BD ADC #52
C4BF STA #52
C4C1 BCC #C4CF
C4C3 INC #53
C4C5 BNE #C4CF
C4C7 INC #54
C4C9 BNE #C4CF
C4CB INC #55
C4CD BMI #C46B
C4CF LDX@#FF
C4D1 BNE #C477
C4D3 TXA
C4D4 BEQ #C463
C4D6 SEC
C4D7 STY #03
C4D9 LDY@#52 .R
C4DB JMP #C99F
C4DE JSR #C279
C4E1 JSR #C78B
C4E1 JSR #C78B
C4E4 LDY #03
C4E6 DEY
C4E7 INY
C4E8 LDA(#05),Y
C4EA CMP@#20
C4EC BEQ #C4E7
C4EE CMP@#3B .;
C4F0 BEQ #C4F6
C4F2 CMP@#0D
C4F4 BNE #C55C
C4F6 CLC
C4F7 TYA
C4F8 ADC #05
C4FA STA #05
C4FC BCC #C500
C4FE INC #06
C500 LDY@#01
C502 STY #03
C504 LDA #B001
C507 AND@#20
C509 BEQ #C547
C50B RTS
C50C JSR #C4E4
C50F DEY
C510 LDA(#05),Y
C512 CMP@#3B .;
C514 BEQ #C50B
C516 LDA #06
C518 CMP@#01
C51A BEQ #C596
C51C INY
C51D LDA(#05),Y
C51F BMI #C55C
C521 STA #02
C523 INY
C524 LDA(#05),Y
C526 STA #01
C528 INY
C529 LDA(#05),Y
C52B DEY
C52C CMP@#61 .a
C52E BCC #C4F7
C530 SBC@#61 .a
C532 CMP@#1B
C534 BCS #C4F6
C536 INY
C537 ASL A
C538 TAX
C539 JSR #C4F6
C53C LDA #05
C53E STA #038D,X
C541 LDA #06
C543 STA #038E,X
C546 RTS
C547 JMP #C2CF
C54A DEY
C54B JSR #C4F6
C54E BNE #C55B
C54E BNE #C55B
C550 JSR #C424
C553 BCC #C558
C555 JMP(#D002)
C558 JSR #C4E4
C55B LDY@#00
C55D LDA(#05),Y
C55F CMP@#3B .;
C561 BNE #C57D
C563 JMP #C31B
C566 JSR #C70C
C569 DEX
C56A STX #04
C56C LDA #16,X
C56E BEQ #C575
C570 LDX@#20
C572 JMP #C233
C575 LDA@#0D
C577 DEY
C578 INY
C579 CMP(#05),Y
C57B BNE #C578
C57D LDA #06
C57F CMP@#01
C581 BEQ #C547
C583 JSR #C51C
C586 JMP #C31B
C589 LDA #43
C58B STA #27
C58D BPL #C593
C58F INX
C590 JSR #C8C4
C593 LDX@#09
C595 LDA@#00
C597 STA #45,X
C599 SEC
C59A LDA #16
C59C SBC #C608,X
C59F PHA
C5A0 LDA #25
C5A2 SBC #C610,X
C5A5 PHA
C5A6 LDA #34
C5A8 SBC #C61A,X
C5AB TAY
C5AC LDA #43
C5AE SBC #C624,X
C5B1 BCC #C5C1
C5B3 STA #43
C5B5 STY #34
C5B7 PLA
C5B8 STA #25
C5BA PLA
C5BB STA #16
C5BD INC #45,X
C5BF BNE #C599
C5C1 PLA
C5C2 PLA
C5C3 DEX
```

```
C5C4 BPL #C595
C5C6 LDX@#ØA
C5C8 DEX
C5C9 BEQ #C5CF
C5CB LDA #45,X
C5CD BEQ #C5C8
C5CF STX #52
C5D1 BIT #27
C5D3 BPL #C5D7
C5D5 INC #52
C5D7 SEC
C5D8 LDA #Ø321
C5DB BEQ #C5DF
C5DD SBC@#Ø1
C5DF SBC #52
C5E1 BEQ #C5EE
C5E3 BCC #C5EE
C5E5 TAY
C5E6 LDA@#2Ø
C5E8 JSR #CA4C
C5EB DEY
C5EC BNE #C5E6
C5EE BIT #27
C5FØ BPL #C5F7
C5F2 LDA@#2D .-
C5F4 JSR #CA4C
C5F7 LDA #45,X
C5F9 CMP@#ØA
C5FB BCC #C5FF
C5FD ADC@#Ø6
C5FF ADC@#3Ø .Ø
C6Ø1 JSR #CA4C
C6Ø4 DEX
C6Ø5 BPL #C5F7
C6Ø7 RTS
  - data -
C62E DEC #Ø4
C63Ø LDX #Ø4
C632 LDY@#ØØ
C634 STY #58
C636 LDA #12
C638 STA #59
C63A DEY
C63B LDA@#ØD
C63D INY
C63E CMP(#58),Y
C64Ø BNE #C63D
C642 JSR #CEA1
C645 LDA(#58),Y
C647 INY
C648 CMP #25,X
C64A BCC #C63B
C64C BNE #C66Ø
C64E LDA(#58),Y
C65Ø CMP #16,X
C652 BCC #C63B
C654 BNE #C66Ø
C656 STA #Ø1
C658 LDA #25,X
C658 LDA #25,X
C65A STA #Ø2
C65C JSR #CEA1
C65F CLC
C66Ø RTS
C661 JSR #C8BC
C664 LDA #42,X
C666 EOR #41,X
C668 STA #52
C66A JSR #C9Ø5
C66D LDY@#53 .S
C66F JSR #C3CD
C672 LDA #42,X
C674 STA #43,X
C676 JSR #C9Ø7
C679 LDY@#57 .W
C67B JSR #C3CD
C67E LDY@#ØØ
C68Ø STY #5B
C682 STY #5C
C684 STY #5D
C686 STY #5E
C688 RTS
C689 JSR #C661
C68C LDA #54
C68E JSR #C7Ø5
C691 BEQ #C67F
C693 LDY@#2Ø
C695 DEY
C696 BEQ #C6D9
C698 ASL #57
C69A ROL #58
C69C ROL #59
C69E ROL #5A
C6AØ BPL #C695
C6A2 ROL #57
C6A4 ROL #58
C6A6 ROL #59
C6A8 ROL #5A
C6AA ROL #5B
C6AC ROL #5C
C6AE ROL #5D
C6BØ ROL #5E
C6B2 SEC
C6B3 LDA #5B
C6B5 SBC #53
C6B7 PHA
C6B8 LDA #5C
C6BA SBC #54
C6BC PHA
C6BD LDA #5D
C6BF SBC #55
C6C1 TAX
C6C2 LDA #5E
C6C4 SBC #56
C6C6 BCC #C6D4
C6C8 STA #5E
C6CA STX #5D
C6CC PLA
C6CC PLA
C6CD STA #5C
C6CF PLA
C6DØ STA #5B
C6D2 BCS #C6D6
C6D4 PLA
C6D5 PLA
C6D6 DEY
C6D7 BNE #C6A2
C6D9 RTS
C6DA JSR #C78B
C6DD DEX
C6DE STX #Ø4
C6EØ LDA #42,X
C6E2 EOR@#8Ø
C6E4 STA #52
C6E6 LDA #43,X
C6E8 EOR@#8Ø
C6EA STA #54
C6EC LDY@#ØØ
C6EE SEC
C6EF LDA #15,X
C6F1 SBC #16,X
C6F3 STA #53
C6F5 LDA #24,X
C6F7 SBC #25,X
C6F9 STA #55
C6FB LDA #33,X
C6FD SBC #34,X
C6FF STA #56
C7Ø1 LDA #52
C7Ø3 SBC #54
C7Ø5 ORA #53
C7Ø7 ORA #55
C7Ø9 ORA #56
C7ØB RTS
C7ØC JSR #C72C
C7ØF LDX@#43 .C
C711 JMP #C233
C714 JSR #C72C
C717 LDA #14,X
C719 AND #15,X
C71B STA #14,X
C71D DEC #Ø4
C71F JMP #C7ØF
C722 JSR #C72C
C725 LDA #14,X
C727 ORA #15,X
C729 JMP #C71B
C72C LDX@#46 .F
C72E JMP #C233
C731 JSR #C78B
C734 JSR #CEAE
C737 LDA #15,X
C739 STA #54
C73B LDA #24,X
C73D STA #55
C73F LDY@#FF
C741 INY
C741 INY
C742 LDA(#54),Y
C744 CMP(#52),Y
C746 BNE #C74F
C748 EOR@#ØD
C74A BNE #C741
C74C TAY
C74D BEQ #C76Ø
C74F LDY@#ØØ
C751 BEQ #C761
C753 JSR #C78B
C756 LDX@#ØØ
C758 JMP #C233
C75B JSR #C6DA
C75E BNE #C761
C76Ø INY
C761 STY #15,X
C763 RTS
C764 JSR #C6DA
C767 BEQ #C76Ø
C769 BCC #C76Ø
C76B BCS #C761
C76D JSR #C6DA
C77Ø BNE #C76Ø
C772 BEQ #C761
C774 JSR #C6DA
C777 BCC #C76Ø
C779 BCS #C761
C77B JSR #C6DA
C77E BCS #C76Ø
C78Ø BCC #C761
C782 JSR #C6DA
C785 BEQ #C761
C787 BCS #C76Ø
C789 BCC #C761
C78B JSR #C8ØB
C78E JMP #C795
C791 STA #41,X
C793 DEC #Ø4
C795 LDX@#ØØ
C797 JMP #C27B
C79A JSR #C8ØB
C79D CLC
C79E LDA #14,X
C7AØ ADC #15,X
C7A2 STA #14,X
C7A4 LDA #23,X
C7A6 ADC #24,X
C7A8 STA #23,X
C7AA LDA #32,X
C7AC ADC #33,X
C7AE STA #32,X
C7BØ LDA #41,X
C7B2 ADC #42,X
C7B4 JMP #C791
C7B7 JSR #C8ØB
C7BA LDA #14,X
C7BC SBC #15,X
C7BE STA #14,X
```

```
C7CØ LDA #23,X
C7C2 SBC #24,X
C7C4 STA #23,X
C7C6 LDA #32,X
C7C8 SBC #33,X
C7CA STA #32,X
C7CC LDA #41,X
C7CE SBC #42,X
C7DØ JMP #C791
C7D3 JSR #C8ØB
C7D6 LDA #14,X
C7D8 ORA #15,X
C7DA STA #14,X
C7DC LDA #23,X
C7DE ORA #24,X
C7EØ STA #23,X
C7E2 LDA #32,X
C7E4 ORA #33,X
C7E6 STA #32,X
C7E8 LDA #41,X
C7EA ORA #42,X
C7EC JMP #C791
C7EF JSR #C8ØB
C7F2 LDA #14,X
C7F4 EOR #15,X
C7F6 STA #14,X
C7F8 LDA #23,X
C7FA EOR #24,X
C7FC STA #23,X
C7FE LDA #32,X
C8ØØ EOR #33,X
C8Ø2 STA #32,X
C8Ø4 LDA #41,X
C8Ø6 EOR #42,X
C8Ø8 JMP #C791
C8ØB JSR #C8BC
C8ØE LDX@#Ø5
C81Ø JMP #C27B
C813 JSR #C661
C816 LSR #5A
C818 ROR #59
C81A ROR #58
C81C ROR #57
C81E BCC #C839
C82Ø CLC
C821 TYA
C822 ADC #53
C824 TAY
C825 LDA #5C
C827 ADC #54
C829 STA #5C
C82B LDA #5D
C82D ADC #55
C82F STA #5D
C831 LDA #5E
C833 ADC #56
C835 AND@#7F
C837 STA #5E
C839 ASL #53
C839 ASL #53
C83B ROL #54
C83D ROL #55
C83F ROL #56
C841 LDA #57
C843 ORA #58
C845 ORA #59
C847 ORA #5A
C849 BNE #C816
C84B STY #5B
C84D LDA #52
C84F PHP
C85Ø LDY@#5B .[
C852 JSR #C99F
C855 PLP
C856 BPL #C85B
C858 JSR #C8C4
C85B JMP #C8ØE
C85E JSR #C689
C861 ROL #57
C863 ROL #58
C865 ROL #59
C867 ROL #5A
C869 BIT #52
C86B PHP
C86C LDY@#57 .W
C86E BNE #C852
C87Ø JSR #C689
C873 LDX #Ø4
C875 LDA #44,X
C877 PHP
C878 JMP #C85Ø
C87B JSR #C8BC
C87E DEX
C87F STX #Ø4
C881 LDA #15,X
C883 AND #16,X
C885 STA #15,X
C887 LDA #24,X
C889 AND #25,X
C88B STA #24,X
C88D LDA #33,X
C88F AND #34,X
C891 STA #33,X
C893 LDA #42,X
C895 AND #43,X
C897 STA #42,X
C899 JMP #C8ØE
C89C JSR #C8A2
C89F JMP #C8ØE
C8A2 JSR #C8BC
C8A5 CLC
C8A6 LDA #15,X
C8A8 ADC #14,X
C8AA TAY
C8AB LDA #24,X
C8AD ADC #23,X
C8AF DEX
C8BØ JMP #C953
C8BØ JMP #C953
C8B3 JSR #C8A2
C8B6 JSR #C962
C8B9 JMP #C8ØE
C8BC LDX@#Ø4
C8BE JMP #C233
C8C1 JSR #C8DC
C8C4 SEC
C8C5 LDA@#ØØ
C8C7 TAY
C8C8 SBC #15,X
C8CA STA #15,X
C8CC TYA
C8CD SBC #24,X
C8CF STA #24,X
C8D1 TYA
C8D2 SBC #33,X
C8D4 STA #33,X
C8D6 TYA
C8D7 SBC #42,X
C8D9 STA #42,X
C8DB RTS
C8DC JSR #C434
C8DF BCC #C8F8
C8E1 LDY #15,X
C8E3 LDA #Ø321,Y
C8E6 STA #15,X
C8E8 LDA #Ø357,Y
C8EB STA #33,X
C8ED LDA #Ø33C,Y
C8FØ STA #24,X
C8F2 LDA #Ø372,Y
C8F5 STA #42,X
C8F7 RTS
C8F8 JSR #C46A
C8FB BCS #C8F7
C8FD LDX@#Ø7
C8FF JMP #C233
C9Ø2 JSR #C8BC
C9Ø5 LDA #42,X
C9Ø7 BMI #C8C4
C9Ø9 RTS
C9ØA LDX@#ØØ
C9ØC STX #52
C9ØE STX #53
C91Ø STX #54
C912 STX #55
C914 DEY
C915 INY
C916 LDA(#Ø5),Y
C918 CMP@#3Ø .Ø
C91A BCC #C93E
C91C CMP@#3A .:
C91E BCC #C92A
C92Ø SBC@#37 .7
C922 CMP@#ØA
C924 BCC #C93E
C926 CMP@#1Ø
C928 BCS #C93E
C928 BCS #C93E
C92A ASL A
C92B ASL A
C92C ASL A
C92D ASL A
C92E LDX@#Ø3
C93Ø ASL A
C931 ROL #52
C933 ROL #53
C935 ROL #54
C937 ROL #55
C939 DEX
C93A BPL #C93Ø
C93C BMI #C915
C93E TXA
C93F BPL #C959
C941 JMP #C4D6
C944 JSR #C7ØC
C947 LDX@#ØC
C949 JMP #C27B
C94C JSR #C8BC
C94F LDY #15,X
C951 LDA #24,X
C953 STA #53
C955 STY #52
C957 DEX
C958 LDY@#ØØ
C95A LDA(#52),Y
C95C JMP #C97C
C95F JSR #C94C
C962 LDY@#Ø1
C964 LDA(#52),Y
C966 STA #24,X
C968 INY
C969 LDA(#52),Y
C96B STA #33,X
C96D INY
C96E LDA(#52),Y
C97Ø STA #42,X
C972 RTS
C973 LDY@#ØD
C975 JSR #C9A1
C978 BEQ #C981
C97A LDA #Ø7
C97C JSR #C9B3
C97F STA #24,X
C981 STA #33,X
C983 STA #42,X
C985 RTS
C986 LDY@#2Ø
C988 LDA #ØA
C98A LSR A
C98B LSR A
C98C LSR A
C98D EOR #ØC
C98F ROR A
C99Ø ROL #Ø8
C992 ROL #Ø9
C994 ROL #ØA
```

```
C996 ROL #0B
C998 ROL #0C
C99A DEY
C99B BNE #C988
C99D LDY@#08
C99F LDX #04
C9A1 LDA #0001,Y
C9A4 STA #25,X
C9A6 LDA #0002,Y
C9A9 STA #34,X
C9AB LDA #0003,Y
C9AE STA #43,X
C9B0 LDA #0000,Y
C9B3 STA #16,X
C9B5 INX
C9B6 STX #04
C9B8 LDY #03
C9BA LDA@#00
C9BC RTS
C9BD JSR #C8BC
C9C0 JSR #C3CB
C9C3 LDY@#00
C9C5 LDA@#0D
C9C7 CMP(#52),Y
C9C9 BEQ #C9CE
C9CB INY
C9CC BNE #C9C7
C9CE TYA
C9CF JMP #C97C
C9D2 JSR #CEB1
C9D5 JMP #C958
C9D8 PLA
C9D9 PLA
C9DA STA #00
C9DC LDA #10
C9DE STA #05
C9E0 LDA #11
C9E2 STA #06
C9E4 JMP #C2F2
  - data -
CA24 JSR #C424
CA27 BCC #CA1B
CA29 JMP(#D004)
CA2C JSR #C78B
CA2F LDX #04
CA31 DEX
CA32 DEX
CA33 STX #04
CA35 LDY #16,X
CA37 LDA #17,X
CA39 STA #0321,Y
CA3C LDA #26,X
CA3E STA #033C,Y
CA41 LDA #35,X
CA43 STA #0357,Y
CA46 LDA #44,X
CA48 STA #0372,Y
CA4B RTS
CA4C INC #07
CA4C INC #07
CA4E JMP(#0208)
CA51 LDA@#00
CA53 JSR #C97C
CA56 LDA@#FF
CA58 JSR #C97C
CA5B STA #04
CA5D LDY@#7F
CA5F STY #26
CA61 JSR #C465
CA64 BCC #CAB8
CA66 JSR #C231
CA69 BCS #CAC3
CA6B JSR #C465
CA6E LDX@#01
CA70 STX #04
CA72 JSR #C4E4
CA75 JSR #C62E
CA78 BCC #CAAA
CA7A DEY
CA7B BCS #CA9E
CA7D LDA@#05
CA7F STA #0321
CA82 JSR #C589
CA85 LDA@#08
CA87 STA #0321
CA8A LDY #03
CA8C LDA(#58),Y
CA8E CMP@#0D
CA90 BEQ #CA98
CA92 JSR #CA4C
CA95 INY
CA96 BNE #CA8C
CA98 JSR #CD54
CA9B JSR #CEA1
CA9E LDA(#58),Y
CAA0 STA #25
CAA2 INY
CAA3 LDA(#58),Y
CAA5 STA #16
CAA7 INY
CAA8 STY #03
CAAA LDA #16
CAAC CLC
CAAD SBC #17
CAAF LDA #25
CAB1 SBC #26
CAB3 BCC #CA7D
CAB5 JMP #C2CF
CAB8 JSR #C231
CABB INC #04
CABD JSR #C465
CAC0 JMP #CA6E
CAC3 LDA #16
CAC5 LDY #25
CAC7 STA #17
CAC9 STY #26
CACB BCS #CA6E
CACD JSR #C434
CACD JSR #C434
CAD0 LDY #15
CAD2 BEQ #CAE4
CAD4 BCC #CAE5
CAD6 DEC #04
CAD8 LDA #15,X
CADA CMP #023F,Y
CADD BEQ #CAE5
CADF DEY
CAE0 STY #15
CAE2 BNE #CADA
CAE4 BRK
CAE5 LDX #023F,Y
CAE8 CLC
CAE9 LDA #0321,X
CAEC ADC #024A,Y
CAEF STA #0321,X
CAF2 STA #52
CAF4 LDA #033C,X
CAF7 ADC #0255,Y
CAFA STA #033C,X
CAFD STA #53
CAFF LDA #0357,X
CB02 ADC #0260,Y
CB05 STA #0357,X
CB08 STA #54
CB0A LDA #0372,X
CB0D ADC #026B,Y
CB10 STA #0372,X
CB13 TAX
CB14 LDA #52
CB16 SEC
CB17 SBC #0276,Y
CB1A STA #52
CB1C LDA #53
CB1E SBC #0281,Y
CB21 STA #53
CB23 LDA #54
CB25 SBC #028C,Y
CB28 STA #54
CB2A TXA
CB2B SBC #0297,Y
CB2E ORA #52
CB30 ORA #53
CB32 ORA #54
CB34 BEQ #CB45
CB36 TXA
CB37 EOR #026B,Y
CB3A EOR #0297,Y
CB3D BPL #CB43
CB3F BCS #CB45
CB41 BCC #CB52
CB43 BCS #CB52
CB45 LDA #02A2,Y
CB48 STA #05
CB4A LDA #02AD,Y
CB4D STA #06
CB4F JMP #CBFF
CB52 DEC #15
CB52 DEC #15
CB54 JMP #C558
CB57 JSR #C434
CB5A BCC #CB6D
CB5C JSR #C279
CB5F JSR #CA2C
CB62 TYA
CB63 LDY #15
CB65 CPY@#0B
CB67 BCS #CB6D
CB69 STA #0240,Y
CB6C LDA@#00
CB6E STA #026C,Y
CB71 STA #0261,Y
CB74 STA #0256,Y
CB77 LDA@#01
CB79 STA #024B,Y
CB7C LDX@#16
CB7E JMP #C233
CB81 JSR #C78B
CB84 LDY #15
CB86 DEX
CB87 STX #04
CB89 LDA #16,X
CB8B STA #0277,Y
CB8E LDA #25,X
CB90 STA #0282,Y
CB93 LDA #34,X
CB95 STA #028D,Y
CB98 LDA #43,X
CB9A STA #0298,Y
CB9D LDX@#1A
CB9F JMP #C233
CBA2 JSR #C78B
CBA5 LDY #15
CBA7 DEX
CBA8 STX #04
CBAA LDA #16,X
CBAC STA #024B,Y
CBAF LDA #25,X
CBB1 STA #0256,Y
CBB4 LDA #34,X
CBB6 STA #0261,Y
CBB9 LDA #43,X
CBBB STA #026C,Y
CBBE JSR #C50C
CBC1 LDY #15
CBC3 LDA #05
CBC5 STA #02A3,Y
CBC8 LDA #06
CBCA STA #02AE,Y
CBCD INC #15
CBCF JMP #C31B
CBD2 JSR #CC1F
CBD5 JSR #C50C
CBD8 LDY #14
CBDA CPY@#0E
CBDC BCS #CC00
CBDE LDA #05
```

```
CBE0 STA #02CF,Y
CBE3 LDA #06
CBE5 STA #02DD,Y
CBE8 INC #14
CBEA BCC #CC0B
CBEC JSR #C4E4
CBEF LDY #14
CBF1 BEQ #CC1D
CBF3 DEC #14
CBF5 LDA #02CE,Y
CBF8 STA #05
CBFA LDA #02DC,Y
CBFD STA #06
CBFF JSR #C500
CC02 JMP #C31B
CC05 JSR #CC1F
CC08 JSR #C4E4
CC0B LDA #57
CC0D BNE #CC14
CC0F JSR #C62E
CC12 BCS #CC7D
CC14 LDY #58
CC16 LDA #59
CC18 STY #05
CC1A JMP #CBFD
CC1D BRK
CC1E INY
CC1F LDA(#05),Y
CC21 CMP@#20
CC23 BEQ #CC1E
CC25 CMP@#61 .a
CC27 BCC #CC79
CC29 STA #57
CC2B SBC@#61 .a
CC2D CMP@#1B
CC2F BCS #CC79
CC31 ASL A
CC32 TAX
CC33 LDA #038D,X
CC36 STA #58
CC38 JSR #C4F6
CC3B LDA #038E,X
CC3E STA #59
CC40 ORA #58
CC42 BNE #CC78
CC44 TAY
CC45 LDA #12
CC47 STA #59
CC49 DEY
CC4A LDA@#0D
CC4C INY
CC4D CMP(#58),Y
CC4F BNE #CC4C
CC51 INY
CC52 LDA(#58),Y
CC54 BMI #CC9B
CC56 STA #02
CC58 INY
CC59 LDA(#58),Y
CC59 LDA(#58),Y
CC5B STA #01
CC5D INY
CC5E LDA(#58),Y
CC60 DEY
CC61 CMP #57
CC63 BEQ #CC6B
CC65 JSR #CEA1
CC68 JMP #CC4A
CC6B JSR #CEA2
CC6E LDA #58
CC70 STA #038D,X
CC73 LDA #59
CC75 STA #038E,X
CC78 RTS
CC79 JSR #C8BC
CC7C LDA@#00
CC7E STA #57
CC80 RTS
CC81 JSR #C372
CC84 JSR #C434
CC87 BCS #CC8E
CC89 LDX@#2B .+
CC8B JMP #C233
CC8E JSR #CD09
CC91 LDA #05
CC93 PHA
CC94 LDA #06
CC96 PHA
CC97 LDA #03
CC99 PHA
CC9A LDY@#00
CC9C STY #03
CC9E INY
CC9F STY #06
CCA1 LDY@#40 .@
CCA3 STY #05
CCA5 JSR #CA2C
CCA8 PLA
CCA9 STA #03
CCAB PLA
CCAC STA #06
CCAE PLA
CCAF STA #05
CCB1 LDX@#2C .,
CCB3 JMP #C233
CCB6 JSR #C78B
CCB9 LDY@#54 .T
CCBB JSR #C3CD
CCBE JSR #CD09
CCC1 LDX@#40 .@
CCC3 LDY@#00
CCC5 LDA #0100,X
CCC8 STA(#54),Y
CCCA CMP@#0D
CCCC BEQ #CC81
CCCE INX
CCCF INY
CCD0 BNE #CCC5
CCD0 BNE #CCC5
CCD2 JSR #C70C
CCD5 LDY #13
CCD7 BEQ #CCC4
CCD9 DEX
CCDA STX #04
CCDC LDA #16,X
CCDE BEQ #CCE5
CCE0 DEC #13
CCE2 JMP #C558
CCE5 LDA #02B8,Y
CCE8 STA #05
CCEA LDA #02C3,Y
CCED JMP #CBFD
CCF0 LDX #13
CCF2 CPX@#0B
CCF4 BCS #CD10
CCF6 DEY
CCF7 JSR #C4F6
CCFA LDA #05
CCFC STA #02B9,X
CCFF LDA #06
CD01 STA #02C4,X
CD04 INC #13
CD06 JMP #C31B
CD09 LDA@#3F .?
CD0B LDY@#40 .@
CD0D BNE #CD11
CD0F LDY@#00
CD11 JSR #CA4C
CD14 STY #52
CD16 LDY #52
CD18 JSR #FFE6
CD1B CMP@#7F
CD1D BNE #CD26
CD1F DEY
CD20 CPY #52
CD22 BPL #CD18
CD24 BMI #CD16
CD26 CMP@#18
CD28 BNE #CD30
CD2A JSR #CD54
CD2D JMP #CD16
CD30 CMP@#1B
CD32 BNE #CD37
CD34 JMP #C2CF
CD37 STA #0100,Y
CD3A CMP@#0D
CD3C BEQ #CD57
CD3E INY
CD3F TYA
CD40 SEC
CD41 SBC #52
CD43 CMP@#40 .@
CD45 BCC #CD18
CD47 JSR #FFE3
CD4A CMP@#7F
CD4C BNE #CD47
CD4E JSR #FFF4
CD4E JSR #FFF4
CD51 JMP #CD1F
CD54 JSR #FFED
CD57 LDA@#00
CD59 STA #07
CD5B RTS
CD5C JSR #C78B
CD5F JSR #CEAE
CD62 LDY@#54 .T
CD64 JSR #C3CD
CD67 LDY@#FF
CD69 INY
CD6A LDA(#52),Y
CD6C STA(#54),Y
CD6E CMP@#0D
CD70 BNE #CD69
CD72 JMP #C558
CD75 JSR #CD81
CD78 JMP #C3F1
CD7B JSR #CD81
CD7E JMP #C409
CD81 JSR #C8E1
CD84 JSR #C8BC
CD87 DEX
CD88 CLC
CD89 LDA #16,X
CD8B ADC #15,X
CD8D STA #15,X
CD8F LDA #25,X
CD91 ADC #24,X
CD93 STA #24,X
CD95 STX #04
CD97 RTS
CD98 JSR #C4E4
CD9B LDA #12
CD9D STA #0E
CD9F LDY@#00
CDA1 STY #0D
CDA3 DEY
CDA4 INY
CDA5 LDA(#0D),Y
CDA7 CMP@#0D
CDA9 BNE #CDA4
CDAB JSR #CDBC
CDAE LDA(#0D),Y
CDB0 BMI #CDB5
CDB2 INY
CDB3 BNE #CDA4
CDB5 INY
CDB6 JSR #CDBC
CDB9 JMP #C2CF
CDBC CLC
CDBD TYA
CDBE ADC #0D
CDC0 STA #0D
CDC2 BCC #CDC6
CDC4 INC #0E
CDC6 LDY@#01
CDC8 RTS
```

```
CDC9 STY #56
CDCB JSR #C62E
CDCE BCS #CE18
CDD0 LDA #58
CDD2 STA #52
CDD4 SBC@#01
CDD6 STA #58
CDD8 STA #0D
CDDA LDA #59
CDDC STA #53
CDDE SBC@#00
CDE0 STA #0E
CDE2 STA #59
CDE4 LDA@#0D
CDE6 INY
CDE7 CMP(#52),Y
CDE9 BNE #CDE6
CDEB CLC
CDEC TYA
CDED ADC #52
CDEF STA #52
CDF1 BCC #CDF5
CDF3 INC #53
CDF5 LDY@#00
CDF7 LDA(#52),Y
CDF9 STA(#0D),Y
CDFB CMP@#0D
CDFD BEQ #CE08
CDFF INY
CE00 BNE #CDF7
CE02 INC #53
CE04 INC #0E
CE06 BNE #CDF7
CE08 INY
CE09 BNE #CE0F
CE0B INC #53
CE0D INC #0E
CE0F LDA(#52),Y
CE11 STA(#0D),Y
CE13 BPL #CDFF
CE15 JSR #CDBD
CE18 LDY@#01
CE1A STY #57
CE1C DEY
CE1D LDA@#0D
CE1F CMP(#56),Y
CE21 BEQ #CE80
CE23 INY
CE24 CMP(#56),Y
CE26 BNE #CE23
CE28 INY
CE29 INY
CE2A LDA #0D
CE2C STA #54
CE2E LDA #0E
CE30 STA #55
CE32 JSR #CDBD
CE35 STA #52
CE37 LDA #0E
CE37 LDA #0E
CE39 STA #53
CE3B DEY
CE3C LDA@#55 .U
CE3E STA(#0D),Y
CE40 CMP(#0D),Y
CE42 BNE #CDF6
CE44 ASL A
CE45 STA(#0D),Y
CE47 CMP(#0D),Y
CE49 BNE #CDF6
CE4B LDA(#54),Y
CE4D STA(#52),Y
CE4F TYA
CE50 BNE #CE56
CE52 DEC #55
CE54 DEC #53
CE56 DEY
CE57 TYA
CE58 ADC #54
CE5A LDX #55
CE5C BCC #CE5F
CE5E INX
CE5F CMP #58
CE61 TXA
CE62 SBC #59
CE64 BCS #CE4B
CE66 LDY@#01
CE68 LDA #25
CE6A STA(#58),Y
CE6C INY
CE6D LDA #16
CE6F STA(#58),Y
CE71 SEC
CE72 JSR #CEA2
CE75 LDY@#FF
CE77 INY
CE78 LDA(#56),Y
CE7A STA(#58),Y
CE7C CMP@#0D
CE7E BNE #CE77
CE80 JMP #C2CF
CE83 JSR #C4E4
CE86 LDY@#00
CE88 STY #05
CE8A STY #03
CE8C LDA #12
CE8E STA #06
CE90 JMP #C55B
CE93 JSR #C4DE
CE96 DEX
CE97 JSR #C3CB
CE9A LDY@#00
CE9C LDA #17,X
CE9E STA(#52),Y
CEA0 RTS
CEA1 CLC
CEA2 TYA
CEA3 ADC #58
CEA3 ADC #58
CEA5 STA #58
CEA7 BCC #CEAB
CEA9 INC #59
CEAB JMP #C500
CEAE JSR #C279
CEB1 LDX@#26 .&
CEB3 JMP #C233
CEB6 JSR #C78B
CEB9 JSR #C3CB
CEBC LDY #03
CEBE RTS
CEBF JSR #C4F6
CEC2 STY #53
CEC4 DEY
CEC5 LDX@#00
CEC7 LDA(#05),Y
CEC9 CMP@#0D
CECB BEQ #CEC6
CECD STA #0140,X
CED0 INX
CED1 INY
CED2 CMP@#22 ."
CED4 BNE #CEC7
CED6 LDA(#05),Y
CED8 CMP@#22 ."
CEDA BEQ #CEEA
CEDC LDA@#0D
CEDE STA #013F,X
CEE1 STY #03
CEE3 LDA@#40 .@
CEE5 STA #52
CEE7 LDX #04
CEE9 RTS
CEEA INY
CEEB BCS #CEC7
CEED JSR #CEFA
CEF0 DEY
CEF1 STY #56
CEF3 SEC
CEF4 JSR #FFE0
CEF7 JMP #CD9B
CEFA JSR #CEB1
CEFD JSR #C4E4
CF00 DEY
CF01 STY #54
CF03 LDA #12
CF05 STA #55
CF07 LDX@#52 .R
CF09 RTS
CF0A JSR #CEFA
CF0D STY #58
CF0F STA #59
CF11 LDA #0D
CF13 STA #5A
CF15 LDA #0E
CF17 STA #5B
CF19 LDA@#B2
CF1B STA #56
CF1B STA #56
CF1D LDA@#C2
CF1F STA #57
CF21 CLC
CF22 JSR #FFDD
CF25 JMP #C55B
CF28 SEC
CF29 LDA@#00
CF2B ROL A
CF2C PHA
CF2D JSR #CF3E
CF30 LDX@#52 .R
CF32 PLA
CF33 JSR #FFDA
CF36 LDY@#52 .R
CF38 JSR #C99F
CF3B STA #42,X
CF3D RTS
CF3E JSR #C8BC
CF41 LDY #15,X
CF43 DEX
CF44 STX #04
CF46 RTS
CF47 JSR #C8BC
CF4A JSR #C4DE
CF4D JSR #C3CB
CF50 JSR #CF41
CF53 LDX@#52 .R
CF55 JSR #FFD7
CF58 JMP #C55B
CF5B JSR #CF3E
CF5E STY #52
CF60 JSR #FFD4
CF63 JMP #C97C
CF66 JSR #CF5B
CF69 LDY #52
CF6B JSR #FFD4
CF6E STA #24,X
CF70 JSR #FFD4
CF73 STA #33,X
CF75 JSR #FFD4
CF78 STA #42,X
CF7A RTS
CF7B JSR #C8BC
CF7E JSR #C231
CF81 JSR #C4E1
CF84 JSR #C3CB
CF87 JSR #CF41
CF8A LDA #52
CF8C JMP(#0216)
CF8F JSR #CF7B
CF92 JMP #C55B
CF95 JSR #CF7B
CF98 LDX@#01
CF9A LDA #52,X
CF9C JSR #FFD1
CF9F INX
CFA0 CPX@#04
CFA2 BCC #CF9A
```

```
CFA4 BCS #CF92
CFA6 SEC
CFA7 PHP
CFA8 JSR #CEB1
CFAB LDX@#52 .R
CFAD PLP
CFAE JSR #FFCE
CFB1 LDX #04
CFB3 JMP #C97C
CFB6 JSR #C8BC
CFB9 JSR #C4E4
CFBC JSR #CF41
CFBF JSR #FFCB
CFC2 JMP #C55B
CFC5 JSR #C22C
CFC8 JSR #CEB1
CFCB JSR #C4E4
CFCE DEY
CFCF LDA(#52),Y
CFD1 STY #55
CFD3 LDY #0F
CFD5 PHA
CFD6 JSR #FFD1
CFD9 PLA
CFDA CMP@#0D
CFDC BEQ #CFC2
CFDE LDY #55
CFE0 INY
CFE1 BNE #CFCF
CFE3 JSR #C22C
CFE6 JSR #C4E1
CFE9 JSR #C3CB
CFEC LDY@#00
CFEE STY #55
CFF0 LDY #0F
CFF2 JSR #FFD4
CFF5 LDY #55
CFF7 STA(#52),Y
CFF9 INY
CFFA CMP@#0D
CFFC BNE #CFEE
CFFE BEQ #CFC2
```

```
F02E LDY #5E
F030 LDA(#05),Y
F032 CMP@#40 .@
F034 BCC #F048
F036 CMP@#5B .[
F038 BCS #F048
F03A INY
F03B CMP(#05),Y
F03D BNE #F048
F03F JSR #F08B
F042 JSR #C94F
F045 JMP #C962
F048 JMP #CA24
F04B LDX@#FF
F04D LDY #5E
F04F DEC #5E
F051 LDA(#05),Y
F053 CMP@#40 .@
F055 BCC #F060
F057 CMP@#5B .[
F059 BCS #F060
F05B INY
F05C CMP(#05),Y
F05E BEQ #F085
F060 LDY #5E
F062 INX
F063 INY
F064 LDA #F000,X
F067 BMI #F075
F069 CMP(#05),Y
F06B BEQ #F062
F06D INX
F06E LDA #EFFF,X
F071 BPL #F06D
F073 BNE #F060
F075 STA #53
F077 LDA #F001,X
F07A STA #52
F07C STY #03
F07E LDX #04
F080 INC #5E
F082 JMP(#0052)
F085 JSR #F08B
F088 JMP #C3F1
F08B INY
F08C STY #03
F08E SBC@#40 .@
F090 PHA
F091 JSR #C8BC
F094 PLA
F095 TAY
F096 LDA #15,X
F098 ASL A
F099 ROL #24,X
F09B ASL A
F09C ROL #24,X
F09E CLC
F09F ADC #02EB,Y
F0A2 STA #15,X
F0A4 LDA #24,X
F0A6 ADC #0306,Y
F0A9 STA #24,X
F0AB BCS #F084
F0AD RTS
F0AE LDA #01
F0B0 ORA #02
F0B2 BEQ #F0D6
F0B4 JSR #C434
F0B7 BCC #F0D7
F0B9 JSR #C8BC
F0BC DEX
F0BD DEX
F0BE STX #04
F0C0 LDY #16,X
F0C0 LDY #16,X
F0C2 SEC
F0C3 LDA #23
F0C5 STA #0321,Y
F0C8 ADC #17,X
F0CA STA #23
F0CC LDA #24
F0CE STA #033C,Y
F0D1 ADC #26,X
F0D3 JMP #F119
F0D6 BRK
F0D7 LDY #03
F0D9 LDA(#05),Y
F0DB CMP@#40 .@
F0DD BCC #F0D6
F0DF CMP@#5B .[
F0E1 BCS #F0D6
F0E3 INY
F0E4 CMP(#05),Y
F0E6 BNE #F0D6
F0E8 SBC@#40 .@
F0EA PHA
F0EB INY
F0EC STY #03
F0EE JSR #C8BC
F0F1 PLA
F0F2 TAY
F0F3 LDA #23
F0F5 STA #02EB,Y
F0F8 LDA #24
F0FA STA #0306,Y
F0FD DEX
F0FE STX #04
F100 LDY #16,X
F102 INY
F103 BNE #F107
F105 INC #25,X
F107 TYA
F108 ASL A
F109 ROL #25,X
F10B ASL A
F10C ROL #25,X
F10E CLC
F10F ADC #23
F111 STA #23
F113 LDA #25,X
F115 ADC #24
F117 BCS #F0D6
F119 STA #24
F11B LDY@#00
F11D LDA@#AA
F11F STA(#23),Y
F121 CMP(#23),Y
F123 BNE #F11C
F125 LSR A
F126 STA(#23),Y
F128 CMP(#23),Y
F12A BNE #F11C
F12C JSR #C434
F12C JSR #C434
F12F BCS #F0D6
F131 LDY #03
F133 LDA(#05),Y
F135 CMP@#2C .,
F137 BNE #F13E
F139 INC #03
F13B JMP #F0AE
F13E JMP #C558
F141 LDA #0D
F143 STA #23
F145 LDA #0E
F147 STA #24
F149 JMP #CE83
F14C JSR #C4E4
F14F JSR #FE66
F152 JMP #C55B
  - data -
F291 LDY #03
F293 LDA(#05),Y
F295 INC #03
F297 CMP@#20
F299 BEQ #F291
F29B RTS
F29C INC #03
F29E JMP #C31B
F2A1 LDA(#05),Y
F2A3 CMP@#5D .]
F2A5 BEQ #F29C
F2A7 JSR #C4F6
F2AA DEC #03
F2AC JSR #F38E
F2AF DEC #03
F2B1 LDA #52
F2B3 PHA
F2B4 LDA #53
F2B6 PHA
F2B7 LDA #0321
F2BA PHA
F2BB LDA@#00
F2BD STA #34
F2BF STA #43
F2C1 LDA@#05
F2C3 STA #0321
F2C6 LDA #01
F2C8 STA #16
F2CA LDA #02
F2CC STA #25
F2CE JSR #C589
F2D1 JSR #F379
F2D4 PLA
F2D5 STA #0321
F2D8 PLA
F2D9 JSR #F37E
F2DC PLA
F2DD JSR #F376
F2E0 LDY@#00
F2E2 CPY #00
F2E4 BEQ #F2EF
```

```
F2E6 LDA #0066,Y
F2E9 JSR #F376
F2EC INY
F2ED BNE #F2E2
F2EF CPY@#03
F2F1 BEQ #F2FF
F2F3 JSR #F379
F2F6 JSR #CA4C
F2F9 JSR #CA4C
F2FC INY
F2FD BNE #F2EF
F2FF LDY@#00
F301 LDA(#05),Y
F303 CMP@#3B .;
F305 BEQ #F311
F307 CMP@#0D
F309 BEQ #F311
F30B JSR #CA4C
F30E INY
F30F BNE #F301
F311 JSR #CD54
F314 JSR #C4E4
F317 DEY
F318 LDA(#05),Y
F31A INY
F31B CMP@#3B .;
F31D BEQ #F32B
F31F LDA #06
F321 CMP@#01
F323 BNE #F328
F325 JMP #C2CF
F328 JSR #C51D
F32B JMP #F2A1
F32E JSR #F291
F331 STA #66
F333 JSR #F291
F336 CMP #66
F338 BNE #F34A
F33A CMP@#40 .@
F33C BCC #F34A
F33E CMP@#5B .[
F340 BCS #F34A
F342 SEC
F343 JSR #F08E
F346 JSR #C3CB
F349 LDY@#00
F34B LDA #0331
F34E STA(#52),Y
F350 LDA #034C
F353 INY
F354 STA(#52),Y
F356 LDA@#00
F358 INY
F359 STA(#52),Y
F35B INY
F35C STA(#52),Y
F35E BNE #F396
F360 JSR #F291
F363 CMP@#3B .;
F363 CMP@#3B .;
F365 BEQ #F36B
F367 CMP@#0D
F369 BNE #F360
F36B LDA #0331
F36E STA #52
F370 LDA #034C
F373 STA #53
F375 RTS
F376 JSR #F37E
F379 LDA@#20
F37B JMP #CA4C
F37E LDX@#FF
F380 PHA
F381 LSR A
F382 LSR A
F383 LSR A
F384 LSR A
F385 JSR #C5F9
F388 PLA
F389 AND@#0F
F38B JMP #C5F9
F38E LDX@#00
F390 STX #00
F392 STX #64
F394 STX #65
F396 JSR #F291
F399 CMP@#3A .:
F39B BEQ #F32E
F39D CMP@#3B .;
F39F BEQ #F36B
F3A1 CMP@#0D
F3A3 BEQ #F36B
F3A5 CMP@#5C .\
F3A7 BEQ #F360
F3A9 LDY@#05
F3AB SEC
F3AC ADC@#00
F3AE ASL A
F3AF ASL A
F3B0 ASL A
F3B1 ASL A
F3B2 ROL #6A
F3B4 ROL #69
F3B6 DEY
F3B7 BNE #F3B1
F3B9 INX
F3BA CPX@#03
F3BC BNE #F396
F3BE ASL #6A
F3C0 ROL #69
F3C2 LDX@#40 .@
F3C4 LDA #69
F3C6 CMP #F154,X
F3C9 BEQ #F3CF
F3CB DEX
F3CC BNE #F3C6
F3CE BRK
F3CF LDY #F194,X
F3CF LDY #F194,X
F3D2 CPY #6A
F3D4 BNE #F3CB
F3D6 LDA #F210,X
F3D9 STA #66
F3DB LDY #F250,X
F3DE STY #0F
F3E0 ROR #64
F3E2 ROR #65
F3E4 DEY
F3E5 BNE #F3E0
F3E7 LDY #0F
F3E9 CPY@#0D
F3EB BNE #F3F2
F3ED LDX@#00
F3EF JMP #F49B
F3F2 JSR #F291
F3F5 CMP@#40 .@
F3F7 BEQ #F454
F3F9 CMP@#28 .(
F3FB BEQ #F462
F3FD LDX@#01
F3FF CMP@#41 .A
F401 BEQ #F3EF
F403 DEC #03
F405 JSR #C78B
F408 JSR #F291
F40B CMP@#2C .,
F40D BNE #F440
F40F JSR #F291
F412 LDY #25
F414 BEQ #F42B
F416 LDX@#09
F418 CMP@#58 .X
F41A BEQ #F49B
F41C DEX
F41D CMP@#59 .Y
F41F BNE #F49A
F421 LDA #0F
F423 CMP@#09
F425 BNE #F49B
F427 LDX@#0E
F429 BNE #F49B
F42B LDX@#04
F42D CMP@#58 .X
F42F BEQ #F49B
F431 CMP@#59 .Y
F433 BNE #F49A
F435 DEX
F436 LDY #0F
F438 CPY@#03
F43A BCS #F49B
F43C LDX@#08
F43E BNE #F49B
F440 DEC #03
F442 LDX@#02
F444 LDY #0F
F446 CPY@#0C
F448 BEQ #F49B
F448 BEQ #F49B
F44A LDX@#05
F44C LDA #25
F44E BEQ #F49B
F450 LDX@#0C
F452 BNE #F49B
F454 JSR #C78B
F457 LDA #0F
F459 LDX@#06
F45B CMP@#01
F45D BEQ #F49B
F45F INX
F460 BNE #F49B
F462 JSR #C78B
F465 JSR #F291
F468 CMP@#29 .)
F46A BEQ #F482
F46C CMP@#2C .,
F46E BNE #F49A
F470 JSR #F291
F473 CMP@#58 .X
F475 BNE #F49A
F477 JSR #F291
F47A CMP@#29 .)
F47C BNE #F49A
F47E LDX@#0B
F480 BNE #F49B
F482 LDX@#0D
F484 LDA #0F
F486 CMP@#0B
F488 BEQ #F49B
F48A LDX@#0A
F48C JSR #F291
F48F CMP@#2C .,
F491 BNE #F49A
F493 JSR #F291
F496 CMP@#59 .Y
F498 BEQ #F49B
F49A BRK
F49B JSR #F360
F49E LDA #F1D5,X
F4A1 BEQ #F4A7
F4A3 AND #64
F4A5 BNE #F4AE
F4A7 LDA #F1E4,X
F4AA AND #65
F4AC BEQ #F49A
F4AE CLC
F4AF LDA #F1F3,X
F4B2 ADC #66
F4B4 STA #66
F4B6 LDA #F202,X
F4B9 LDX@#00
F4BB STX #04
F4BD LDY #16
F4BF STY #67
F4C1 LDY #25
F4C3 STY #68
F4C5 CMP@#0F
```

```
F4C7 BEQ #F4EC
F4C9 AND@#ØF
F4CB TAY
F4CC INY
F4CD STY #ØØ
F4CF CPY@#Ø2
F4D1 BNE #F4D7
F4D3 LDY #68
F4D5 BNE #F49A
F4D7 LDY@#ØØ
F4D9 LDA #ØØ66,Y
F4DC STA(#52),Y
F4DE INY
F4DF INC #Ø331
F4E2 BNE #F4E7
F4E4 INC #Ø34C
F4E7 CPY #ØØ
F4E9 BNE #F4D9
F4EB RTS
F4EC LDA@#Ø2
F4EE STA #ØØ
F4FØ SEC
F4F1 LDA #67
F4F3 SBC #Ø331
F4F6 STA #67
F4F8 LDA #68
F4FA SBC #Ø34C
F4FD STA #68
F4FF SEC
F5ØØ LDA #67
F5Ø2 SBC@#Ø2
F5Ø4 STA #67
F5Ø6 TAY
F5Ø7 LDA #68
F5Ø9 SBC@#ØØ
F5ØB BEQ #F52C
F5ØD CMP@#FF
F5ØF BEQ #F527
F511 JSR #F7D1
  - data -
F523 STY #67
F525 BMI #F4D7
F527 TYA
F528 BMI #F4D7
F52A BPL #F511
F52C TYA
F52D BPL #F4D7
F52F BMI #F511
F531 JSR #C4E4
F534 DEY
F535 STY #52
F537 LDA #12
F539 STA #53
F53B TYA
F53C INY
F53D STA(#52),Y
F53F JMP #CD9B
F542 LDX@#Ø5
F544 BNE #F548
F544 BNE #F548
F546 LDX@#ØC
F548 STX #16
F54A INC #Ø4
F54C BNE #F554
F54E JSR #C8BC
F551 JSR #C231
F554 JSR #C8BC
F557 JSR #C231
F55A JSR #C8BC
F55D JSR #C4E4
F56Ø LDA #15,X
F562 STA #5C
F564 LDA #24,X
F566 STA #5D
F568 LDA #14,X
F56A STA #5A
F56C LDA #23,X
F56E STA #5B
F57Ø LDX@#ØØ
F572 STX #Ø4
F574 LDX@#Ø3
F576 LDA #Ø3C1,X
F579 STA #52,X
F57B DEX
F57C BPL #F576
F57E LDA #16
F58Ø AND@#Ø4
F582 BNE #F597
F584 LDX@#Ø2
F586 CLC
F587 LDA #5A,X
F589 ADC #52,X
F58B STA #5A,X
F58D LDA #5B,X
F58F ADC #53,X
F591 STA #5B,X
F593 DEX
F594 DEX
F595 BPL #F586
F597 LDX@#Ø3
F599 LDA #5A,X
F59B STA #Ø3C1,X
F59E DEX
F59F BPL #F599
F5A1 LDA #16
F5A3 AND@#Ø3
F5A5 BEQ #F5B2
F5A7 STA #5E
F5A9 LDA #16
F5AB AND@#Ø8
F5AD BEQ #F5B5
F5AF JSR #F678
F5B2 JMP #C55B
F5B5 LDX@#Ø2
F5B7 SEC
F5B8 LDA #5A,X
F5BA SBC #52,X
F5BC LDY #52,X
F5BC LDY #52,X
F5BE STY #5A,X
F5CØ STA #52,X
F5C2 LDY #53,X
F5C4 LDA #5B,X
F5C6 SBC #53,X
F5C8 STY #5B,X
F5CA STA #53,X
F5CC STA #56,X
F5CE BPL #F5DD
F5DØ LDA@#ØØ
F5D2 SEC
F5D3 SBC #52,X
F5D5 STA #52,X
F5D7 LDA@#ØØ
F5D9 SBC #53,X
F5DB STA #53,X
F5DD DEX
F5DE DEX
F5DF BPL #F5B7
F5E1 LDA #54
F5E3 CMP #52
F5E5 LDA #55
F5E7 SBC #53
F5E9 BCC #F61C
F5EB LDA@#ØØ
F5ED SBC #54
F5EF STA #57
F5F1 LDA@#ØØ
F5F3 SBC #55
F5F5 SEC
F5F6 ROR A
F5F7 STA #59
F5F9 ROR #57
F5FB JSR #F678
F5FE LDA #5C
F6ØØ CMP #Ø3C3
F6Ø3 BNE #F6ØF
F6Ø5 LDA #5D
F6Ø7 CMP #Ø3C4
F6ØA BNE #F6ØF
F6ØC JMP #C55B
F6ØF JSR #F655
F612 LDA #59
F614 BMI #F5FB
F616 JSR #F644
F619 JMP #F5FB
F61C LDA #53
F61E LSR A
F61F STA #59
F621 LDA #52
F623 ROR A
F624 STA #57
F626 JSR #F678
F629 LDA #5A
F62B CMP #Ø3C1
F62E BNE #F637
F63Ø LDA #5B
F632 CMP #Ø3C2
F632 CMP #Ø3C2
F635 BEQ #F6ØC
F637 JSR #F644
F63A LDA #59
F63C BPL #F626
F63E JSR #F655
F641 JMP #F626
F644 SEC
F645 LDA #57
F647 SBC #54
F649 STA #57
F64B LDA #59
F64D SBC #55
F64F STA #59
F651 LDX@#ØØ
F653 BEQ #F664
F655 CLC
F656 LDA #57
F658 ADC #52
F65A STA #57
F65C LDA #59
F65E ADC #53
F66Ø STA #59
F662 LDX@#Ø2
F664 LDA #56,X
F666 BPL #F671
F668 LDA #5A,X
F66A BNE #F66E
F66C DEC #5B,X
F66E DEC #5A,X
F67Ø RTS
F671 INC #5A,X
F673 BNE #F67Ø
F675 INC #5B,X
F677 RTS
F678 JMP(#Ø3FE)
F67B JSR #C3C8
F67E LDY@#ØØ
F68Ø LDA #52
F682 BEQ #F6C2
F684 CMP@#Ø5
F686 BCC #F68A
F688 LDA@#Ø4
F68A LDX@#8Ø
F68C STX #54
F68E STY #53
F69Ø STA #52
F692 TAX
F693 LDA #F6CE,X
F696 LDX #12
F698 BPL #F69E
F69A CMP #12
F69C BCS #F67F
F69E TAX
F69F TYA
F6A0 STA(#53),Y
F6A2 DEY
F6A3 BNE #F6A0
F6A5 INC #54
```

```
F6A7 CPX #54
F6A9 BNE #F6A0
F6AB LDY #52
F6AD LDA #F6D8,Y
F6B0 STA #03FF
F6B3 LDA #F6D3,Y
F6B6 STA #03FE
F6B9 LDA #F6DD,Y
F6BC STA #B000
F6BF JMP #C558
F6C2 LDA@#40 .@
F6C4 STA #8000,Y
F6C7 STA #8100,Y
F6CA DEY
F6CB BNE #F6C4
F6CD BEQ #F6AB
  - data -
F6E2 LDA #5B
F6E4 ORA #5D
F6E6 BNE #F73A
F6E8 LDA #5A
F6EA CMP@#40 .@
F6EC BCS #F73A
F6EE LSR A
F6EF STA #5F
F6F1 LDA@#2F ./
F6F3 SEC
F6F4 SBC #5C
F6F6 CMP@#30 .0
F6F8 BCS #F73A
F6FA LDX@#FF
F6FC SEC
F6FD INX
F6FE SBC@#03
F700 BCS #F6FD
F702 ADC@#03
F704 STA #61
F706 TXA
F707 ASL A
F708 ASL A
F709 ASL A
F70A ASL A
F70B ASL A
F70C ORA #5F
F70E STA #5F
F710 LDA@#80
F712 ADC@#00
F714 STA #60
F716 LDA #5A
F718 LSR A
F719 LDA #61
F71B ROL A
F71C TAY
F71D LDA #F7CB,Y
F720 LDY@#00
F722 LDX #5E
F724 DEX
F725 BEQ #F736
F727 DEX
F727 DEX
F728 BEQ #F731
F72A EOR@#FF
F72C AND(#5F),Y
F72E STA(#5F),Y
F730 RTS
F731 EOR(#5F),Y
F733 STA(#5F),Y
F735 RTS
F736 ORA(#5F),Y
F738 STA(#5F),Y
F73A RTS
F73B LDA #5B
F73D ORA #5D
F73F BNE #F73A
F741 LDA #5A
F743 BMI #F73A
F745 LSR A
F746 LSR A
F747 LSR A
F748 STA #5F
F74A LDA@#3F .?
F74C SEC
F74D SBC #5C
F74F CMP@#40 .@
F751 BCC #F785
F753 RTS
F754 LDA #5B
F756 ORA #5D
F758 BNE #F73A
F75A LDA #5A
F75C BMI #F73A
F75E LSR A
F75F LSR A
F760 LSR A
F761 STA #5F
F763 LDA@#5F ._
F765 SEC
F766 SBC #5C
F768 CMP@#60 .`
F76A BCC #F785
F76C RTS
F76D LDA #5B
F76F ORA #5D
F771 BNE #F73A
F773 LDA #5A
F775 BMI #F73A
F777 LSR A
F778 LSR A
F779 LSR A
F77A STA #5F
F77C LDA@#BF
F77E SEC
F77F SBC #5C
F781 CMP@#C0
F783 BCS #F73A
F785 LDY@#00
F787 STY #60
F789 ASL A
F789 ASL A
F78A ROL #60
F78C ASL A
F78D ROL #60
F78F ASL A
F790 ROL #60
F792 ASL A
F793 ROL #60
F795 ADC #5F
F797 STA #5F
F799 LDA #60
F79B ADC@#80
F79D STA #60
F79F LDA #5A
F7A1 AND@#07
F7A3 TAY
F7A4 LDA #F7C9,Y
F7A7 JMP #F720
F7AA LDA #5B
F7AC ORA #5D
F7AE BNE #F76C
F7B0 LDA #5A
F7B2 LSR A
F7B3 LSR A
F7B4 LSR A
F7B5 STA #5F
F7B7 LDA@#BF
F7B9 SEC
F7BA SBC #5C
F7BC CMP@#C0
F7BE BCS #F76C
F7C0 LDY@#00
F7C2 STY #60
F7C4 ASL A
F7C5 ROL #60
F7C7 BPL #F789
  - data -
F7D1 PLA
F7D2 STA #E8
F7D4 PLA
F7D5 STA #E9
F7D7 LDY@#00
F7D9 INC #E8
F7DB BNE #F7DF
F7DD INC #E9
F7DF LDA(#E8),Y
F7E1 BMI #F7E9
F7E3 JSR #FFF4
F7E6 JMP #F7D7
F7E9 JMP(#00E8)
F7EC LDX@#D4
F7EE JSR #F7F1
F7F1 LDA #01,X
F7F3 JSR #F802
F7F6 INX
F7F7 INX
F7F8 LDA #FE,X
F7FA JSR #F802
F7FD LDA@#20
F7FD LDA@#20
F7FF JMP #FFF4
F802 PHA
F803 LSR A
F804 LSR A
F805 LSR A
F806 LSR A
F807 JSR #F80B
F80A PLA
F80B AND@#0F
F80D CMP@#0A
F80F BCC #F813
F811 ADC@#06
F813 ADC@#30 .0
F815 JMP #FFF4
F818 JSR #F876
F81B LDX@#00
F81D CMP@#22 ."
F81F BEQ #F827
F821 INX
F822 BNE #F83F
F824 JMP #FA7D
F827 INY
F828 LDA #0100,Y
F82B CMP@#0D
F82D BEQ #F824
F82F STA #0140,X
F832 INX
F833 CMP@#22 ."
F835 BNE #F827
F837 INY
F838 LDA #0100,Y
F83B CMP@#22 ."
F83D BEQ #F827
F83F LDA@#0D
F841 STA #013F,X
F844 LDA@#40 .@
F846 STA #C9
F848 LDA@#01
F84A STA #CA
F84C LDX@#C9
F84E RTS
F84F LDY@#00
F851 LDA #00,X
F853 STA #00C9,Y
F856 INX
F857 INY
F858 CPY@#0A
F85A BCC #F851
F85C LDY@#FF
F85E LDA@#0D
F860 INY
F861 CPY@#0E
F863 BCS #F86C
F865 CMP(#C9),Y
F867 BNE #F860
F869 CPY@#00
F86B RTS
F86C JSR #F7D1
```

```
F86C JSR #F7D1
  - data -
F873 NOP
F874 BRK
F875 INY
F876 LDA #0100,Y
F879 CMP@#20
F87B BEQ #F875
F87D RTS
F87E CMP@#30 .0
F880 BCC #F891
F882 CMP@#3A .:
F884 BCC #F88E
F886 SBC@#07
F888 BCC #F891
F88A CMP@#40 .@
F88C BCS #F890
F88E AND@#0F
F890 RTS
F891 SEC
F892 RTS
F893 LDA@#00
F895 STA #00,X
F897 STA #01,X
F899 STA #02,X
F89B JSR #F876
F89E LDA #0100,Y
F8A1 JSR #F87E
F8A4 BCS #F8BB
F8A6 ASL A
F8A7 ASL A
F8A8 ASL A
F8A9 ASL A
F8AA STY #02,X
F8AC LDY@#04
F8AE ASL A
F8AF ROL #00,X
F8B1 ROL #01,X
F8B3 DEY
F8B4 BNE #F8AE
F8B6 LDY #02,X
F8B8 INY
F8B9 BNE #F89E
F8BB LDA #02,X
F8BD RTS
  - data -
F8EF LDX@#FF
F8F1 CLD
F8F2 LDY@#00
F8F4 STY #DD
F8F6 JSR #F876
F8F9 DEY
F8FA INY
F8FB INX
F8FC LDA #F8BE,X
F8FF BMI #F919
F901 CMP #0100,Y
F904 BEQ #F8FA
F906 DEX
F906 DEX
F907 INX
F908 LDA #F8BE,X
F90B BPL #F907
F90D INX
F90E LDA #0100,Y
F911 CMP@#2E ..
F913 BNE #F8F2
F915 INY
F916 DEX
F917 BCS #F8FC
F919 STA #CA
F91B LDA #F8BF,X
F91E STA #C9
F920 CLC
F921 LDX@#00
F923 JMP(#00C9)
F926 JSR #F7D1
  - data -
F92D NOP
F92E BRK
F92F JSR #FB8E
F932 BVC #F92E
F934 BEQ #F92F
F936 JSR #FC2B
F939 LDY@#00
F93B JSR #FFD4
F93E STA(#CB),Y
F940 INC #CB
F942 BNE #F946
F944 INC #CC
F946 LDX@#D4
F948 JSR #FA08
F94B BNE #F93B
F94D SEC
F94E ROR #DD
F950 CLC
F951 ROR #DD
F953 PLP
F954 RTS
F955 SEC
F956 ROR #DD
F958 JSR #F818
F95B LDX@#CB
F95D JSR #F893
F960 BEQ #F966
F962 LDA@#FF
F964 STA #CD
F966 JSR #FA76
F969 LDX@#C9
F96B JMP(#020C)
F96E PHP
F96F SEI
F970 JSR #F84F
F973 PHP
F974 JSR #FC3E
F977 PLP
F978 BEQ #F92F
F97A LDA@#00
F97A LDA@#00
F97C STA #D0
F97E STA #D1
F980 JSR #F9A2
F983 BCC #F94E
F985 INC #D0
F987 INC #CC
F989 BNE #F980
F98B CLC
F98C BCC #F94E
F98E JSR #FFF4
F991 INY
F992 LDA #00ED,Y
F995 CMP@#0D
F997 BNE #F98E
F999 INY
F99A JSR #F7FD
F99D CPY@#0E
F99F BCC #F999
F9A1 RTS
F9A2 LDA@#00
F9A4 STA #DC
F9A6 JSR #FB8E
F9A9 BVC #F9A3
F9AB BNE #F9A2
F9AD JSR #FBC9
F9B0 PHP
F9B1 JSR #FBE2
F9B4 PLP
F9B5 BEQ #F9C7
F9B7 LDA #DB
F9B9 AND@#20
F9BB ORA #EA
F9BD BNE #F9A2
F9BF JSR #F992
F9C2 JSR #FFED
F9C5 BNE #F9A2
F9C7 LDX@#02
F9C9 LDA #DD
F9CB BMI #F9E0
F9CD LDA #CF,X
F9CF CMP #D8,X
F9D1 BCS #F9DB
F9D3 LDA@#05
F9D5 JSR #FC40
F9D8 JSR #FC3E
F9DB BNE #F9A2
F9DD DEX
F9DE BNE #F9CD
F9E0 JSR #FC2B
F9E3 BIT #DB
F9E5 BVC #F9F2
F9E7 DEY
F9E8 INY
F9E9 JSR #FFD4
F9EC STA(#CB),Y
F9EE CPY #D8
F9F0 BNE #F9E8
F9F2 LDA #DC
F9F2 LDA #DC
F9F4 STA #CE
F9F6 JSR #FFD4
F9F9 CMP #CE
F9FB BEQ #FA05
F9FD JSR #F7D1
  - data -
FA03 NOP
FA04 BRK
FA05 ROL #DB
FA07 RTS
FA08 INC #00,X
FA0A BNE #FA0E
FA0C INC #01,X
FA0E LDA #00,X
FA10 CMP #02,X
FA12 BNE #FA18
FA14 LDA #01,X
FA16 CMP #03,X
FA18 RTS
FA19 DEX
FA1A JSR #FA76
FA1D STX #EA
FA1F RTS
FA20 JSR #F958
FA23 BIT #DD
FA25 BVS #FA73
FA27 JMP(#00D6)
FA2A PHP
FA2B JSR #FA76
FA2E JSR #FC3E
FA31 JSR #FB8E
FA34 BVS #FA38
FA36 PLP
FA37 RTS
FA38 BEQ #FA44
FA3A LDY@#00
FA3C JSR #F999
FA3F JSR #F7EC
FA42 BNE #FA5D
FA44 JSR #FBC9
FA47 JSR #FBE2
FA4A JSR #F992
FA4D JSR #F7EC
FA50 ROL #DB
FA52 BPL #FA5D
FA54 INX
FA55 JSR #F7F1
FA58 LDA #FD,X
FA5A JSR #F802
FA5D JSR #FFED
FA60 BNE #FA31
FA62 JMP #FFED
FA65 JSR #F893
FA68 BEQ #FA7D
FA6A RTS
FA6B LDX@#CB
FA6D JSR #FA65
FA70 JSR #FA76
```

```
FA73 JMP(#00CB)
FA76 JSR #F876
FA79 CMP@#0D
FA7B BEQ #FA1F
FA7D JSR #F7D1
  - data -
FA84 NOP
FA85 BRK
FA86 SEC
FA87 LDA #D1
FA89 SBC #CF
FA8B PHA
FA8C LDA #D2
FA8E SBC #D0
FA90 TAY
FA91 PLA
FA92 CLC
FA93 ADC #CB
FA95 STA #CD
FA97 TYA
FA98 ADC #CC
FA9A STA #CE
FA9C LDY@#04
FA9E LDA #00CA,Y
FAA1 JSR #FFD1
FAA4 DEY
FAA5 BNE #FA9E
FAA7 LDA(#CF),Y
FAA9 JSR #FFD1
FAAC INC #CF
FAAE BNE #FAB2
FAB0 INC #D0
FAB2 LDX@#CB
FAB4 JSR #FA08
FAB7 BNE #FAA7
FAB9 PLP
FABA RTS
FABB JSR #F818
FABE LDX@#CB
FAC0 JSR #FA65
FAC3 LDX@#D1
FAC5 JSR #FA65
FAC8 LDX@#CD
FACA JSR #F893
FACD PHP
FACE LDA #CB
FAD0 LDX #CC
FAD2 PLP
FAD3 BNE #FAD9
FAD5 STA #CD
FAD7 STX #CE
FAD9 STA #CF
FADB STX #D0
FADD JSR #FA76
FAE0 LDX@#C9
FAE2 JMP(#020E)
FAE5 PHP
FAE6 SEI
FAE7 JSR #F84F
FAE7 JSR #F84F
FAEA PHP
FAEB LDA@#06
FAED JSR #FC40
FAF0 LDX@#07
FAF2 JSR #FB7A
FAF5 PLP
FAF6 BEQ #FA86
FAF8 LDX@#04
FAFA LDA #CE,X
FAFC STA #D2,X
FAFE DEX
FAFF BNE #FAFA
FB01 STX #D0
FB03 STX #D1
FB05 LDA #D5
FB07 BNE #FB0B
FB09 DEC #D6
FB0B DEC #D5
FB0D CLC
FB0E ROR #D2
FB10 SEC
FB11 LDX@#FF
FB13 LDA #D5
FB15 SBC #D3
FB17 STA #CF
FB19 LDA #D6
FB1B SBC #D4
FB1D PHP
FB1E ROR #D2
FB20 PLP
FB21 BCC #FB29
FB23 CLC
FB24 BEQ #FB29
FB26 STX #CF
FB28 SEC
FB29 ROR #D2
FB2B INX
FB2C JSR #FB3B
FB2F INC #D0
FB31 INC #D4
FB33 INC #CC
FB35 ROL #D2
FB37 BCS #FB0E
FB39 PLP
FB3A RTS
FB3B LDX@#07
FB3D JSR #FB7A
FB40 STX #DC
FB42 LDY@#04
FB44 LDA@#2A .*
FB46 JSR #FFD1
FB49 DEY
FB4A BNE #FB44
FB4C LDA(#C9),Y
FB4E JSR #FFD1
FB51 INY
FB52 CMP@#0D
FB54 BNE #FB4C
FB54 BNE #FB4C
FB56 LDY@#08
FB58 LDA #00CA,Y
FB5B JSR #FFD1
FB5E DEY
FB5F BNE #FB58
FB61 JSR #FB81
FB64 BIT #D2
FB66 BVC #FB73
FB68 DEY
FB69 INY
FB6A LDA(#D3),Y
FB6C JSR #FFD1
FB6F CPY #CF
FB71 BNE #FB69
FB73 LDA #DC
FB75 JSR #FFD1
FB78 LDX@#04
FB7A STX #B002
FB7D LDX@#78 .x
FB7F BNE #FB83
FB81 LDX@#1E
FB83 JSR #FE66
FB86 DEX
FB87 BNE #FB83
FB89 RTS
FB8A LDX@#06
FB8C BNE #FB83
FB8E BIT #B001
FB91 BPL #FB8E
FB93 BVC #FB8E
FB95 LDY@#00
FB97 STA #C3
FB99 LDA@#10
FB9B STA #C2
FB9D BIT #B001
FBA0 BPL #FBB1
FBA2 BVC #FBB1
FBA4 JSR #FCBD
FBA7 BCS #FB95
FBA9 DEC #C3
FBAB BNE #FB9D
FBAD DEC #C2
FBAF BNE #FB9D
FBB1 BVS #FBB4
FBB3 RTS
FBB4 LDY@#04
FBB6 PHP
FBB7 JSR #FBE4
FBBA PLP
FBBB LDY@#04
FBBD LDA@#2A .*
FBBF CMP #00D3,Y
FBC2 BNE #FBC7
FBC4 DEY
FBC5 BNE #FBBF
FBC7 RTS
FBC8 INY
FBC9 JSR #FFD4
FBC9 JSR #FFD4
FBCC STA #00ED,Y
FBCF CMP@#0D
FBD1 BNE #FBC8
FBD3 LDY@#FF
FBD5 INY
FBD6 LDA(#C9),Y
FBD8 CMP #00ED,Y
FBDB BNE #FBC7
FBDD CMP@#0D
FBDF BNE #FBD5
FBE1 RTS
FBE2 LDY@#08
FBE4 JSR #FFD4
FBE7 STA #00D3,Y
FBEA DEY
FBEB BNE #FBE4
FBED RTS
FBEE STX #EC
FBF0 STY #C3
FBF2 PHP
FBF3 SEI
FBF4 LDA@#78 .x
FBF6 STA #C0
FBF8 JSR #FCBD
FBFB BCC #FBF4
FBFD INC #C0
FBFF BPL #FBF8
FC01 LDA@#53 .S
FC03 STA #C4
FC05 LDX@#00
FC07 LDY #B002
FC0A JSR #FCCD
FC0D BEQ #FC0F
FC0F BEQ #FC12
FC11 INX
FC12 DEC #C4
FC14 BNE #FC0A
FC16 CPX@#0C
FC18 ROR #C0
FC1A BCC #FC01
FC1C LDA #C0
FC1E PLP
FC1F LDY #C3
FC21 LDX #EC
FC23 PHA
FC24 CLC
FC25 ADC #DC
FC27 STA #DC
FC29 PLA
FC2A RTS
FC2B LDA #CD
FC2D BMI #FC37
FC2F LDA #D4
FC31 STA #CB
FC33 LDA #D5
FC35 STA #CC
FC37 RTS
FC38 BCS #FC3E
```

```
FC3A LDA@#06
FC3C BNE #FC40
FC3E LDA@#04
FC40 LDX@#07
FC42 STX #B002
FC45 BIT #EA
FC47 BNE #FC76
FC49 CMP@#05
FC4B BEQ #FC63
FC4D BCS #FC58
FC4F JSR #F7D1
  - data -
FC56 BNE #FC6D
FC58 JSR #F7D1
  - data -
FC61 BNE #FC6D
FC63 JSR #F7D1
  - data -
FC6C NOP
FC6D JSR #F7D1
  - data -
FC75 NOP
FC76 JSR #FFE3
FC79 JMP #FFED
FC7C STX #EC
FC7E STY #C3
FC80 PHP
FC81 SEI
FC82 PHA
FC83 JSR #FC23
FC86 STA #C0
FC88 JSR #FCD8
FC8B LDA@#0A
FC8D STA #C1
FC8F CLC
FC90 BCC #FC9C
FC92 LDX@#07
FC94 STX #B002
FC97 JSR #FCDA
FC9A BMI #FCAF
FC9C LDY@#04
FC9E LDA@#04
FCA0 STA #B002
FCA3 JSR #FCD8
FCA6 INC #B002
FCA9 JSR #FCD8
FCAC DEY
FCAD BNE #FC9E
FCAF SEC
FCB0 ROR #C0
FCB2 DEC #C1
FCB4 BNE #FC90
FCB6 LDY #C3
FCB8 LDX #EC
FCBA PLA
FCBB PLP
FCBC RTS
FCBD LDX@#00
FCBF LDY #B002
FCBF LDY #B002
FCC2 INX
FCC3 BEQ #FCCC
FCC5 JSR #FCCD
FCC8 BEQ #FCC2
FCCA CPX@#08
FCCC RTS
FCCD STY #C5
FCCF LDA #B002
FCD2 TAY
FCD3 EOR #C5
FCD5 AND@#20
FCD7 RTS
FCD8 LDX@#00
FCDA LDA@#10
FCDC BIT #B002
FCDF BEQ #FCDC
FCE1 BIT #B002
FCE4 BNE #FCE1
FCE6 DEX
FCE7 BPL #FCDC
FCE9 RTS
FCEA CMP@#06
FCEC BEQ #FD0B
FCEE CMP@#15
FCF0 BEQ #FD11
FCF2 LDY #E0
FCF4 BMI #FD19
FCF6 CMP@#1B
FCF8 BEQ #FD0B
FCFA CMP@#07
FCFC BEQ #FD1A
FCFE JSR #FD44
FD01 LDX@#0A
FD03 JSR #FEC5
FD06 BNE #FD29
FD08 JMP #FEB7
FD0B CLC
FD0C LDX@#00
FD0E STX #B000
FD11 LDX@#02
FD13 PHP
FD14 ASL #DE,X
FD16 PLP
FD17 ROR #DE,X
FD19 RTS
FD1A LDA@#05
FD1C TAY
FD1D STA #B003
FD20 DEX
FD21 BNE #FD20
FD23 EOR@#01
FD25 INY
FD26 BPL #FD1D
FD28 RTS
FD29 CMP@#20
FD2B BCC #FD44
FD2D ADC@#1F
FD2F BMI #FD33
FD2F BMI #FD33
FD31 EOR@#60 .`
FD33 JSR #FE6B
FD36 STA(#DE),Y
FD38 INY
FD39 CPY@#20
FD3B BCC #FD42
FD3D JSR #FDEC
FD40 LDY@#00
FD42 STY #E0
FD44 PHA
FD45 JSR #FE6B
FD48 LDA(#DE),Y
FD4A EOR #E1
FD4C STA(#DE),Y
FD4E PLA
FD4F RTS
FD50 JSR #FE35
FD53 LDA@#20
FD55 JSR #FE6B
FD58 STA(#DE),Y
FD5A BPL #FD42
FD5C JSR #FE35
FD5F JMP #FD42
FD62 JSR #FDEC
FD65 LDY #E0
FD67 BPL #FD42
FD69 LDY@#80
FD6B STY #E1
FD6D LDY@#00
FD6F STY #B000
FD72 LDA@#20
FD74 STA #8000,Y
FD77 STA #8100,Y
FD7A INY
FD7B BNE #FD74
FD7D LDA@#80
FD7F LDY@#00
FD81 STA #DF
FD83 STY #DE
FD85 BEQ #FD42
FD87 JSR #FE3A
FD8A JMP #FD42
FD8D CLC
FD8E LDA@#10
FD90 STA #E6
FD92 LDX@#08
FD94 JSR #FD13
FD97 JMP #FD44
FD9A LDA #E7
FD9C EOR@#60 .`
FD9E STA #E7
FDA0 BCS #FDAB
FDA2 AND@#05
FDA4 ROL #B001
FDA7 ROL A
FDA8 JSR #FCEA
FDAB JMP #FE9A
FDAE LDY #E0
FDAE LDY #E0
FDB0 JSR #FE6B
FDB3 LDA(#DE),Y
FDB5 EOR #E1
FDB7 BMI #FDBB
FDB9 EOR@#60 .`
FDBB SBC@#20
FDBD JMP #FDE9
FDC0 LDA@#5F ._
FDC2 EOR@#20
FDC4 BNE #FDE9
FDC6 EOR #E7
FDC8 BIT #B001
FDCB BMI #FDCF
FDCD EOR@#60 .`
FDCF JMP #FDDF
FDD2 ADC@#39 .9
FDD4 BCC #FDC8
FDD6 EOR@#10
FDD8 BIT #B001
FDDB BMI #FDDF
FDDD EOR@#10
FDDF CLC
FDE0 ADC@#20
FDE2 BIT #B001
FDE5 BVS #FDE9
FDE7 AND@#1F
FDE9 JMP #FE60
FDEC LDA #DE
FDEE LDY #DF
FDF0 CPY@#81
FDF2 BCC #FE2C
FDF4 CMP@#E0
FDF6 BCC #FE2C
FDF8 LDY #E6
FDFA BMI #FE08
FDFC DEY
FDFD BNE #FE06
FDFF JSR #FE71
FE02 BCS #FDFF
FE04 LDY@#10
FE06 STY #E6
FE08 LDY@#20
FE0A JSR #FE66
FE0D LDA #8000,Y
FE10 STA #7FE0,Y
FE13 INY
FE14 BNE #FE0D
FE16 JSR #FE6B
FE19 LDA #8100,Y
FE1C STA #80E0,Y
FE1F INY
FE20 BNE #FE19
FE22 LDY@#1F
FE24 LDA@#20
FE26 STA(#DE),Y
FE28 DEY
FE29 BPL #FE26
FE2B RTS
```

```
FE2C ADC@#20
FE2E STA #DE
FE30 BNE #FE34
FE32 INC #DF
FE34 RTS
FE35 DEY
FE36 BPL #FE51
FE38 LDY@#1F
FE3A LDA #DE
FE3C BNE #FE49
FE3E LDX #DF
FE40 CPX@#80
FE42 BNE #FE49
FE44 PLA
FE45 PLA
FE46 JMP #FD65
FE49 SBC@#20
FE4B STA #DE
FE4D BCS #FE51
FE4F DEC #DF
FE51 RTS
FE52 JSR #FEFB
FE55 PHP
FE56 PHA
FE57 CLD
FE58 STY #E5
FE5A STX #E4
FE5C JSR #FCEA
FE5F PLA
FE60 LDX #E4
FE62 LDY #E5
FE64 PLP
FE65 RTS
FE66 BIT #B002
FE69 BPL #FE66
FE6B BIT #B002
FE6E BMI #FE6B
FE70 RTS
FE71 LDY@#3B .;
FE73 CLC
FE74 LDA@#20
FE76 LDX@#0A
FE78 BIT #B001
FE7B BEQ #FE85
FE7D INC #B000
FE80 DEY
FE81 DEX
FE82 BNE #FE78
FE82 BNE #FE78
FE84 LSR A
FE85 PHP
FE86 PHA
FE87 LDA #B000
FE8A AND@#F0
FE8C STA #B000
FE8F PLA
FE90 PLP
FE91 BNE #FE76
FE93 RTS
FE94 PHP
FE95 CLD
FE96 STX #E4
FE98 STY #E5
FE9A BIT #B002
FE9D BVC #FEA4
FE9F JSR #FE71
FEA2 BCC #FE9A
FEA4 JSR #FB8A
FEA7 JSR #FE71
FEAA BCS #FEA7
FEAC JSR #FE71
FEAF BCS #FEA7
FEB1 TYA
FEB2 LDX@#17
FEB4 JSR #FEC5
FEB7 LDA #FEE3,X
FEBA STA #E2
FEBC LDA@#FD
FEBE STA #E3
FEC0 TYA
FEC1 JMP(#00E2)
FEC4 DEX
FEC5 CMP #FECB,X
FEC8 BCC #FEC4
FECA RTS
  - data -
FEFB PHA
FEFC CMP@#02
FEFE BEQ #FF27
FF00 CMP@#03
FF02 BEQ #FF38
FF04 CMP #FE
FF06 BEQ #FF36
FF08 LDA #B80C
FF0B AND@#0E
FF0D BEQ #FF36
FF0D BEQ #FF36
FF0F PLA
FF10 BIT #B801
FF13 BMI #FF10
FF15 STA #B801
FF18 PHA
FF19 LDA #B80C
FF1C AND@#F0
FF1E ORA@#0C
FF20 STA #B80C
FF23 ORA@#02
FF25 BNE #FF33
FF27 LDA@#7F
FF29 STA #B803
FF2C LDA #B80C
FF2F AND@#F0
FF31 ORA@#0E
FF33 STA #B80C
FF36 PLA
FF37 RTS
FF38 LDA #B80C
FF3B AND@#F0
FF3D BCS #FF33
FF3F LDX@#17
FF41 LDA #FF9A,X
FF44 STA #0204,X
FF47 DEX
FF48 BPL #FF41
FF4A TXS
FF4B TXA
FF4C INX
FF4D STX #EA
FF4F STX #E1
FF51 STX #E7
FF53 LDX@#33 .3
FF55 STA #02EB,X
FF58 DEX
FF59 BPL #FF55
FF5B LDA@#0A
FF5D STA #FE
FF5F LDA@#8A
FF61 STA #B003
FF64 LDA@#07
FF66 STA #B002
FF69 JSR #F7D1
  - data -
FF7C LDA@#82
FF7E STA #12
FF7E STA #12
FF80 CLI
FF81 LDA@#55 .U
FF83 STA #2901
FF86 CMP #2901
FF89 BNE #FF97
FF8B ASL A
FF8C STA #2901
FF8F CMP #2901
FF92 BNE #FF97
FF94 JMP #C2B2
FF97 JMP #C2B6
  - data -
FFB2 STA #FF
FFB4 PLA
FFB5 PHA
FFB6 AND@#10
FFB8 BNE #FFC0
FFBA LDA #FF
FFBC PHA
FFBD JMP(#0204)
FFC0 LDA #FF
FFC2 PLP
FFC3 PHP
FFC4 JMP(#0202)
FFC7 PHA
FFC8 JMP(#0200)
FFCB JMP(#021A)
FFCE JMP(#0218)
FFD1 JMP(#0216)
FFD4 JMP(#0214)
FFD7 JMP(#0212)
FFDA JMP(#0210)
FFDD JMP(#020E)
FFE0 JMP(#020C)
FFE3 JMP(#020A)
FFE6 JSR #FFE3
FFE9 CMP@#0D
FFEB BNE #FFF4
FFED LDA@#0A
FFEF JSR #FFF4
FFF2 LDA@#0D
FFF4 JMP(#0208)
FFF7 JMP(#0206)
FFFA ---
FFFB ---
FFFC ---
FFFD ---
```

CHAPTER 6
WORKING EXAMPLES USING THE ROM ROUTINES

For normal interpreting use there are six major subroutines that are most useful:

1. C8BC - Read (5),Y to the workspace stack.
2. C231 - Expect and skip past a "," sign.
3. C589 - Print the w/s stack in decimal.
4. C349 - Print the w/s stack in hex.
5. CD09/F - input with editing to an input buffer.
6. F7D1 - machine code version of PRINT"....." .

Further, the best way to end any m/c code routine is JMP #C55B, rather than using RTS. The examples below use these and other routines to illustrate how they can be incorporated into you own systems.

1) To print out messages on the screen .

```
100 DIM P-1
110 M=P
120 [;JSR #F7D1;]        CALL IN-LINE PRINTER
130 $P="THIS IS A MESSAGE"
140 P=P+LEN P
150 [; NOP               TERMINATE PRINTER WITH A NEGATIVE
                         CHARACTER SUCH AS "NOP"
160 JSR #FFED            EXECUTE CR+LF
170 RTS ; ]
180 DO;LINK M; UNTIL 0      TEST IT OUT
```

2)To copy a value on the w/s stack to an integer variable.

```
100 DIM P-1;M=P;[
110 LDY@ CH"N"-40            COPIES W/S STACK VALUE IN
120 LDX@  #FF                #16,25,34,43 TO INTEGER
130 JSR  #CA37 ; ]           VARIABLE N
140 ?16=9 ; LINK M ; PRINT N ; E.
```

3)To print out the value of one of the integer variables.

```
100 DIM P-1;M=P;[
110 LDY@ CH"N"-40        FETCH VARIABLE N TO THE
120 LDX@ 1
130 JSR #CE83            WORKSPACE STACK.
140 JSR #C589 ; ]        PRINT W/S STACK AS DECIMAL
150 LET N=20;LINK M;E.
```

4)For those with DISATOM, using [X] to pass on a number that fills the screen.

```
10 DIM JJ1;JJ0=-1;JJ1=-1
20 FOR X=0 TO 1           TWO PASSES
30 P= #3B00               ASSEMBLE AT  3B00
40 [                      START ASSEMBLING
50 JSR #C8BC              READ VALUE AFTER [X] TO W/S STACK
60 JSR #C4E4              CHECK FOR RUBBISH,<CR> OR ;  OK
70 LDA @ 0 ; STA 4        RESET W/S STACK POINTER
```

-continued-

```
80 LDA  #16 ; LDX @ 0     PUT VALUE INTO ALL SCREEN RAM
90:JJ0
100 STA #8000,X
110 STA #8100,X
120 INX ; BNE JJ0
130 JMP #C55B ; ]         BACK TO INTERPRETER
140 NEXT ; END
```

N.B.- The [X] command must be spaced away from the line number if it is the first command in a line, or the interpreter will mistake it for a label. All [X] routines must end in JMP C55B.

A BASIC program to use the above m/c code is:

```
10 ! #180= #3B00
20 F. A=0 TO 255
30 [X] A
40 F.I=1 TO 60;WAIT;N.
50 N.A
50 E.
```

5. To INPUT numbers into your routines.

```
100 DIM P-1;M=P;[
110 JSR #CD09             INPUT WITH EDITING TO #140 BUFFER
120 LDY@ 1 ; STY 6         120-140,POINT (5),3 AT #140
130 DEY ; STY 3
140 LDA@ #40; STA 5
150 JSR #C8BC             READ #140 BUFFER TO W/S STACK
160 JSR #C589             PRINT W/S AS DECIMAL IN FIELD @
170 RTS ;]
180 LINK M ; E.           TEST IT
```

NOTE: This input allows decimal or # prefixed hexadecimal. Repeated calls to C8BC should be prefixed with LDA@ 0;STA4 to reset the w/s stack. Unless (5),3 is PUSHed before entry to this routine, then PULLed at the end, it will exit to direct mode.

6. To INPUT Hex numbers into your routines.

```
100 DIM P-1; M=P ; [
110 LDA@ CH"#"            PROMPT WITH CHARACTER #
120 JSR #CD0F             INPUT WITH EDIT TO #100 BUFFER
130 LDY@ 0                RESET Y
140 LDX@ #80              READ #100 BUFFER AS HEX, STORE TO
150 JSR #F893             VECTOR X POINTS AT- HERE #80
160 JSR #F7F1             PRINT VECTOR X POINTS AT AS HEX
170 RTS ; ]
180 LINK M ; E.           TEST IT
```

NOTE: F893 stores the 100 buffer as a two-byte vector in Page 0, which is pointed at by X on entry to the routine. The accumulator is stored in the third byte, so P.! #80 gives a strange result.

7. Hex Printer

```
100 DIM P-1;M=P;[
110 JSR #CD09              INPUT WITH EDIT TO #140 ? PROMPT
120 LDY@ 0 ;STY 3          SET UP VECTOR (5),Y WHERE
130 INY ; STY 6            Y=?3
140 LDA@ #40 ; STA 5       TO POINT AT #140
150 JSR #C8BC              READ (5),Y TO W/S STACK
160 JSR #C349              PRINT W/S STACK IN HEX
170 RTS ; ]
180 LINK M; E.             TEST IT
```

8. Inverting the screen.

```
10 DIM JJ2;F.I=0TO2;JJI=-1;N.;F.X=0TO1;P= #2800;[
20:JJ0 LDY@ 0; JSR #FE66          SYNC TO TV FLYBACK
30:JJ1 LDA #8000,Y
40 EOR@ #80 ; STA #8000,Y         DO TOP OF SCREEN
50 INY ; BNE JJ1
60 JSR #FE6B                CHECK STILL IN FLYBACK OR WAIT
70:JJ2 LDA #8100,Y
80 EOR@ #80 ; STA #8100,Y         DO LOWER SCREEN
90 INY ; BNE JJ2
100 RTS ; ]
110 NEXT X
120 DO; LINK JJ0                  TEST IT
130 F.X=1TO30;WAIT;N.
140 UNTIL 0
```

9.Unsigned Multiply : Executes (R)=(M)*Acc .

```
10 R= #80                   2-BYTE RESULT
20 M= #82                   2-BYTE MULTIPLIER
30 DIM JJ2;F.I=0TO2;JJI=-1;N.;F.X=0TO1;P= #2800;[
40:JJ0 PHA
50 LDA@ 0;STA R;STA R+1
60 PLA ; LDX@ 8
70:JJ1 CLC
80 ROL R ; ROL R+1
90 ASL A ; BCC JJ2
100 PHA ; CLC
110 LDA R ; ADC M ; STA R
120 LDA R+1; ADC M+1 ; STA R+1
130 PLA
140:JJ2 DEX ; BNE JJ1
150 RTS ; ]
160 NEXT X
170 ! M= #100;A= #B       TEST IT
180 LINK JJ0
190 PRINT &(! R&#FFFF);E.
```

10. Unsigned divide : executes (D)=(D)/V

```
10 D= #80                                  2-BYTE DIVIDEND
20 V= #82                                  1-BYTE DIVISOR
30 R= #83                                  1-BYTE REMAINDER
40 DIM JJ5;F.I=0TO5;JJI=-1;N.;F.X=0TO1;P= #2800;[
50:JJ0 LDA2 0; STA R
60 LDX@ #11; BNE JJ2
70:JJ1 SEC
80 LDA R ; SBC V ; BPL JJ3
90:JJ2 CLC ; BCC JJ4
100:JJ3 STA R ; SEC
110:JJ4 ROL D ; ROL D+1
120 DEX ; BEQ JJ5
130 ROL R ; JMP JJ1
140:JJ5 RTS ;]
150 NEXT X
160 ! D= #400 ; ?V=#21                    TEST IT
170 LINK JJO
180 PRINT &(! D&#FFFF) , ?R
190 END
```

11. Cyclic Redundancy Check (CRC). Has many uses, but for example, if the CRC is known for a Program, it should give the same result again after reloading from tape. See Chapter 7 for application.

```
100 DIM JJ4;P.$21
110 F.I=0TO4;JJI=#FFFF;N.
120 F.I=1TO2;DIMP-1;M=P;[
130 JSR #F7D1;]
140 $P="START ADDR ";P=P+LENP;[
150 NOP
160 LDA@ CH"#";JSR #CD0F
170 LDY@ 0;LDX@ #90;JSR #F893
180 JSR #F7D1;]
190 $P="  END ADDR ";P=P+LENP;[
200 NOP
210 LDA@ CH"#";JSR #CD0F
220 LDY@ 0;LDX@ #92;JSR #F893
230 LDY@ 0;STY #A0;STY #A1
240:JJ1 JSR JJ2
250 LDX@ #90;JSR #FA08
260 BNE JJ1
270 JSR JJ2
280 JSR #F7D1;]
290 $P="SIGNATURE IS ";P=P+LENP;[
300 NOP
310 LDX@ #A0;JSR #F7F1;JSR #FFED
320 JMP #C55B
330:JJ2 LDX@ 8;CLC
340 LDA(#90),Y
350:JJ3 LSR A;ROL #A0;ROL #A1;BCC JJ4
360 PHA
370 LDA #A0;EOR@ #2D;STA #A0
380 PLA
390:JJ4 DEX;BNE JJ3
400 RTS
410 ];P.$6;P."M/C CODE IS AT "M;LINK M;E.
```

CHAPTER 7
TAPE FILES, CRC , AND PRINTER USAGE

THE TAPE:

The ATOM normally stores information to tape at 300 BAUD. Some chips on the market, such as DISATOM, allow 1200 BAUD, but in all cases the format of the files are the same. It is useful to study this format in case there is some corruption of the tape that prevents loading. The bulk of the information can often be recovered.

There are three types of SAVE command used in the ATOM 1)*SAVE named file 2)SAVE named file 3)*SAVE unnamed file. The ATOM manual gives details of how these are used. In the first two cases the block header format is identical. The diagram below represents the individual bytes on the tape header for a file called ADVENTURE which will begin at 2900, finish at 3BFF, and have a GO (*RUN) address of 3B50.This file has been *SAVED as a named file using *SAVE"ADVENTURE"2900 3C00 3B50.

*	*	*	*	A	D	V	E	N	T	U	R	E

0D	E3	00	00	FF	3B	50	29	00

As can be seen, the operating system always places four stars in front of the file name. if any of these stars are corrupted the file cannot be loaded. The title of the file can be up to 13 characters (bytes) long, and so the actual length of the header is variable depending on the size of the title. It can be as short as 14 bytes, or as long as 26. The title is always terminated by 0D (Carriage Return). It is possible to get up to some real tricks with the title (see PROGRAM PROTECTION).

The next byte is the Header Checksum, to insure that the header itself has not been corrupted.

The next two bytes are the Block Number, which is given during a *CAT. The first block in a file is always numbered zero (By the way- you can abbreviate *CAT as simply *. and it works fine).

The next byte on the header holds the number of bytes in this block of information (excluding the header itself and the checksum). Normally this is FF, since the block contains a full page of memory. However, it may be less than FF if either 1)you save a very short program, or 2) it is the last block in a file that does not finish at the end of a page.

The next two bytes are the GO address. If you were to RUN the program, the operating system would automatically jump to this address and begin executing the machine code that should be there. In our example the address is 3B50.

The final two bytes of the header is the location where this block will be placed. For BASIC programs this is normally 2900 for the first block, filling up from there. Of course you may change this in either the SAVE or LOAD commands. Since our example block is FF bytes long, it will be loaded into the memory beginning at 2900 and finishing at 29FF.

The last byte of any block is the CHECKSUM, which includes the header and the program proper, but not the checksum itself. As the tape is read in the operating system executes ?DC=?DC + X, where X is the byte being read. It then compares ?DC with the checksum at the end, and gives SUM ERROR 6 if they do not match. Since this is not a true Cyclic Redundancy Check, it is possible to get no SUM error if there are errors which exactly cancel out, and the program will be loaded but will be corrupt.

If we had saved this file using the BASIC command SAVE"ADVENTURE" the header would be of exactly the same format, but BASIC would fill in the missing details of the title before actually saving it. Thus it would find the value of TOP, and would save to tape all memory from (?#12), which contains a pointer to the bottom of the program, to TOP. It would use C2B2 as the GO address, which when executed just places you in Command Mode. This would be catatrophic for our example, since it contains machine code AFTER the BASIC part of the program, and is designed to have this accomplished starting at 3B50. This is quite a common fault when people copy programs. If there is any machine code that is not within the BASIC program, or written by it in the course of execution, then it is not saved, and the copied program will fail.

The Unnamed file is the fastest way to save memory, but does not have any checksums, and the header is extremely brief. Since the memory is not divided up into blocks, the information is as one continuous stream, and the header is needed only once. If our example were saved thus: *SAVE 2900 3C00 , the header would be

3C	00	29	00

and that's all.

If a tape is corrupted, it is possible to write machine code routines that bring the entire contents of the tape, including the Header and Checksum, into memory (or use the TAPEXXXX function on DISATOM). It is stored in a temporary area, such as 8200. The memory at that area is then inspected, and the block of FF bytes of actual program is then COPYied to its corect address, say at 2900. Let us assume we captured the corrupt first block of our example above at 8200. Since the actual program begins at 8217 we would then type COPY #8216,(#8216+ #FF), #2900 . This would put the first block in its rightful place, but has left behind the tape header and checksum. It does not of course insure that there is no corruption in the program itself.

CRC FOR THE ATOM

CRC is short for 'Cyclic Redundancy Check'. There is no real need to understand the mathematical theory of why it works, but it is useful to see how its works, and we'll deal with this later. It can be especially important to ATOM owners, since we have no CRC on the tape input routine, and it is thus possible to load a program in without getting an error message, but in fact there is an (undetected) error. This is because the tape header stores a checksum that is just the sum (modulo 256) of all the bytes in that block, and so it is possible to get two (or more) errors that exactly cancel each other by giving the same sum as the correct version. There are really two check bytes, one for the tape header itself, and one for the block of information.

Most machines use a true CRC check, and so the chances of getting an undetected error are very much smaller (indeed almost 0) than for a simple sum check. Further, since the check is in

ROM as part of the operating system, it is never lost on power-down. The best that ATOM users can do is to 'hide' a CRC in an area of RAM that is not normally used, but of course this will have to be reloaded each time the machine is powered up.

What is the advantage of this CRC? Well, just this - most programs are resident from address #2900 to #3BFF in the expanded ATOM, and once a program is SAVEd to tape there is no way to load it back and run it without destroying the original (assuming the program uses the graphics area). Therefore, if there was an error on the taped version, you have lost the original by over-writing it. Now if you had , say, a BBC machine you could have sent your program to tape then LOAD it back into a ROM area. Of course the program will not actually be remembered by the computer as you can't write to the ROM. However, the point is that as the program is read from tape it is checked with CRC. If we get no errors we can thus be assured that it was saved correctly. If we do get an error, we still have the original in RAM, and so can save it again.

Using the CRC program below, it is also possible to do this with the ATOM, but is slightly more laborious. The procedure is this:

i. Load in the CRC program to an out-of-the-way area.
ii. Write or load a program into the normal text area.
iii. Save your main program to tape.
iv. *LOAD your program back, starting at #8200.
v. Run a CRC on both versions of the program.

If CRC gives the same result, you can be assured that the programs are identical, and so you have correctly saved it.

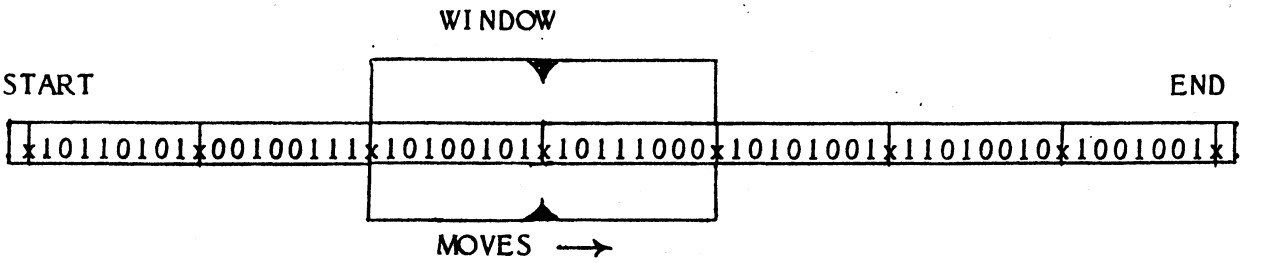
But what if they are not identical? This is harder to work out. Here are the possible reasons:

1. The program was correctly saved to tape, but there was an error in reloading (recorder volume wrong etc.)
2. The program was correctly saved to tape and correctly loaded back, but there is a fault in RAM (rare).
3. The program was not correctly saved to tape (usually a fault of the tape material or recorder).

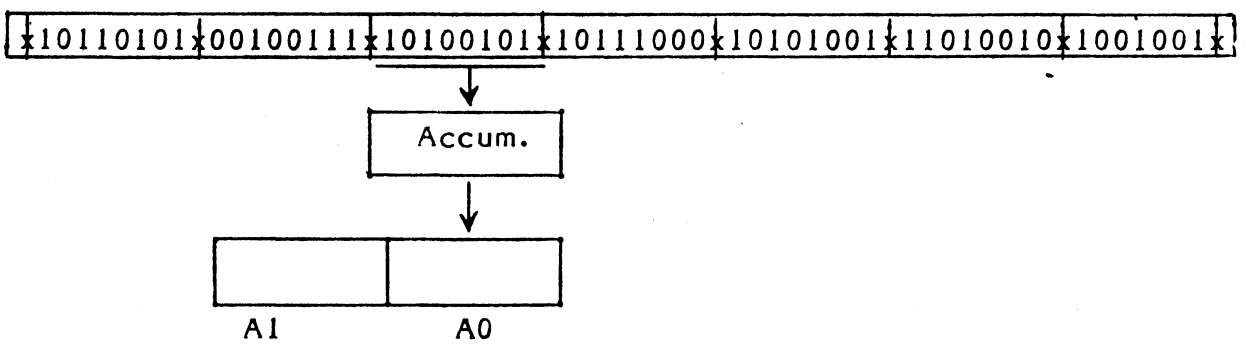
You must now go through various diagnostic procedures to find out just what the problem is. This is the rub. CRC is excellent at telling you that things are not right, but tells you nothing about where the error is. You can of course be lucky and have an error where it doesn't make any difference anyway (such as in a REM statement)! One of the few things that can be done with CRC is to divide the program in half and use CRC on each half, then repeat this until the error is located (a binary search method).

HOW CRC WORKS

Imagine any area of memory as a long tape, on which is printed a series of 0's and 1's . These numbers are organised into blocks of 8. Each 0 or 1 is called a bit, and each block of eight bits is called a byte. Now imagine that you had this tape in front of you, and that you had a square of card with a 'window' cut in it , so that you could view 16 bits (2 bytes) at a time:



Start moving the window to the right. Each time a 1 appears off the left side of the window, EOR the right side 8 bits with #2D. When the window bumps up against the end, the number left in it is the 'signature' of that area of memory. In practice, we will use locations #A0,A1 as the window, and the accumulator is used to put the next 8 bits of memory into the window. Doing it in this way, the memory itself is not disturbed.



Locations #90,91 will be used to 'point' at the area of memory under scrutiny, and #92,93 to hold the address of the END.

LOCATING THE CRC PROGRAM

So far as we know, the memory area from #3CA to #3FC is free, and so is the area from #21C to #23F. It is possible to just squeeze a CRC program into these areas by putting the input and control part at #3CA, and the main subroutine at #21C. We have tested these areas out, and so far neither the operating system nor application programs have 'stomped' on them.

THE SOURCE PROGRAM

This program uses ROM calls that are described in 'Splitting the ATOM', and sets up the DISATOM command [X] to point at it.

Code	Remark
`10 DIM JJ4;P.$12,$21;! #180= #3CA`	Set up labels, screen off, Point DISATOM
`20 F.I=0 TO 4;JJI=-1;N.`	Clear labels
`30 F.I=1 TO 2;P= #3CA;[`	Two passes, put this at #3CA, START assembler
`40 LDA@CH"S";JSR #CD0F`	Prompt S, in. start adrs
`50 LDY@ 0;LDX@ #90;JSR #F893`	Store it at #90,91
`60 LDA@CH"E";JSR #CD0F`	Prompt E, in. END adrs
`70 LDY@ 0;LDX@ #92;JSR #F893`	Store it at #92,93
`80 LDY@ 0;STY #A0;STY #A1`	Wipe the window
`90:JJ1 JSR JJ2`	Control area, moves the
`100 LDX@ #90;JSR #FA08`	window from start to end
`110 BNE JJ1`	
`120 JSR JJ2`	We've hit the end, so
`130 LDX@ #A0;JSR #F7F1`	Print window
`140 JMP #C55B;]`	Back to BASIC
`150 P= #21C;[`	Assemble at #21C
`160:JJ2 LDX@ 8;CLC`	Set up for 8 Bits
`170 LDA(#90),Y`	Get a byte from memory
`180:JJ3 LSR A;ROL #A0;ROL #A1;BCC JJ4`	Push it into the window
`190 PHA`	If a 1 fell off, do this:
`200 LDA #A0;EOR@ #2D;STA #A0`	EOR the piece of window
`210 PLA`	
`220:JJ4 DEX;BNE JJ3`	Next bit
`230 RTS`	Back to control area
`240 ];N.;P.$6"ASSEMBLEY COMPLETE";E.`	Screen on, end assembly.

Since this source code is in BASIC you can SAVE it in the usual way as "CRCSOURCE" after having RUN it. The machine code is now at #3CA and #21C, so you have a choice of either Saving #21C to #3FF as one big block (most of which isn't wanted), or alternatively save the two areas #21C to #23F and #3CA to #3FF as separate blocks. Only shutting off the machine will remove the machine code, so you are safe after hitting <BREAK>.

USING THE PROGRAM

If you have a DISATOM ROM fitted, you need only type [X] after running the source code. When reloading the m/c code, type

```
! #180= #3CA
```

and this will point DISATOM's [X] at the routine again. For those without the chip, type LINK #3CA each time you want CRC. The letter S (meaning Start) should appear on the screen. Type in the four figure HEX address where you want CRC to begin, then hit <RETURN>. CAUTION! - there was not enough room for input error checks, so that while you are allowed to edit your input before hitting <RETURN>, you cannot do so afterwards. An E (for END) now appears on the screen. Type in the four figure HEX address of the last byte you want checked, and hit <RETURN>. Within a few seconds the four figure HEX 'Signature' of that ara of memory appears on the screen. From your ATOM manual page 93, you will see that a BASIC program of this type takes many minutes, so we have a big time saving in addition to everythng else. Try these tests on your resident ROMs to confirm correct function of the program:

ROM Name	Start	End	Signature
Integer BASIC	C000	CFFF	D67D
Integer BASIC	F000	FFFF	E386
Floating BASIC	D000	DFFF	AAA1

If you have a COPY function such as the one in DISATOM, you can also use CRC to test RAM. Do this by COPYing one area of RAM to another, then checking both areas with CRC, which should give the same signature. As already mentioned, you can dump a program to tape then *LOAD it to #8200 and use the CRC to confirm correct saving. With this confirmation ability, we have taken to writing down the CRC signature next to the title of the program, and SAVEd our programs as UNnamed files. This gives a great reduction in of loading time. Further, if you have a 1200 Baud SAVE/LOAD facility such as in DISATOM, you can use unnamed 1200 files. It is now possible to load in a big games program extending from #2800 to #3BFF in just 40 seconds and be assured of a correct load!

THE PRINTER:

The ATOM is initialised such that line feed characters (0A) are not sent to the parallel printer port used for operation of a Centronics-type printer. It assumes that the printer has been configured to give an auto-line feed on receiving a carriage return (0D).

Where this is inconvenient, the ATOM can be made to pass the line feed character by setting ?FE=FF . The address location FE normally contains the character which will NOT be sent to the printer, and setting it to FF will ensure all ASCII codes and characters are transmitted.

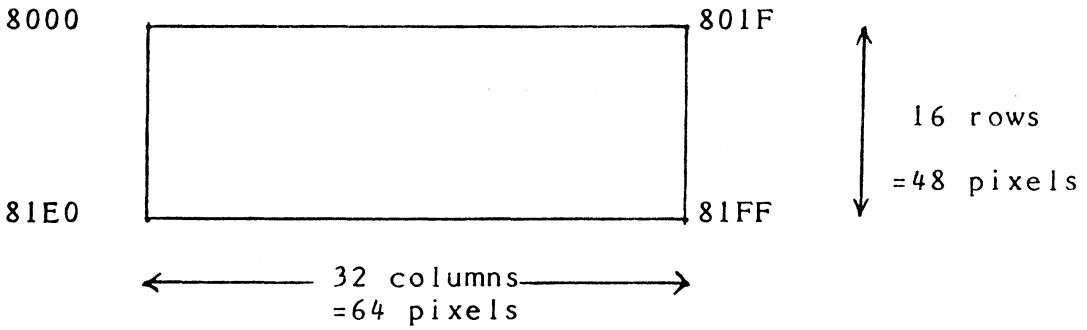
You can check whether the printer is connected or not by testing bit 7 of B800 (handshake signal). You can then avoid locking up the machine, by executing $2 only after a positive handshake test.

CHAPTER 8
THE MEMORY MAPPED V.D.U.

This section is intended for reference, and shows how the V.D.U. screen is memory mapped in each display mode. At the end of the chapter a map for each display mode can be found.

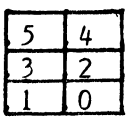
1. TEXT MODE / GRAPHICS MODE 0

In this mode the VDU display is mapped from 512 memory locations (0.5K) in the format 32 across and 16 down.

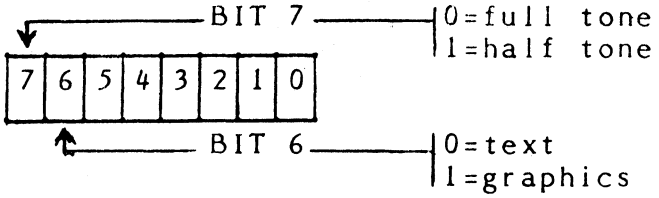


In text mode this allows 32 columns by 16 rows of text characters to be printed. In graphics mode it allows 64 columns by 48 rows of pixels to be individually accessed. Each memory location, or graphics cell, is divided into 6 pixels corresponding to the 6 lower bits of the number stored in that location.

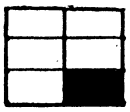
When the bit is SET (=1) the pixel is white (or grey). When the bit is CLEAR(=0) the pixel is black. If the last bit(bit 7) of the number is set, the graphics are changed from white to grey.

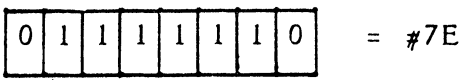


Organisation of Graphics cell



E.g.:



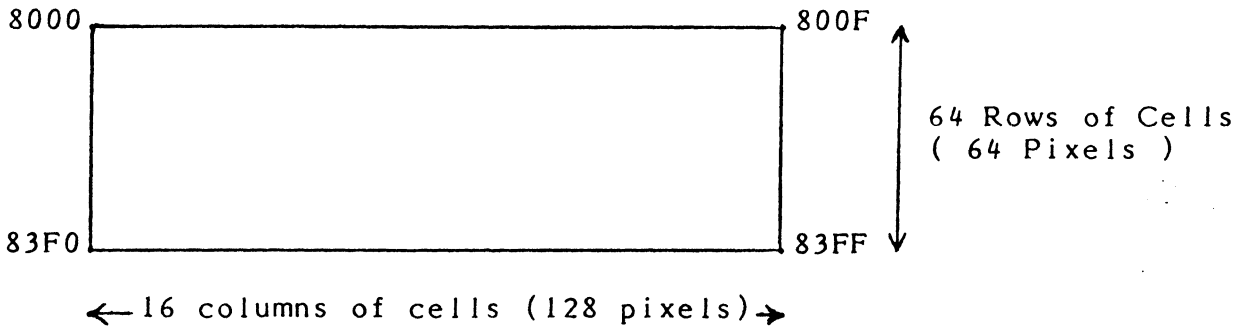


GRAPHICS MODES

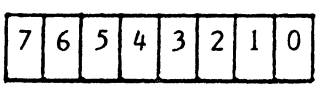
In the true graphics modes 1 to 4 the screen is bit-mapped. The VDU memory maps show the screen divided in cells, each a byte (8 bits) wide. Each bit may be either set or clear.

GRAPHICS MODE 1

In this mode the display is mapped from 1024 bytes (1K) in the format 16 cells across and 64 cells down.



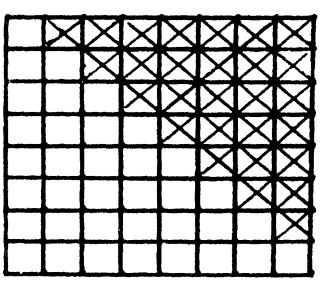
Each graphics cell has eight pixels which correspond to the eight bits in a byte (labeled 0-7).



One graphics cell

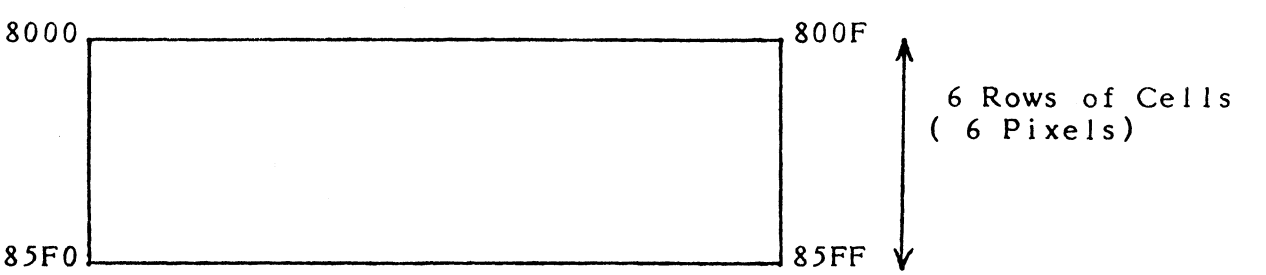
Bit label number

When the bit is set the pixel is white, and when the bit is clear the pixel is black. Thus:

Cell Shade	Binary	HEX
	1000 0000	80
	1100 0000	C0
	1110 0000	E0
	1111 0000	F0
	1111 1000	F8
	1111 1100	FC
	1111 1110	FE
	1111 1111	FF

GRAPHICS MODE 2

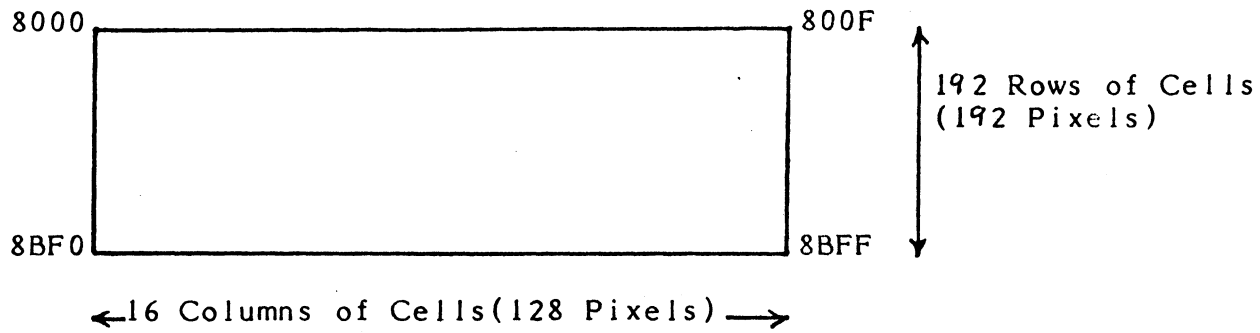
The display is mapped from 1536 bytes (1.5K), with 16 cells across and 96 cells down:



The arrangement of the graphics cell is the same as for Mode 1.

GRAPHICS MODE 3

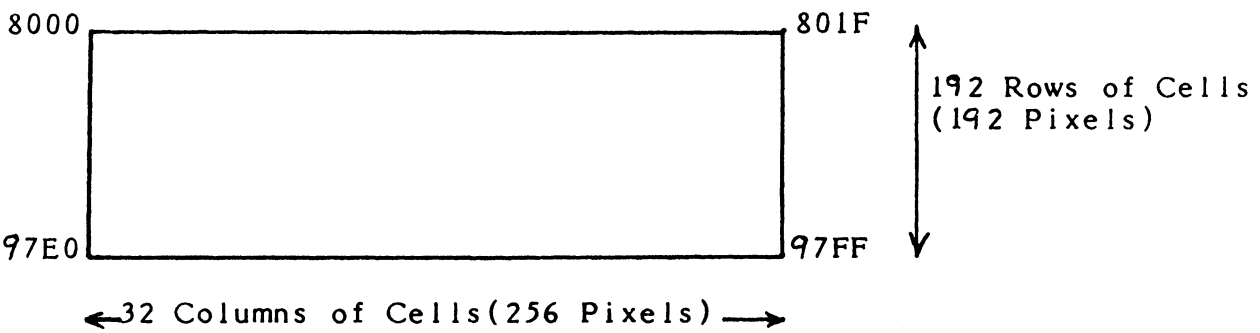
The display is mapped from 3072 bytes (3K). The format is 16 cells across, and 192 cells down.



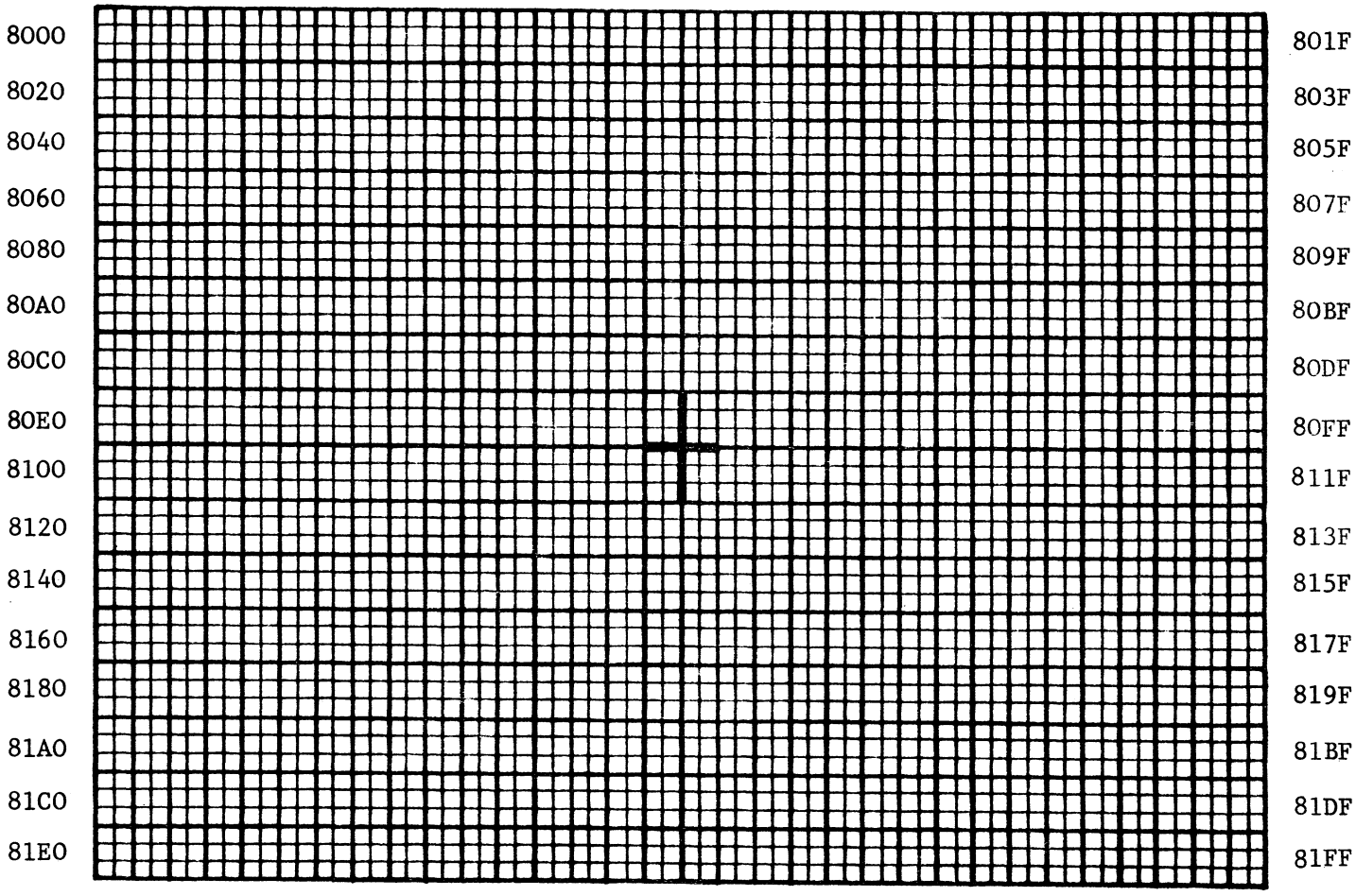
The arrangement of the graphics cell is the same as for Mode 1.

GRAPHICS MODE 4

The display is mapped from 6144 bytes (6K). The format is 32 cells across and 192 cells down.

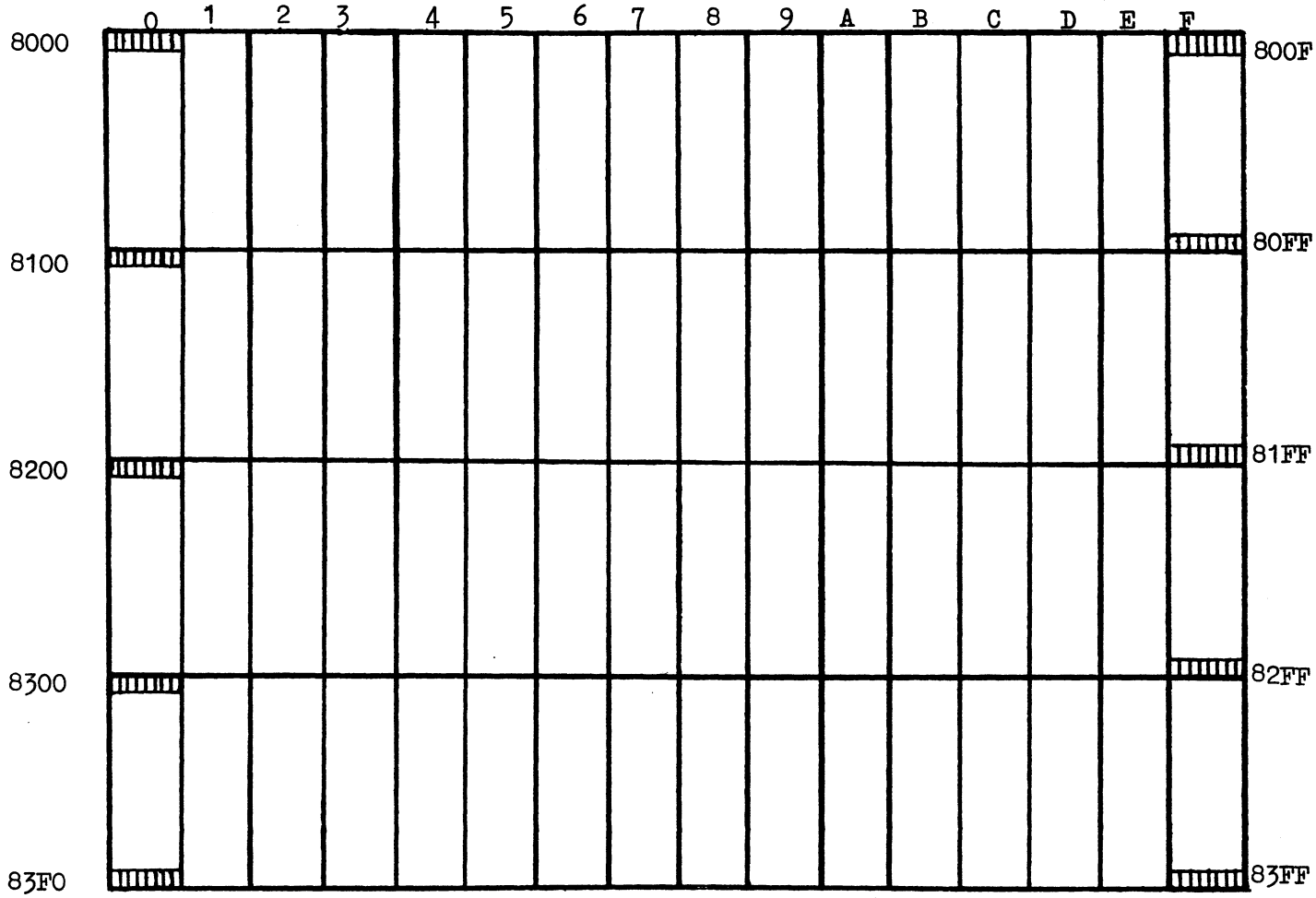


The arrangement of the graphics cell is the same as for Mode 1.



MEMORY MAPPED V.D.U.

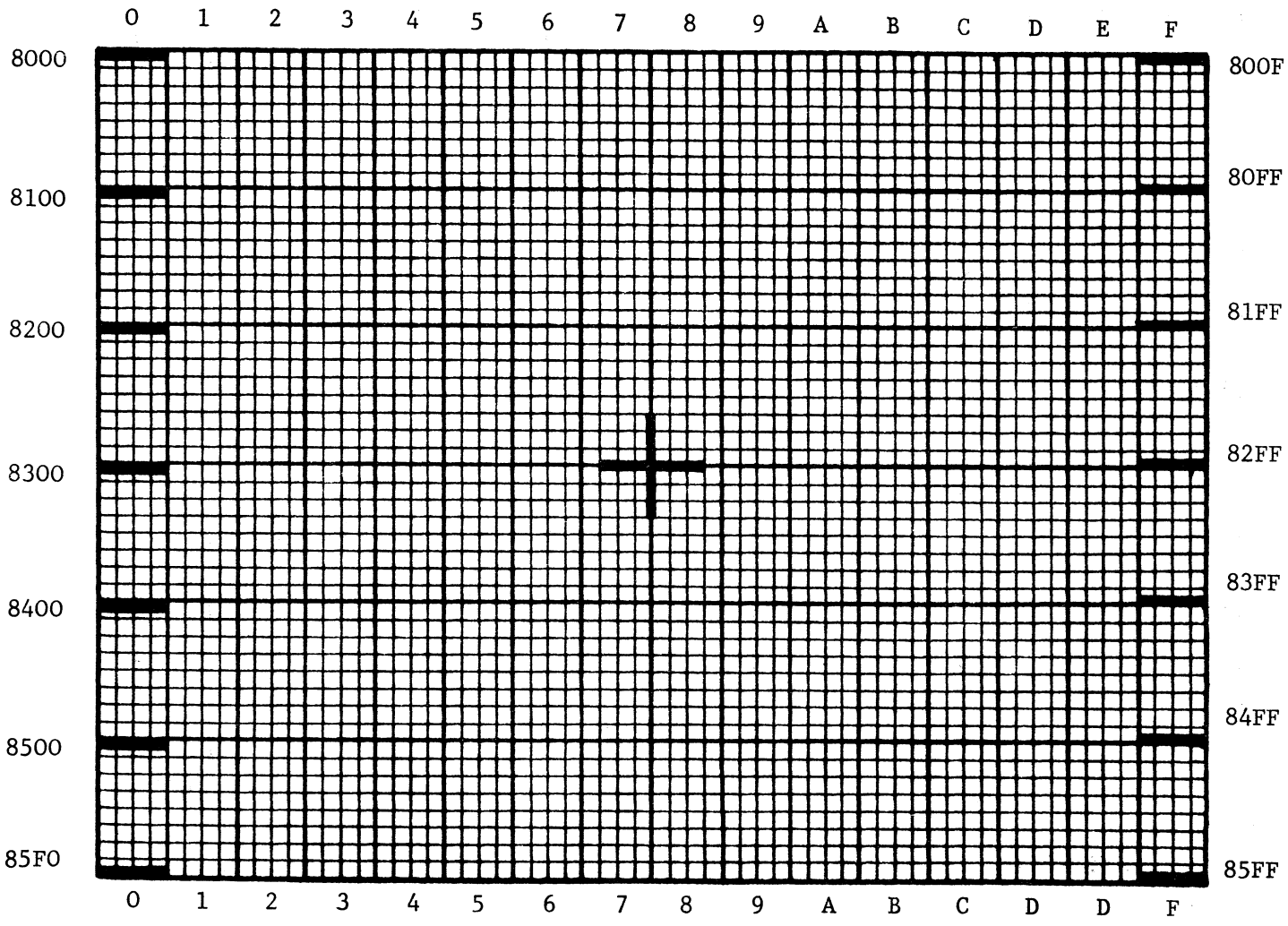
TEXT MODE/GRAPHICS MODE 0



MEMORY MAPPED VDU

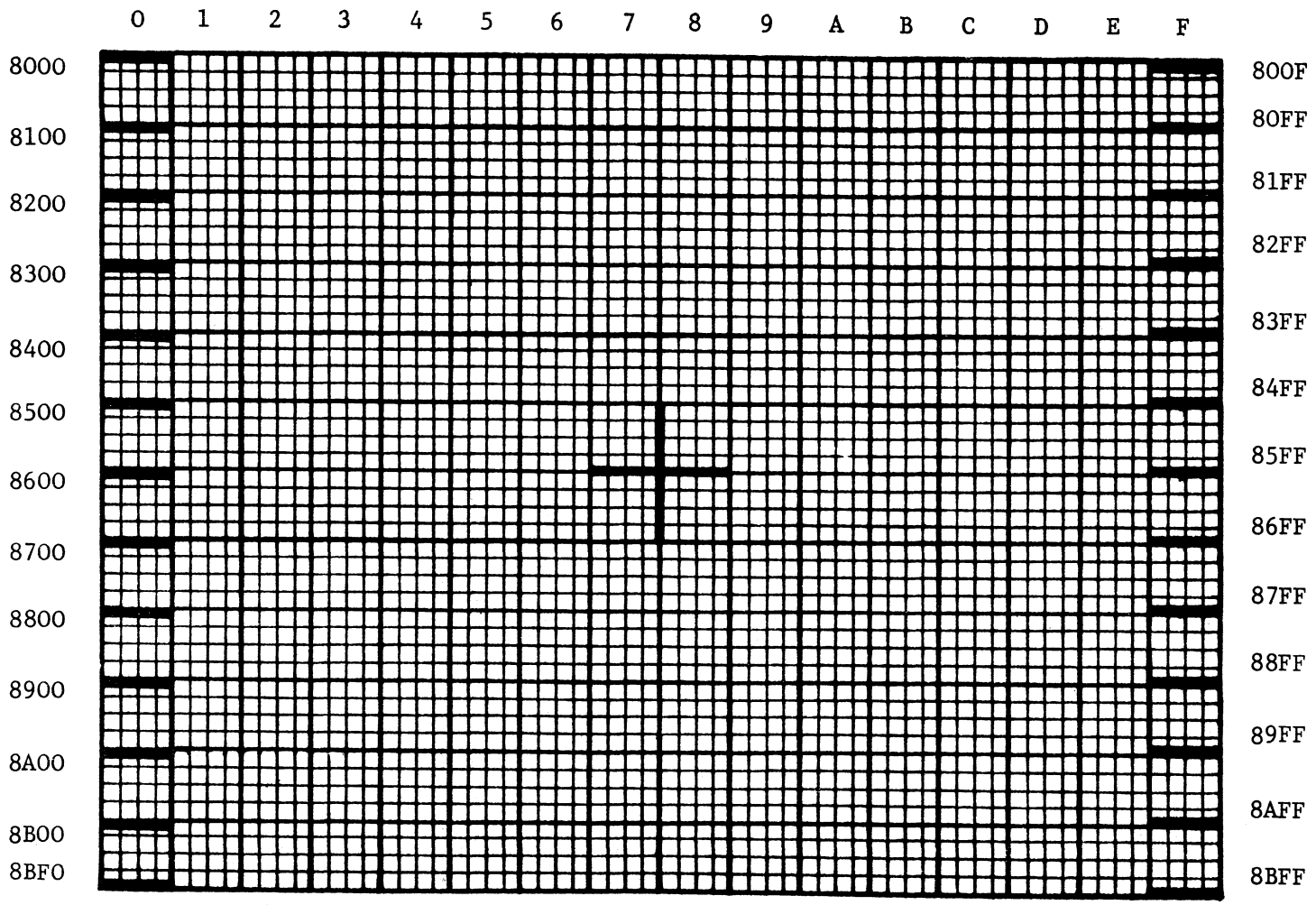
GRAPHICS MODE 1

MODE 2



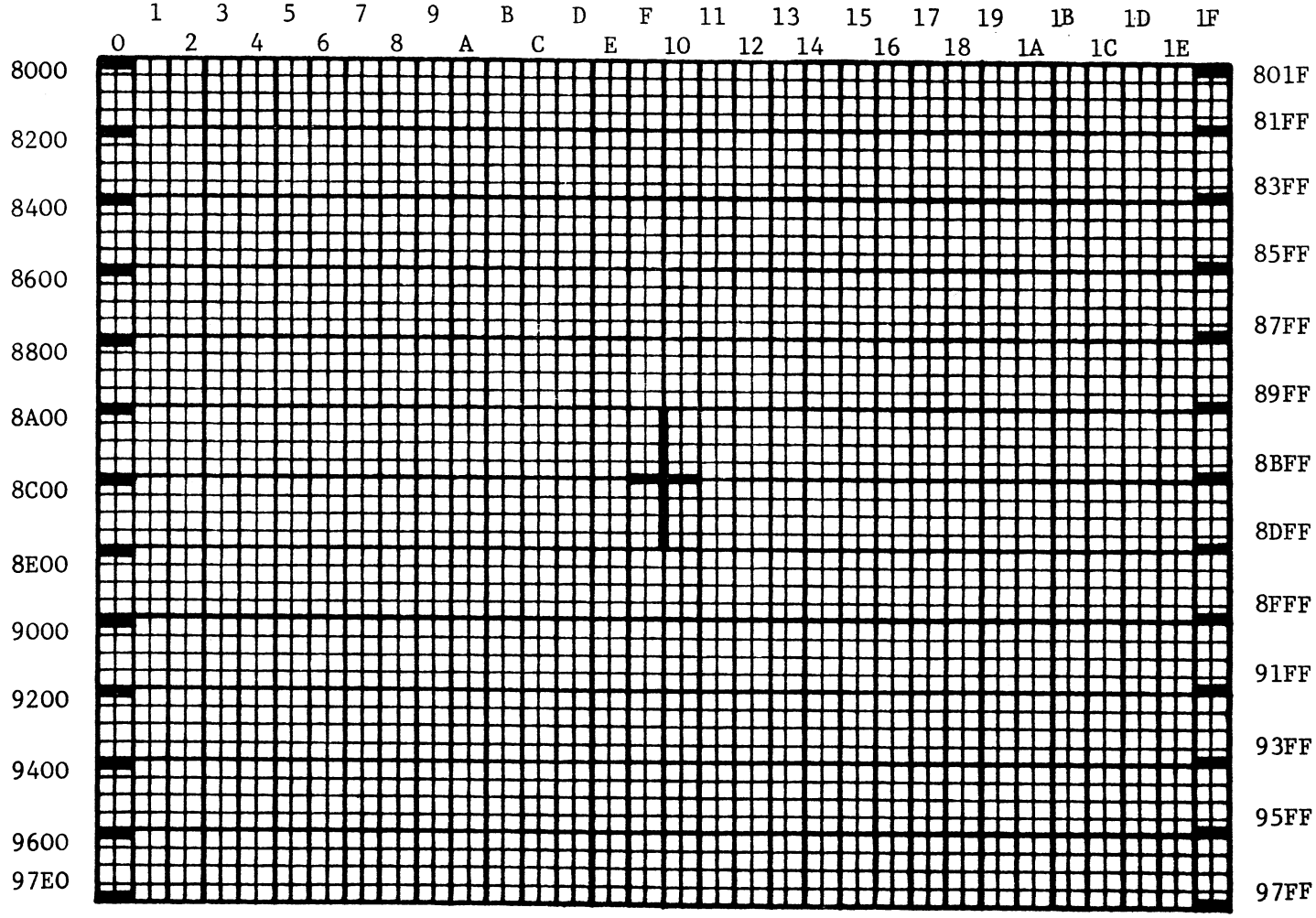
MEMORY MAPPED V.D.U.

GRAPHICS MODE 2



MEMORY MAPPED V.D.U.

GRAPHICS MODE 3



MEMORY MAPPED V.D.U.

GRAPHICS MODE 4

CHAPTER 9
METHODS FOR PREVENTION OF COPYING

Programs saved on tape by ATOM can be protected from copying by several general methods: I)Prevent the Program from being LISTed , II) when the program runs it alters some part of the machine's cassette operating system (COS) , III) load the main program by using a 'preloader' program involving some machine code. Several techniques for each of these methods are given below.

Of course, there is no way to totally protect a program other than by using a mathematical 'trap door' function, and these are unsuitable for small machines. Thus any program can be copied if a pirate has the right hardware and software , plus the skill, for the job. As with most things this is a two-edged sword, since the techniques for preventing copying are the same as for pirating.

In general, a machine-oriented chip such as DISATOM will allow any of the protection techniques given to be overcome by a skilled user. If you do not have one in your machine, see the HEX DUMP program in this book, which allows you to directly inspect and modify memory. Although this is more awkward than using a ROM, it's better than no tools at all.

The examples and techniques below are in order of increasing complexity. In all cases the following symbols will be used, and virtually ALL NUMBERS WILL BE HEX ! ! !

<....> = push this key, such as <space> or <CR> or <CTRL-C>.

[XX] = an actual byte in memory, as a HEX number.
e.g. [0D] or [03 CD 9F] .

A)Using the REM statement

1) Start the program with:

10REM<CTRL-L> <CTRL-C> <CTRL-U> <CR>

This clears the screen, turns off the printer, and turns off the VDU screen. As you type <CTRL-U> the screen is disabled, but carry on typing the line, then type <CTRL-F> <ESC> and the screen is re-enabled. Now any attempt to LIST will send the control characters behind the REM to the print stream, and they will take effect. However, a RUN is still OK, since BASIC disregards anything after the REM. It is easier to insert the control codes directly into memory using the DISATOM, but this should all be done after the program is completely perfected. For example, type in

10REM<space> <space> <space> CAN'T <space> <CR>

Then type [A] 2900<CR> <REPT> <ESC> .

This will give an ASCII DUMP of memory at 2900 as follows:

```
[A] 2900 OD OO OA .R .E .M 20 20
[A] 2908 20 .C .A .N .' .T 20 0D
```

Move the cursor up to the A 2900 and <copy> the line over to the first 20. Change this and the next 20 to 0C 03, then hit <CR> and <ESC>. In the same way edit the final 20 (before the 0D) to 15 (see the appendix on the DISATOM toolkit for further details on its use). After editing, an ASCII DUMP gives:

```
[A] 2900 OD OO OA .R .E .M 0C 03
[A] 2908 20 .C .A .N .' .T 15 0D
```

Now any attempt to LIST will clear the screen, the word CAN'T will appear in the upper left corner, and the printer and screen will be turned off.

Unfortunately, since this must be physically the first line in the program, a pirate can overcome it by simply typing

```
0 <CR>  L. <CR>
```

and if this has no effect he recovers the screen with

```
<CTRL-F> <CR>          followed by

1 <CR> L. <CR>     and so on.
```

This has the effect of eventually removing the BASIC line with the offending REM statement in it. Alternatively, if the pirate has DISATOM, he may do an ASCII DUMP and replace the 0C,03 and 15 with 20's .

2) A Rem statement can be used after a genuine BASIC statement . The REM is followed by four backspaces [08], and then some apparently legitimate BASIC statement such as X=3*A, then a [15] (screen off).The line is best set up by typing out

```
10DIM XX(12);REM <3 spaces> X=3*A <space> <CR>
```

The first three spaces [20] are then replaced by [08] (backspace), and the final [20] by [15] (screen off). When a list is attempted, the following appears on the screen:

```
10DIM XX(12);X=3*A  (screen fails)
```

The dimensioning of the array is genuine, but the second statement is a fake. The purpose of all this is to convince the user that the entire line is real, and leave him baffled as to why the screen failed. It can of course be overcome by an ASCII dump, which would reveal the REM, and it can then be removed. However, if someone attempts to delete the entire line (as in the last example) the entire program fails. You can see that there are several possible twists in this technique.

A slight sophistication is to have another REM as the last line of the program that reads

```
10000REM[06]
```

Now when a LIST is called this results in

```
>LIST
        10DIM XX(12);X=3*A
>
```

and the program appears to have only one line. The technique can again be defeated by an ASCII or HEX dump that reveals either the first or last REM.

3) Machine Code in a REM statement

Quite a lot of machine code can be put in a disguised REM statement. As with the previous example, the first part of the line is valid BASIC, but buried in the REM is some machine code to be accessed by a later part of the program. Thus

```
40X=6;?18=41;REM[7F 7F 7F 15 < m/c code here > 06 0D ]
      !! M/C CODE MUST NOT CONTAIN 0D !!
```

The first two instructions are valid and appear on LISTing. The REM causes three backspacing deletes and turns off the screen so your machine code is not seen, then turns it on again at the end. The line can be placed anywhere in the program, but the deeper in the better, since this decreases the likelyhood that someone will stumble on it with an ASCII dump. The m/c code can be anything at all, but for example, it might alter the SAVVEC of the COS system to disable the tape saving function. There are two disadvantages to this method i. you must exactly determine the entry point for the hidden machine code (then set P equal to that address and have it assemled there), and ii. you must eventually LINK to that address. Someone seeing a LINK into the BASIC text area will of course be suspicious, and in any event all LINKS can be found with a DISATOM using FIND"LINK" and FIND"LI." . It is possible to access this code via another, less suspicious m/c code routine. Using this method without the camouflage is a good way to save short machine code routines within BASIC itself, instead of having to assemble it each time, or using *SAVE to ensure machine code outside BASIC text is also saved. To prevent someone hitting <break> and then copying, site the BASIC program to start at 2800 , and then having a hidden REM that contains NOP;RTS ([EA 60]) such that the NOP is at 2900 and the RTS at 2901. On one occasion in the program proper, LINK to 2900. If a pirate breaks from the program and then copies it, the 2900,2901 machine code is lost, and the program will crash. An even more effective way is to have 28FF= JMP 28XX , and somewhere in the 2800's is another REM containing an RTS. Hitting <break> distorts the JMP location and the program crashes. Indirect jumps can also be used, via an address stored here. Once the program is running, it is easy to prevent the <ESC> key operating by intercepting the code from the keyboard and changing it. This is done by :

```
LDA@ 0 ; STA #B000      ENABLE the keyboard
JSR #FE94               GET a key in accumulator
CMP@  #D                IF a <CR> jump to DISABLE
BEQ P+8
CMP@ 32
BCS P+4                 IF >= <space> jump to DISABLE
LDA@ 32                 CHANGE code to a <space>
PHA
LDA@ 10;STA #B000       DISABLE keyboard again
PLA
RTS
```

Finally, spanner the vector at 20A,20B to point at this routine. The routine also prevents entry of any other control codes, some of which re-enable the <ESC> key.Remember to set B000=10 as the first part of your BASIC program. This can be beaten by causing an error, which will return the user to the direct mode. To be safe you should therefore alter the BRK vector at 202,203.

4) The Long Line

The BASIC interpreter is perfectly happy to work on a line which is (almost) infinitely long, with the statements being separated by semi-colons. The practical consequences of this are that i)the LIST command will turn back on itself (recycle from the start) if the line is greater than 258 bytes (two of these are the line number), and ii) If this is the first line in the program then BASIC is unable to add any new lines or delete any old ones, since it cannot find the end of the first line. If the first line consists of something like P. ;P. ;P. ; etc. etc. for the whole of page 29, then the rest of the program cannot be LISTed and the program cannot be edited, nor do the commands OLD or END work, since the real size of the program is now unknown to the operating system. The real program can be terminated with LINK #C2B2, which accomplishes a NEW, or a GOTO X , where X is a real line number or label in the program, if you wish to repeat the program. Below is a procedure for setting up such a method, and it is given so that those without DISATOM can also do it, given some extra work. Make sure that your program is perfect BEFORE you protect it, and note that you have 258 bytes less space for your real program.

1) DIRECT COMMAND:

```
F.I= #2900TO #2A04S.4; ! I= #3B202E50;N.
```

2)DIRECT COMMAND:

```
?18= #2A
NEW
```

Now write and completely debug your program as normal, but THE FIRST LINE MUST BE 1REM <3 spaces> <CR> .

3)When your program is perfect give DIRECT COMMANDS:

```
?18= #29
! #2900= #5000000D
! #2904= #3B20202E
! #2A00= #3B202E50
! #2A04= #20202E50
```

*SAVE the program in the usual way, remembering that the total program does start at 2900.

B. Disabling the SAVE

This can be done by spannering the SAVVEC at 20E,20F to point at a different location. However, this is easy to spot. A much more subtle method is to point the SAVVEC to a machine code routine that displays the "RECORD TAPE", then waits an appropriate amount of time, say 2 minutes. Of course nothing meaningful goes to the tape, but the pirate won't know this. Some examples, which will easily fit in a hidden REM are given below.

```
a)      JSR #FC40        print 'RECORD TAPE' and wait for key

b)      LDY@   05
        JSR #FB7D        each Y is worth 2 seconds, so this
        DEY              section delays 10 seconds
        BNE P-4
        RTS              TOTAL=11 BYTES.
```

Another possibility is

```
        JSR #FC40        as before
        JSR #FAF8        put a small amount of garbage on tape
                         then use part (b) from above.
```

C)Using Preloaders

This technique involves using one program to call another from tape. At the same time the first program should be accessable only via machine codes routes, should alter such things as the SAVVEC, and then destroy itself.

Both the *SAVE and *LOAD commands can be carried out from in program, and the usual prompts 'PLAY TAPE etc' avoided. This is done by changing the SAVVEC and/or LODVEC to point at your short routine (given below). Assume here that your saving routine will be at 8350, and loading routine at 8300 :

	Vector Changes	Your Program
*LOAD	20C= 00 20D= #83	PHP;JMP #F97A
*SAVE	20E= #50 20F= #83	PHP;JMP #FAF8

A BASIC program at 8000 (on the screen) can be used to perform the actual *LOAD of the main program. Programs on screen are convenient because P.$12,<BRK>,etc. erases them. They are best saved to tape by using yet another BASIC utility program, say at 8500, which i)uses the DISATOM"COPY"command to place the program on the screen, and then ii) *SAVE the screen to tape. The BASIC program on the screen should not be a runnable program, but should contain missing parts ,deliberate errors and misinformation. These will be corrected by an m/c program located at 8200

Finally, the main program should have some title which turns off the screen (to prevent *CAT), and should also make reference to some part of the m/c code preloader at 8200, so that it will not run unless the m/c code has run first. In summary, what we find is this:

ADDRESS	PROGRAM TYPE	
2900 and up	Main Program	
8000	Preloader, BASIC part	
8200	Preloader,m/c code	ONE PROGRAM
8300	*LOAD spannering program	
8350	*SAVE spannering program	
8500	BASIC utility program to save on tape the main and preloader programs.	

Given below are examples of each program.

PROGRAM TYPE:Main program LOCATION: 2900 and up
PROGRAM TITLE: "[15 03]MAIN"

```
10 ! #80= #C98046AD;! #84= #AD0FD04D;! #88= #0DC98000
   ! #8C= #15AD08D0

20 ! #90= #D000C902;! #94= #B24C6001;? #98= #C2

30 LI. #80

40 CLEAR 2 ; P.$12

50 REST OF PROGRAM FROM HERE ON
```

Lines 10 and 20 write a m/c code, and 30 LINKs to it. The purpose is to ensure that the m/c preloader program has already been run. The code reads as:

```
80=   LDA  8046 ; CMP@  4D ; BNE  96
      LDA  8000 ; CMP@  0D ; BNE  96
      LDA  0215 ; CMP@  00 ; BNE  96
      RTS
96=   JMP  C2B2
```

Finally, line 30 erases the entire preloader program.

PROGRAM TYPE: Preloader,BASIC part LOCATION: 8000
PROGRAM TITLE: "LOAD" 8000 8255 8200

NOTE: i) Deliberate errors are underlined, ii)you MUST fill in the values for the TOP vector of your main program at 13,14 in line 30 of this program (given here as XX XX).

```
[0D FF 00] ! #80=#03902000;! #84= #4C81FF8D
   ! #88= #FBEE;! #214= #FC7C0080

20 *LOAD"<3 spaces>AIN"

30 ?18=41;?13=XX;?14=XX;LI. #FE86
```

This program should originally be written with the first line numbered as,say, line 5. You will later change this with DISATOM so that the area of memory storing this line number will read [0D FF 00], where it used to read [0D 00 05]. This is of course creating an error, but the m/c code part of the preloader will correct this to [0D 00 00], which BASIC interprets as the start of the program, line 0 . The effect of the line when it runs is to set up m/c code at #80, and then spanner the tape byte-getting routine to point at it. This causes the bytes loading from tape to be appear in the lower right of the screen, thus visually confirming further LOADs, and then passes the byte to the correct location. Line 20 LOADs the main program into 2900. The title starts out as "<3 spaces>AIN". The m/c code preloader will change the 3 spaces so the title reads "[15 03]MAIN". The title is therefore actually "TURN OFF SCREEN,TURN OFF PRINTER,MAIN", and so it cannot be *CATed, nor can the title or the rest of the program be LISTed once it is altered to its final form. Line 30 changes the BASIC text pointer to 2900,sets the value of TOP for the main program (you MUST provide this), then starts the main program at 2900. NOTE-if the main program contains DIM statements, you must first set the DIM POINTER (at #23 , 24) to the value of top, somewhere in the main program before the DIM statement.

The previous BASIC program has several deliberate errors to hinder copying and relocation. Here is a summary.

ERROR BYTE	AT	BECOMES	COMMENT
FF	8001	00	BECOMES LINE NUMBER 0
30	800F	43	ALTERS M/C CODE
30	8010	39	from 00 to C9
20	8044	15	CONVERTS PROGRAM LOAD TITLE
20	8045	03	to
20	8046	4D	"[15 03]MAIN"
46	806C	43	ALTERS LINK from FE86 to CE86

PROGRAM TYPE: Preloader, m/c code part LOCATION: 8200
PROGRAM TITLE:same as BASIC part, all saved as one program.

LOCATION	SOURCE	OBJECT	REM
8200	LDA@ 08	A9 08	WRITE M/C CODE
	STA #8300	AD 00 83	TO ENABLE
	LDA@ #4C	A9 4C	*LOAD BY BASIC
	STA #8301	AD 01 83	PRELOADER.
	LDA@ #7A	A9 7A	
	STA #8302	AD 02 83	
	LDA@ #F9	A9 F9	
	STA #8303	AD 03 83	
8214	LDA@ 00	A9 00	FIX DELIBERATE
	STA #8001	AD 01 80	ERRORS IN BASIC
	LDA@ #43	A9 43	PRELOADER
	STA #800F	AD 0F 80	
	LDA@ #39	A9 39	
	STA #8010	AD 10 80	
	LDA@ #15	A9 15	
	STA #8044	AD 44 80	
	LDA@ #03	A9 03	
	STA #8045	AD 45 80	

```
                LDA@ #4D        A9 4D
                STA #8046       AD 46 80
                LDA@ #43        A9 43
                STA #806A       AD 6A 80
8237            LDA@  00        A9 00           SPANNER LODVEC
                STA #020C       AD 0C 02        TO POINT AT
                LDA@ #83        A9 83           OUR PROGRAM AT 83
                STA #020D       AD 0D 02
8241            LDA@ #40        A9 40           SPANNER SAVVEC
                STA #020E       AD 0E 02        TO PRINT MESSAGE
                LDA@ #FC        A9 FC           THEN FAIL
                STA #020F       AD 0F 02
824B            LDA@ #80        A9 80           SPANNER TEXT
                STA  #12        85 12           POINTER AND JUMP
                JMP #CE86       4C 86 CE        TO BASIC
LAST BYTE AT 8251.
```

How to construct the entire preloader program:

i)Use the DIRECT COMMANDS

```
F.I= #8200 TO #8600; ?I=32;N.
?18= #82
NEW
```

ii)Type in the BASIC part of the preloader, as mentioned above, including the errors. Start with line 5. When finished, type NEW, then alter the program title in line 20 using either HEX DUMP or a DISATOM ASCII dump.

iii)Use the DIRECT COMMANDS

```
?18= #85
NEW
```

iv)Construct the SOURCE code given for the m/c code part of the preloader here at #8500, with P= #8400.

Needless to say, a program combining ALL the techniques listed here will be a truly formidable program to pirate.

APPENDIX 1
SPECIFICATIONS FOR THE DISATOM SUPER ROM

The DISATOM is contained in a 4K ROM that is fitted in the utility socket (address A000). It contains two major areas: Machine Level with Memory Handling, and Additons to BASIC. It is permanently resident, does not require a LINK command, and does not use any addresses (such as zero page) you are likely to use. Most words may be abbreviated, and used in BASIC programs.

I. Additions to the BASIC Language

AULD XX : where XX is a hex number. This allows recovery of text from any text space you wish (Celtic OLD !).
It executes ? #12= XX then OLD (See command PAGE XX).

AUTO X,Y (or A. X,Y) : produces automatic line numbering for writing programs, beginning at X in steps of Y. Default is 100,10. RETURN or ESC exits.

COPY X,Y,Z : copies everything from X to Y inclusive to the new location starting at Z. It takes account of direction so the copy won't overwrite the source. COPY uses the same syntax as PLOT, so X,Y,Z may be numbers, variables, or arbitrarily complex functions enclosed in brackets. AVOID addresses that encompass 0000 or FFFF!

CURSOR X,Y : places the cursor where you wish. X is horizontal, Y is vertical, and defaults are the current position, but either X or Y MUST be given. Thus CURSOR X will operate as a screen TAB(X). 0,0 is top left of screen. Does not operate after a NAK.

DELETE X,Y : deletes all BASIC lines from X to Y inclusive. If X and Y are not specified DELETE will not operate.

DUMP : prints out all simple BASIC variables which have currently been used, and their values.

DIR : directory, to list all the functions of DISATOM.

ERUN : runs a program with error check. If one is found the line is displayed with the cursor over the probable error.

EXEC$X : where X is a string variable, results in the string being executed as a function. So for example
```
10 $A="Y=3*2+20/10"
20 EXEC$A
```
results in Y being set equal to 8. Any arbitrarily complex function or command is allowed in the string.

FIND .A.T.O.M : returns hex address of all locations containing the ASCII code for ATOM.

FIND[LDA@ 0;STA #80] : returns hex address of all locations containing machine code A9 00 85 80.

FIND"PRINT X" : displays all BASIC lines containing the words PRINT X.

FIND 20 30 7F : returns hex address of all locations containing machine code sequence 20 30 7F.

HEADER X : where X=0 thru 6, causes X lines at the top of the screen to NOT scroll, so anything there can be used as a header. LOW or HEADER 0 cancels.

HELP : makes anything coming in from tape visible via the cursor. If the tape is faulty and a SUM ERROR occurs,an automatic *FLOAD is executed, so you can rewind a bit and continue loading any number of times. Syntax is:
HELP"filename". NOTE-cannot be used to relocate!

HIGH : causes **all** cassette tape read or write operations to be performed at 1200 BAUD, and made visible in the cursor. The cursor symbol is forbidden in tape filenames. LOW returns rates to normal.

INKEY X,T : where X is a variable, captures the key pushed in the variable. T is the time allowed to push the key, in units of 50 msec. (default 0, max 128). If no key was pushed in the time allowed the variable will contain an FF (255).

[↑] : as for HIGH, but for this **ONE TIME ONLY**. E.g.[↑] *LOAD"TEST" or [↑] LOAD"MYTAPE" or [↑] *SAVE 2900 3C00.

LOW : causes **all** casssette tape read or write operations to be performed at 300 BAUD (normal ATOM speed).This also returns all vectors in page 2 to normal values.

NUKE : the really thorough **NEW**. It punches FF into all ram memory up to A000, then BREAKS.

ON ERROR <any valid command or function> : will accomplish the command or function (this is usually a GOTO) when an error occurs instead of BREAKING.

OUT X,Y : causes output from the tape socket in RS232 format, with handshake. X=BAUD rate , Y=Number of line feeds (default=1) per emitted line feed. Values of X are:
1=2400 BAUD
2=1200
4=600
8=300 etc. Default=1200 BAUD
Pin Connections are:
6=serial ouput
2 =earth
4=handshake,which MUST have a 1K resistor to the printer's 5V handshake. If there is no handshake then connect this pin to 5V via a 1K resistor.

PAGE XX : where XX are two hex digits. This has the effect of
? #12=XX
NEW
This enables you to establish a new text space without fuss.

PULL N or U or R : ATOM allows only a certain number of nests for FOR..NEXT, DO..UNTIL, and GOSUB..RETURN loops. PULL allows you to leave loops at any time by pulling the NEXT or UNTIL or RETURN from the memory.

READ-DATA-RESTORE : This combination is used as in standard BASIC However, this version is much more powerful. RESTORE can be used to 1)restore to the beginning of data 2)restore to a line number 3)restore to a label 4)restore to the line number arrived at by soluton of an equation 5)restore to the next highest line number if the solution does not point at a line number. The DATA list can contain strings (in quotes),decimal and/or hex numerics, variables, or arbitrarily complex functions. The READ statement will accept ANYTHING that can be placed on the left of an equals sign! (e.g. READ $A+LEN A). You can READ into bytes,words,arrays,variables, etc. . E.g.:

```
5 C=15;DIM XX(1),Y(15),S(4)
10 [X] DATA "help",10,32,C+7
15 RESTORE 10 (or RESTORE C*2/3 or RESTORE [X] or RESTORE)
20 READ $S;READ XX(1);READ Y(C);READ Z;END
```

Results in $S="help",XX(1)=10,Y(15)=32,Z=22 .
ALWAYS RESTORE before attempting the first READ in the program (to set the data pointer).

REN X,Y : Renumbers all BASIC lines to start at X and increments at Y (Default is 100,10), and then lists results.

TAPE XXXX : where XXXX is a hex address. This captures anything on tape, including the header, and places it direct into memory starting at XXXX. Especially useful to recover badly damaged tapes.

TONE X,$Y : to create music and sounds. X is the duration in 50 msec units (NO defaults, max=127), and $Y the note. There are 6 octaves numbered 0-5, + means sharp, and - means flat. "R" means rest.The minumum note is "0C" and the max is "5D". For example TONE 5,"2C+" will give 250msec of the third ocatave C sharp. Both durations and strings can be read from data statements.All tones are automatically outputed through the tape socket for you to record.

ZERO : sets all simple BASIC variables to zero.

II. Machine Level Functions

[D]xxxx : disassembles starting at location hex xxxx,and waits for the REPEAT key. Otherwise [D]xxxx,yyyy doesn't wait. This will appear on the screen as:
ADDRESS OBJECT CODE SOURCE CODE ASCII Equivilent
The # is not needed, and all xx's need not be used. For example, [D]80 disassembles at hex 80. REPEAT key continues, and ESC gets out of the mode.To Edit, see instructions below.

[H]xxxx : Hex dump of memory starting at hex xxxx. This may be used to edit the memory as given below. Pushing REPEAT will continue the dump, and ESC exits the mode. [H]xxxx,yyyy will dump without waiting for the REPEAT key.

[A]xxxx : ASCII dump of memory starting at hex xxxx. The contents of memory are displayed on the screen as their ASCII equivilents. These may also be edited as given below.If no ASCII equivilent the hex is shown. [A]xxxx,yyyy will dump without waiting.

EDITING MEMORY USING THE ABOVE FUNCTIONS:
All the above modes will display memory contents as either a two-digit hex number (one byte), or its ASCII equivilent, in which case it will appear with a full stop in front (e.g. 41 will appear as .A in an ASCII Dump). To change the memory contents, hit ESC, and the prompt > will return. Move the cursor over the line you want to edit, then COPY to the point on the line where you want to make the change. You may then type in EITHER the ASCII equivilent with a dot in front OR the two digit hex number, and this may be done as many times as you wish along the line. At the end of the line hit RETURN and ESC. DO NOT edit more than one line at a time without hitting RETURN and ESC. You need not go to the end of the line before hitting RETURN-the rest of the line will copy automatically.This method of editing is used in all three of the above modes.

[T]xxxx A X Y Sp S : Machine code TRACE Function, where xxxx is the hex address of a machine code program. A,X,Y,Sp,S can be set before entry.A=Accumulator;X, Y=X and Y stack Sp=stack pointer(always FF), S=status register. Default is all zeros except Sp=FF. Type in the command and hit <CR>, then <SHIFT> executes the next instruction ,but JSR without displaying the subroutine, while <REPT> shows the actions in the subroutine (! these may be tortuous !).The top of the screen displays the contents of all the registers and all the flags, plus the ASCII equivilent of Accumulator contents.

[X]: runs the machine code routine pointed to by location hex 180. On its own this has the effect of LINK (?180,181) or JMP (180). Your m/c code routine MUST end in JMP #C55B. However, the real strength is that it is possible to put various parameters after the [X] , and then capture them using the 5,Y pointer. This function then becomes an invaluable development tool for machine code routines.

APPENDIX 2
HEX DUMP AND MODIFY

Below is the source code to enable a HEX DUMP of memory contents, and modification if this is required. This is one of the features found in a DISATOM ROM . Remember that the m/c code must be resident for it to work, so don't overwrite it once it has been assembled. LINK to the first code to activate (here #2800).

```
40 V= #70;K= #72;T= #75
50 DIM JJ5;F.I=0TO5;JJ(I)=-1;N.
60 PRINT $21
70 FOR X=0 TO 1
80 P= #2800
90[
100 LDA @ JJ0/256
110 STA #207
120 LDA @ JJ0%256
130 STA #206
140 RTS
150:JJ0
160 LDY @ 0
170 STY T
180 JSR #F876
190 CMP @ CH"*"
200 BEQ JJ1
210 JMP #F8EF
220:JJ1
230 LDA @ 11
240 JSR #FFF4
250 LDX @ V
260 INY
270 JSR #F893
280 LDX @ K
290:JJ2
300 JSR #F876
310 CMP @#0D
320 BEQ JJ3
330 JSR #F893
340 TYA
350 PHA
360 LDA K
370 LDY T
380 STA(V),Y
390 INC T
400 PLA
410 TAY
420 BNEJJ2
430:JJ3
440 LDX @ V
450 JSR #F7D1
460]
470 $P=" **";P=P+LEN(P)
480[
490 NOP
500 JSR #F7F1
510:JJ4
520 LDA(V),Y
530 JSR #F7FA
540 INY
550 CPY @ 8
560 BNE JJ4
570 TYA
580 CLC
590 ADC V
600 STA V
610 BCC JJ5
620 INC V+1
630:JJ5
640 BIT #B002
650 BVC JJ3
660 JSR #C504
670 BNE JJ5
680]
690 NEXT X;PRINT$6;END
```

TO OPERATE: type **XXXX. This gives a hex dump of memory starting at hex xxxx. This may be used to edit the memory as given below. Pushing <REPEAT> will continue the dump, and <ESC> exits the mode.

EDITING MEMORY: This program displays memory contents as a two-digit hex number (one byte). To change the memory contents, hit ESC, and the prompt > will return. Move the cursor over the line you want to edit, then COPY to the point on the line where you want to make the change. You may then type in the two digit hex number, and this may be done as many times as you wish along the line. At the end of the line hit <RETURN> and <ESC>. DO NOT edit more than one line at a time without hitting <RETURN> and <ESC>. You need not go to the end of the line before hitting RETURN- the rest of the line will copy automatically.

6502 OP CODES (for disassembly)

ØØ - BRK	2Ø - JSR	4Ø - RTI	6Ø - RTS	8Ø - Future Expansion	AØ - LDY - Immediate	CØ - CPY - Immediate	EØ - CPX - Immediate
Ø1 - ORA - (Indirect,X)	21 - AND - (Indirect,X)	41 - EOR - (Indirect,X)	61 - ADC - (Indirect,X)	81 - STA - (Indirect,X)	A1 - LDA - (Indirect,X)	C1 - CMP - (Indirect,X)	E1 - SBC - (Indirect,X)
Ø2 - Future Expansion	22 - Future Expansion	42 - Future Expansion	62 - Future Expansion	82 - Future Expansion	A2 - LDX - Immediate	C2 - Future Expansion	E2 - Future Expansion
Ø3 - Future Expansion	23 - Future Expansion	43 - Future Expansion	63 - Future Expansion	83 - Future Expansion	A3 - Future Expansion	C3 - Future Expansion	E3 - Future Expansion
Ø4 - Future Expansion	24 - BIT - Zero Page	44 - Future Expansion	64 - Future Expansion	84 - STY - Zero Page	A4 - LDY - Zero Page	C4 - CPY - Zero Page	E4 - CPX - Zero Page
Ø5 - ORA - Zero Page	25 - AND - Zero Page	45 - EOR - Zero Page	65 - ADC - Zero Page	85 - STA - Zero Page	A5 - LDA - Zero Page	C5 - CMP - Zero Page	E5 - SBC - Zero Page
Ø6 - ASL - Zero Page	26 - ROL - Zero Page	46 - LSR - Zero Page	66 - ROR - Zero Page	86 - STX - Zero Page	A6 - LDX - Zero Page	C6 - DEC - Zero Page	E6 - INC - Zero Page
Ø7 - Future Expansion	27 - Future Expansion	47 - Future Expansion	67 - Future Expansion	87 - Future Expansion	A7 - Future Expansion	C7 - Future Expansion	E7 - Future Expansion
Ø8 - PHP	28 - PLP	48 - PHA	68 - PLA	88 - DEY	A8 - TAY	C8 - INY	E8 - INX
Ø9 - ORA - Immediate	29 - AND - Immediate	49 - EOR - Immediate	69 - ADC - Immediate	89 - Future Expansion	A9 - LDA - Immediate	C9 - CMP - Immediate	E9 - SBC - Immediate
ØA - ASL - Accumulator	2A - ROL - Accumulator	4A - LSR - Accumulator	6A - ROR - Accumulator	8A - TXA	AA - TAX	CA - DEX	EA - NOP
ØB - Future Expansion	2B - Future Expansion	4B - Future Expansion	6B - Future Expansion	8B - Future Expansion	AB - Future Expansion	CB - Future Expansion	EB - Future Expansion
ØC - Future Expansion	2C - BIT - Absolute	4C - JMP - Absolute	6C - JMP - Indirect	8C - STY - Absolute	AC - LDY - Absolute	CC - CPY - Absolute	EC - CPX - Absolute
ØD - ORA - Absolute	2D - AND - Absolute	4D - EOR - Absolute	6D - ADC - Absolute	8D - STA - Absolute	AD - LDA - Absolute	CD - CMP - Absolute	ED - SBC - Absolute
ØE - ASL - Absolute	2E - ROL - Absolute	4E - LSR - Absolute	6E - ROR - Absolute	8E - STX - Absolute	AE - LDX - Absolute	CE - DEC - Absolute	EE - INC - Absolute
ØF - Future Expansion	2F - Future Expansion	4F - Future Expansion	6F - Future Expansion	8F - Future Expansion	AF - Future Expansion	CF - Future Expansion	EF - Future Expansion
1Ø - BPL	3Ø - BMI	5Ø - BVC	7Ø - BVS	9Ø - BCC	BØ - BCS	DØ - BNE	FØ - BEQ
11 - ORA - (Indirect),Y	31 - AND - (Indirect),Y	51 - EOR - (Indirect),Y	71 - ADC - (Indirect),Y	91 - STA - (Indirect),Y	B1 - LDA - (Indirect),Y	D1 - CMP - (Indirect),Y	F1 - SBC - (Indirect),Y
12 - Future Expansion	32 - Future Expansion	52 - Future Expansion	72 - Future Expansion	92 - Future Expansion	B2 - Future Expansion	D2 - Future Expansion	F2 - Future Expansion
13 - Future Expansion	33 - Future Expansion	53 - Future Expansion	73 - Future Expansion	93 - Future Expansion	B3 - Future Expansion	D3 - Future Expansion	F3 - Future Expansion
14 - Future Expansion	34 - Future Expansion	54 - Future Expansion	74 - Future Expansion	94 - STY - Zero Page,X	B4 - LDY - Zero Page,X	D4 - Future Expansion	F4 - Future Expansion
15 - ORA - Zero Page,X	35 - AND - Zero Page,X	55 - EOR - Zero Page,X	75 - ADC - Zero Page,X	95 - STA - Zero Page,X	B5 - LDA - Zero Page,X	D5 - CMP - Zero Page,X	F5 - SBC - Zero Page,X
16 - ASL - Zero Page,X	36 - ROL - Zero Page,X	56 - LSR - Zero Page,X	76 - ROR - Zero Page,X	96 - STX - Zero Page,Y	B6 - LDX - Zero Page,Y	D6 - DEC - Zero Page,X	F6 - INC - Zero Page,X
17 - Future Expansion	37 - Future Expansion	57 - Future Expansion	77 - Future Expansion	97 - Future Expansion	B7 - Future Expansion	D7 - Future Expansion	F7 - Future Expansion
18 - CLC	38 - SEC	58 - CLI	78 - SEI	98 - TYA	B8 - CLV	D8 - CLD	F8 - SED
19 - ORA - Absolute,Y	39 - AND - Absolute,Y	59 - EOR - Absolute,Y	79 - ADC - Absolute,Y	99 - STA - Absolute,Y	B9 - LDA - Absolute,Y	D9 - CMP - Absolute,Y	F9 - SBC - Absolute,Y
1A - Future Expansion	3A - Future Expansion	5A - Future Expansion	7A - Future Expansion	9A - TXS	BA - TSX	DA - Future Expansion	FA - Future Expansion
1B - Future Expansion	3B - Future Expansion	5B - Future Expansion	7B - Future Expansion	9B - Future Expansion	BB - Future Expansion	DB - Future Expansion	FB - Future Expansion
1C - Future Expansion	3C - Future Expansion	5C - Future Expansion	7C - Future Expansion	9C - Future Expansion	BC - LDY - Absolute,X	DC - Future Expansion	FC - Future Expansion
1D - ORA - Absolute,X	3D - AND - Absolute,X	5D - EOR - Absolute,X	7D - ADC - Absolute,X	9D - STA - Absolute,X	BD - LDA - Absolute,X	DD - CMP - Absolute,X	FD - SBC - Absolute,X
1E - ASL - Absolute,X	3E - ROL - Absolute,X	5E - LSR - Absolute,X	7E - ROR - Absolute,X	9E - Future Expansion	BE - LDX - Absolute,Y	DE - DEC - Absolute,X	FE - INC - Absolute,X
1F - Future Expansion	3F - Future Expansion	5F - Future Expansion	7F - Future Expansion	9F - Future Expansion	BF - Future Expansion	DF - Future Expansion	FF - Future Expansion

```
DC - ADD WITH CARRY
     ZERO=65                                  Z,X=75
     ABS =6D              ABS,X=7D            ABS,Y=79
     IMM =69              PRE,X=61            POST,Y=71
ND - LOGICAL AND
     ZERO=25                                  Z,X=35
     ABS =2D              ABS,X=3D            ABS,Y=39
     IMM =29              PRE,X=21            POST,Y=31
SL - ARITHMETIC SHIFT LEFT
     ZERO=06              ABS,X=1E            ACCUM =0A
     ABS =0E              ZERO,X=16
CC - BRANCH ON CARRY CLEAR =90
CS - BRANCH ON CARRY SET =B0
EQ - BRANCH ON RESULT ZERO =F0
VC - BRANCH ON OVERFLOW CLEAR =50
IT - BITS 6+7 TO STAT REG(IF A+M=0 THEN Z=0,OTHERWISE Z=1)
     ABS=2C               ZERO=24
MI - BRANCH ON RESULT MINUS=30
NE - BRANCH ON RESULT NOT EQUAL ZERO =D0
PL - BRANCH ON RESULT PLUS =10
RK - BREAK =00
VC - BRANCH ON OVERFLOW CLEAR =50
VS - BRANCH ON OVERFLOW SET =70
LC - CLEAR CARRY FLAG =18
LD - CLEAR DECIMAL MODE =D8
LI - CLEAR INTERUPT DISABLE BIT=58
LV - CLEAR OVERFLOW FLAG =B8
MP - COMPARE WITH ACCUMULATOR
     ZERO=C5                                  Z.X =D5
     ABS =CD              ABS,X=DD            ABS,Y=D9
     IMM =C9              PRE,X=C1            POST,Y=D1
PX - COMPARE WITH INDEX
     ZERO=E4              ABS =EC             IMM=E0
PY - COMPARE WITH INDEX Y
     ZERO=C4              ABS =CC             IMM=C0
EC - DECREMENT BY ONE
     ZERO=C6              Z,X=D6              ABS=CE
EX - DECREMENT X REGISTER BY ONE =CA
EY - DECREMENT Y REGISTER BY ONE =88
OR - EXCLUSIVE OR WITH ACCUM
     ZERO=45                                  Z,X =55
     ABS =4D              ABS,X=5D            ABS,Y=59
     IMM =49              PRE,X=41            POST,Y=NONE
NC - INCREMENT BY ONE
     ZERO=E6              Z,X=F6
     ABS =EE              ABS,X=FE
NX - INCREMENT X REGISTER BY ONE =E8
NY - INCREMENT Y REGISTER BY ONE =C8
MP - JUMP TO NEW LOCATION
     ABS =4C              INDIRECT=6C
SR - JUMP TO SUBROUTINE =20
```

```
LDA - LOAD ACCUMULATOR
     ZERO=A5                                  Z,X  =B5
     ABS =AD              ABS,X=BD            ABS,Y=B9
     IMM =A9              PRE,X=A1            POST,Y=B1
LDX - LOAD REGISTER X
     ZERO=A6              IMM  =A2            ABS,Y=BE
     ABS =AE                Z,Y=B6
LDY - LOAD REGISTER Y
     ZERO=A4                IMM=A0            ABS,X=BC
     ABS =AC              Z,X=B4
LSR - LOGICAL SHIFT RIGHT
     ZERO=46              ABS,X=5E            ACCUM=5A
     ABS =4E                Z,X=56
NOP - NO OPERATION =EA
ORA - LOGICAL OR WITH ACCUMULATOR
     ZERO=05                                  Z,X =15
     ABS =0D              ABS,X=1D            ABS,Y=19
     IMM =09              PRE,X=01            POST,Y=11
PHA - PUSH ACCUM TO STACK =48
PHP - PUSH PROCESSOR STATUS ON STACK =08
PLA - PULL ACCUM FROM STACK =68
PLP - PULL PROCESSOR STATUS FROM STACK =28
ROL - ROTATE LEFT ONE BIT
     ZERO=26              ABS,X=3E            ACCUM=2A
     ABS =2E                Z,X=36
ROR - ROTATE RIGHT ONE BIT
     ZERO=66              ABS,X=7E            ACCUM=6A
     ABS =6E                Z,X=76
RTI - RETURN FROM INTERUPT =40
RTS - RETURN FROM SUBROUTINE =60
SBC - SUBTRACT WITH CARRY
     ZERO=E5                                  Z,X  =F5
     ABS =ED              ABS,X=FD            ABS,Y=F9
     IMM =E9              PRE,X=E1            POST,Y=F1
SEC - SET CARRY FLAG =38
SED - SET DECIMAL MODE FLAG =F8
SEI - SET INTERUPT DIASABLE BIT =78
STA - STORE ACCUMULATOR
     ZERO=85
     ABS =8D              ABS,X=9D            ABS,Y=99
     Z,X =95              PRE,X=81            POST,Y=91
STX - STORE X REGISTER
     ZERO=86              ABS=8E                Z,Y=96
STY - STORE Y REGISTER
     ZERO=84              ABS =8C               Z,X =94
TAX - TRANSFER ACCUM TO REGISTER X =AA
TAY - TRANSFER ACCUM TO REGISTER Y =A8
TSX - TRANSFER STACK POINTER TO REGISTER X =BA
TXA - TRANSFER REGISTER X TO ACCUM =8A
TXS - TRANSFER REGISTER X TO STACK POINTER =9A
TYA - TRANSFER REGISTER Y TO ACCUM =98
```

IMM =IMMEDIATE ENTRY ,X or ,Y=INDEXED BY X or Y

PRE=INDIRECT PRE-INDEXED POST=INDIRECT POST-INDEXED

ABS=ABSOLUTE ADDRESS(4 BYTE) Z or ZERO=ZERO PAGE ADDRESS (2 BYTE)

ONE PAGE =256 BYTES (00 TO FF)

APPENDIX 4
ASCII AND CONTROL CODES

All 128 ASCII codes are available from the ATOM, but only 127 direct from the keyboard. the "back arrow" (#5F) can only be gotten from program.

DECIMAL	HEX	<CTRL> + KEY	CALL	ATOM ACTION
2	2	B	STX	START PRINTER
3	3	C	ETX	END PRINTER
6	6	F	ACK	START VDU
7	7	G	BELL	BEEP SPEAKER
8	8	H	BS	BACKSPACE-NO ERASE
9	9	I	HT	FORWARD SPACE
10	A	J	LF	LINE FEED
11	B	K	VT	UP ONE LINE
12	C	L	FF	CLEAR VDU+HOME UP
13	D	M	CR	CARRIAGE RETURN
14	E	N	SO	PAGE MODE ON
15	F	O	SI	PAGE MODE OFF
21	15	U	NAK	SCREEN OFF
24	18	X	CAN	ERASE CURRENT LINE
27	1B	[	ESC	ESCAPE FROM BASIC
		Hit twice,	set	VDU to character mode
30	1E		RS	HOME UP LEFT.

DECIMAL	HEX	CHARACTER
32	20	<SPACE>
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:

DECIMAL	HEX	CHARACTER
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	↑
95	5F	←

DECIMAL	HEX	<SHIFT> + KEY	ASCII CHAR	DISPLAYED AS
96	60	@	\	[@]
97	61	A	a	[A]
98	62	B	b	[B]
99	63	C	c	[C]
100	64	D	d	[D]
101	65	E	e	[E]
102	66	F	f	[F]
103	67	G	g	[G]
104	68	H	h	[H]
105	69	I	i	[I]
106	6A	J	j	[J]
107	6B	K	k	[K]
108	6C	L	l	[L]

DECIMAL	HEX	<SHIFT> + KEY	ASCII CHAR	DISPLAYED AS
109	6D	M	m	[M]
110	6E	N	n	[N]
111	6F	O	o	[O]
112	70	P	p	[P]
113	71	Q	q	[Q]
114	72	R	r	[R]
115	73	S	s	[S]
116	74	T	t	[T]
117	75	U	u	[U]
118	76	V	v	[V]
119	77	W	w	[W]
120	78	X	x	[X]
121	79	Y	y	[Y]
122	7A	Z	z	[Z]
123	7B	[	{	[
124	7C	\	|	[\]
125	7D	]	}	[ ]
126	7E	↑	~	[↑]
127	7F	BACKSPACE WITH DELETE		

I N D E X T O R O U T I N E S

(*) Represents a usable routine, (!*!) Recommended routine.

```
INPUT BUFFER-SEE 'STRING INPUT BUFFER'
INPUT CD09(!*!),CC81
INTEGER VARIABLES CA2F(*),C8D7(*),CA37(*)
IRQ FFB2
KEYPRESS SEE 'GET'
LABEL CC1F,C54A(*), SEE 'RAM'38D - 3C0
LEN C9BD(*)
LET C31B
LINE ENTRY CDC9
LINE NUMBER CC1F(*),C54A(*)
LINE NUMBER SEARCH C62E(*)
LINK C3B2
LOAD CEED(*)
LOAD FILE F96E,FFE0(*)
MINUS C8C1(*)
MOVE-SEE 'PLOT'
MULTIPLICATION C813,C661,C689
NAME F86C
NEGATION C8C1(*)
NEW C2AD(*)
NEXT CACD
NMI FFC7
NUMERIC ASSIGNMENTS SEE'ASSIGNMENTS'
OLD F531
OPERATING SYSTEM VECTORS FFCB AND ONWARD
OR C7D3
PLING C3EE,C9F5
PLOT F542 AND ONWARD
POINT PLOTS F6E2(*)
PRINT ACCUM. CA4C
PRINT CHAR FE52
PRINT COMMAND C334
PRINT F3FE
PRINT ROUTINES C33F,W/S STACK=C589(*),ACC AS ASCII CA4C(*),ACC AS
        HEX =F376(*),F37E,IN-LINE ASCII F7D1(!*!),NUMBERS
        F7EC(!*!), CHARACTERS FE52(*),W/S STACK AS HEX
        C349(*),SEE 'RAM' F
PRINTER SEE CHAPTER 7
PUT CF95(*)
QUESTION MARK C406,C94C
QUOTES CEB1,CEBF(*)
RAM CHECK F119
RANDOM NUMBER C986(!*!), SEE 'EXAMPLES',SEE 'RAM' 8 TO C
READ NUMERIC C465(*),F893
REM C575
RESET FF3F(*)
RETURN CBEC,C4E4(*),C55B(!*!)
ROM CHECK CA24(*),C54A,CA24
RUBBISH CHECKS C4E4,FA65(*),FA76(*)
RUN F141(*),CE83(*)
SAVE CF0A(*),FA86,FABB,FAE5,SEE 'O/S VECTORS'
SEMI-COLON C4E4(*)
SGET CFE3
SPUT CFC5
STEP CBA2
STRING COPY CEBF(*),F818(*)
STRING INPUT BUFFER CEBF(*),CEFA(*),F818(*),F875(*),F893(*)
```

```
SUBTRACTION C7B7
SYNCHRONISE AT 2.4 KHZ FCD8(*)
TAPE FBEE(*),FC7C(*)
TAPE FILES SEE CHAPTER 7
TAPE TITLE CEFA,SEE CHAPTER 7
TEXT AREA SEE 'RAM' 12,CE83(*),F141(*), SEE APPENDX 1'AULD''PAGE'
TEXT POINTER AND OFFSET SEE 'RAM' 5,6 AND 0
TIMING-SEE 'WAIT'
TITLE CEFA(*)
TO CB81
TOP C973(*),CD98(*),SEE 'RAM' D,E
TRUTH TEST C70C(*),C714,C722,C731
UNTIL CCD2, SEE 'DO'
VARIABLES SEE 'INTEGER VARIABLES'
VECTOR COMPARE FA08(*)
VECTOR DECREMENT F668(*),INCREMENT F671(*),FA08(*)
VECTORS-OPERATING FFCB AND ONWARDS
WAIT F14C,FB3B(!*!),FE66(*),FCD8(*)
WORKSPACE STACK CA2F(*),C589(*),CA37(*),SEE CHAPS 3+6,SEE 'RAM' 4
        AND 16 TO 51
```