DISATOM
SUPER ROM

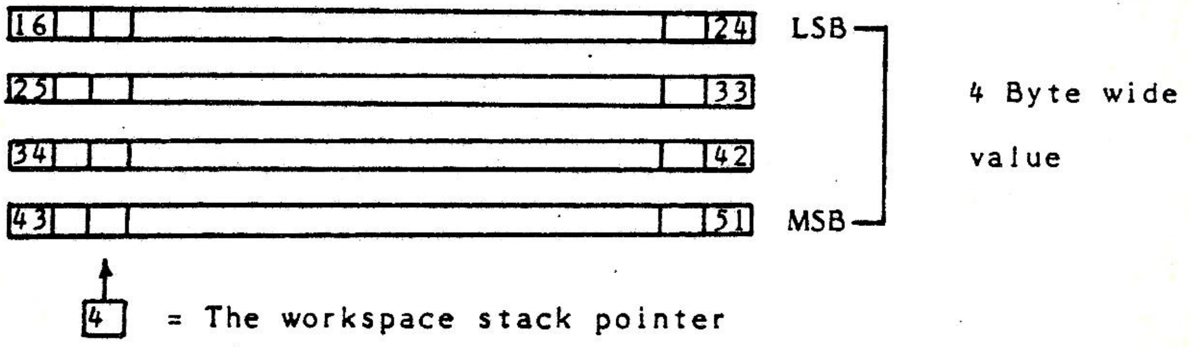
User manual

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.

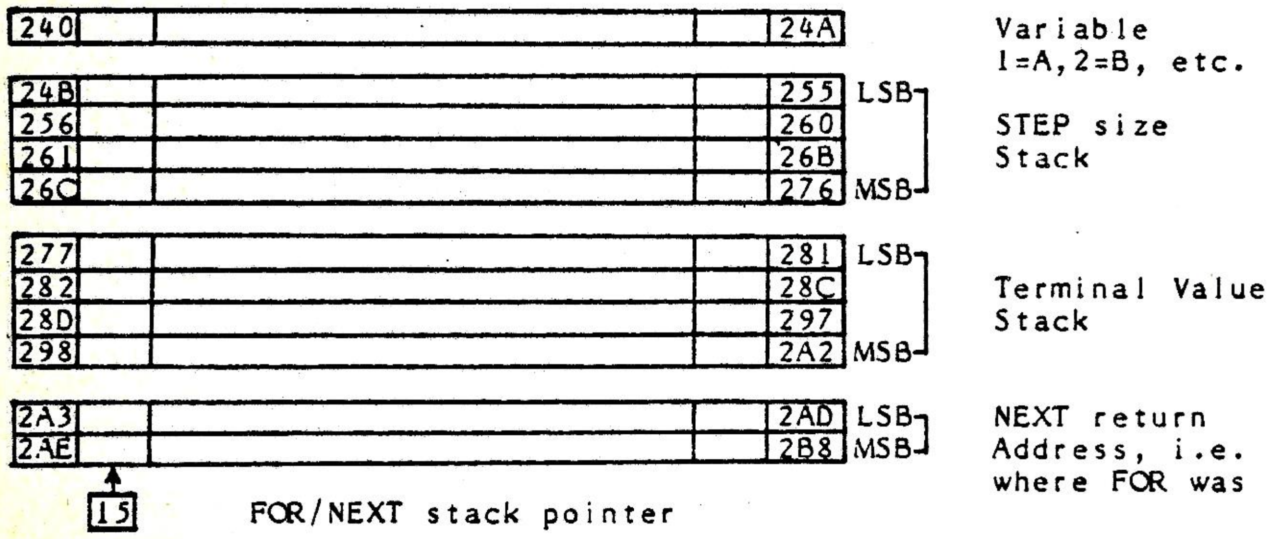
Many references are made in this book to routines which read or write values to the workspace stack, and may be used fairly freely by those writing machine code routines. One example is given below. It is extracted from the ATOM ROM at C99D, and is part of a routine to copy a random number in location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

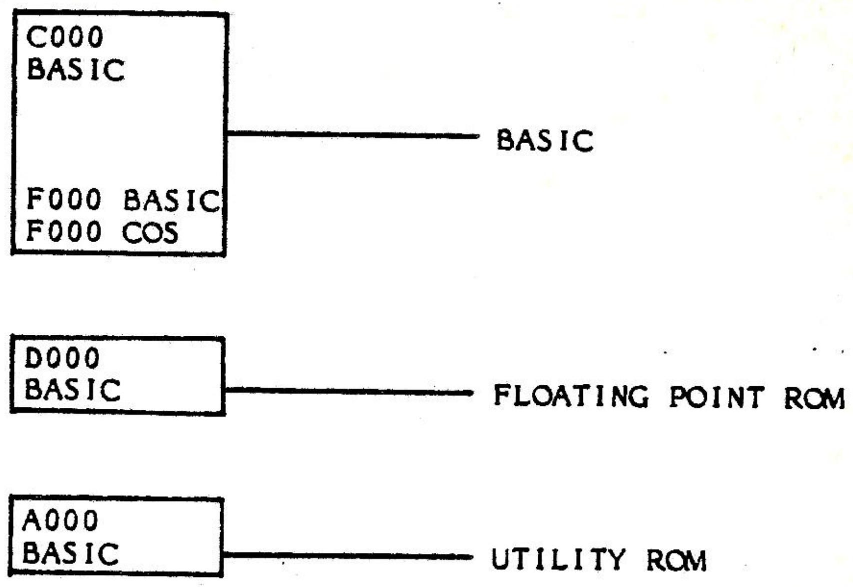
II. The FOR/NEXT Stacks



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 (negative=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= $\boxed{@}$;38F,390= $\boxed{A}$ 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
$\boxed{A}$	38D	38E
$\boxed{B}$	38F	390
$\boxed{C}$	391	392
$\boxed{D}$	393	394
$\boxed{E}$	395	396
$\boxed{F}$	397	398
$\boxed{G}$	399	39A
$\boxed{H}$	39B	39C
$\boxed{I}$	39D	39E
$\boxed{J}$	39F	3A0
$\boxed{K}$	3A1	3A2
$\boxed{L}$	3A3	3A4
$\boxed{M}$	3A5	3A6
$\boxed{N}$	3A7	3A8
$\boxed{O}$	3A9	3AA
$\boxed{P}$	3AB	3AC
$\boxed{Q}$	3AD	3AE
$\boxed{R}$	3AF	3B0
$\boxed{S}$	3B1	3B2
$\boxed{T}$	3B3	3B4
$\boxed{U}$	3B5	3B6
$\boxed{V}$	3B7	3B8
$\boxed{W}$	3B9	3BA
$\boxed{X}$	3BB	3BC
$\boxed{Y}$	3BD	3BE
$\boxed{Z}$	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 CD0F 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 workspacestack 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 CD0F 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: O,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

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 #FD71;]          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 $\boxed{X}$ 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 $\boxed{X}$ 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 $\boxed{X}$ 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 $\boxed{X}$ 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 $\boxed{X}$ 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.

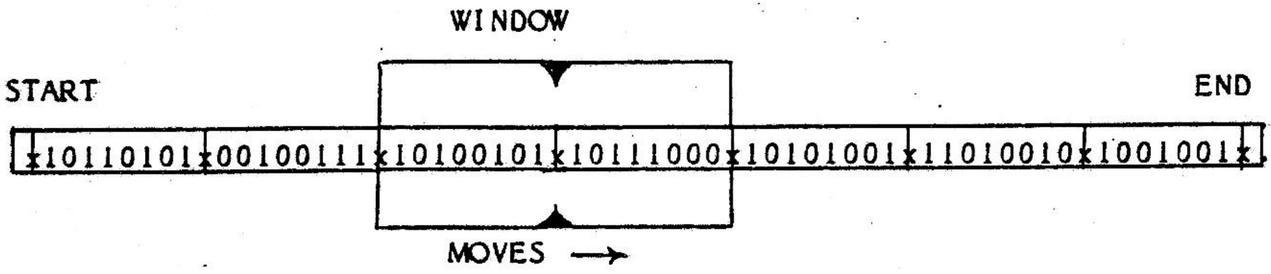
But what if they are not identical? This is harder to work out. Here are the possible reasons:

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).

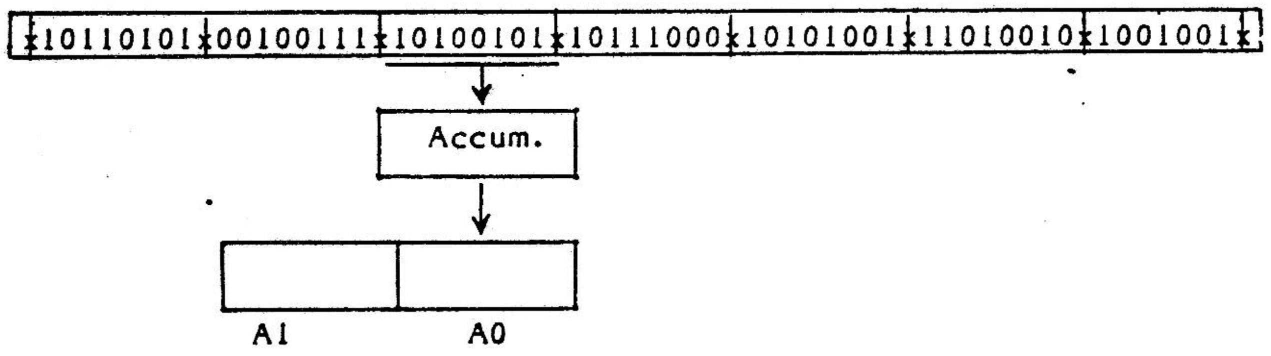
You must now go through various diagnostic procedures to find out just what the problem is. This is the rub. CRC is excellent at telling you that things are not right, but tells you nothing about where the error is. You can of course be lucky and have an error where it doesn't make any difference anyway (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:


WINDOW
START
END
x10110101x00100111x10100101x10111000x10101001x11010010x1001001x
MOVES →

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.


x10110101x00100111x10100101x10111000x10101001x11010010x1001001x
Accum.
A1
A0

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 $\boxed{X}$ 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 $\boxed{X}$ after running the source code. When reloading the m/c code, type
    ! #180= #3CA
and this will point DISATOM's $\boxed{X}$ 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.

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).

$\boxed{\uparrow}$: as for HIGH, but for this **ONE TIME ONLY.** E.g.$\boxed{\uparrow}$*LOAD"TEST" or $\boxed{\uparrow}$LOAD"MYTAPE" or $\boxed{\uparrow}$*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 finish 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

$\boxed{D}$xxxx : disassembles starting at location hex xxxx,and waits for the REPEAT key. Otherwise $\boxed{D}$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, $\boxed{D}$80 disassembles at hex 80. REPEAT key continues, and ESC gets out of the mode.To Edit, see instructions below.

$\boxed{H}$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. $\boxed{H}$xxxx,yyyy will dump without waiting for the REPEAT key.

$\boxed{A}$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. $\boxed{A}$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.

$\boxed{T}$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.

$\boxed{X}$: 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 $\boxed{X}$, 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.

INDEX TO ROUTINES

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

ABS C902 (*)
ADDITION C79D
ALPHANUMERIC CONVERSION C434 (*)
AND C87B
ARRAY PRE-TEST F02E,F04B
ARRAY ADDRESSES F08B
ASCII CHARACTERS F87E
ASSEMBLER FI55, F2A1,F38E
ASSIGNMENTS,NUMERIC C8F8,C8DC(*),CA2F(*)
ASSIGNMENTS C3E5,C8DC(*)
BGET CF5B(*)
BPUT CF8F
BRACKETS C944
BREAK C2B2(*)
BREAK KEY FF3F(*)
BRK C9D8,FFC0
CARRIAGE RETURN C4E4(*)
CH (ASCII) C9D2
CLEAR F67B
COMMAND MEANINGS C279
COMPARE VECTOR FA08(*)
CONTROL CODES FCEA(*)
COS COMMANDS,EXECUTION C40F
COS INTERPRETER F8F0(*)
COS MESSAGES FC38(*)
COS WORDS F8BE
COUNT C97A(*),CA4C(*),SEE 'RAM' 7
DATA C000,C608,F000,F155,F7C9,F8BE,FECB,FF9A et. al.
DECIMAL STRING C465(*)
DECREMENT VECTOR F668(*)
DIM F0AE,F141(*), SEE 'RAM' 23,24
DO CCF0,SEE RAM 13
DOLLAR CEB1(*)
DRAW SEE 'PLOT'
END CD98(*)
EOR C7EF
ERROR HANDLING C9E7(*),SEE 'RAM' 0 + 10,11
ERROR-COS F926(*)
ESC KEY C504(*)
EVALUATE A FUNCTION C3C8(*),C8BC(*)
FETCH KEYPRESS - SEE 'GET'
FETCH NEXT CHAR F291(*),F875(*)
FIELD FLYBACK FE66(*)
FIN CFA6(*)
FOR CB57, SEE 'RAM' 15
FOUT CFA7(*)
FUNCTION INTERPRETER C22C,C3C8(*),C8BC(*)
GET CF66(*),FE94(*),FE71(*)
GOSUB CBD2,SEE 'RAM' 14
GOTO CC05
GRAPHICS F6CF
HEX SIGN (#)C90A
IF C566
INCREMENT VECTOR F671(*),FA08(*)
INTERPRET A STATEMENT (!*!)C55B

```
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(*)
```