

PLEASE READ AND UNDERSTAND THESE INSTRUCTIONS BEFORE PROCEEDING.

Your programmer's toolkit is supplied in a 4K EPROM which must be inserted in socket 24 of your Acorn Atom. To fit the chip, turn the Atom upside down and remove the two large crosspoint screws which attach the base to the keyboard Lift off the base and identify socket 24; this is the 24-pin socket which lies between two 40-pin sockets, one holding the 6502 microprocessor, and the other the INS 8255 I/0 chip. If a colour board is fitted this will have to be removed before proceeding, as it partly obscures socket 24. Insert the Toolkit in socket 24 so that the end of the chip with the small semi-circular indent points towards the aluminium heatsink at the edge of the printed circuit board. Make sure that none of the pins of the EPROM have become bent, and that they are all in the socket. When you are sure that the Toolbox has been inserted correctly, check once more to be certain, if necessary refit the colour board, then replace the base and screws.

After the Acorn has been switched on the Toolkit is brought into operation by entering LINK #AF00 or LINK 44800. The screen will clear and the message

PROGRAMMERS TOOLKIT

will be displayed (the lower case letters appear in inverse video on the screen). All the toolkit commands are now available. Whenever you press the 'Break' key you must re-link the Toolkit to activate the extra commands. The Toolkit has no effect on the operation of the Atom until it has been initialised with the LINK command.

THE CASSETTE INTERFACE

The Toolkit provides you with a cassette system which operates at 1200 Baud, in addition to the standard 300 Baud

system. The Baud rate is automatically set to 1200 by the command LINK #AFOO (LINK 44800). Each block of data can be recorded and read in one quarter of the time tahen at 300 Baud. Because the Atom inserts a long delay between blocks, you will get the maximum advantage from the increased Baud rate by using unnamed files. You may find that the level setting of your recorder needs to be adjusted for 12O0 Baud. It is also important to keep the tape head clean to ensure reliable operation.

Of course, data must be read at the Baud rate at which it was recorded. You can switch between Baud rates in the Toolkit by using the command VECTOR; VECTOR 0 sets the rate to 300, VECTOR 1 to 1200. Alternatively, you can retain the Baud rate at 300 when you initialise the Toolbox by using the command LINK 04F04 (LINK 44804).

At either Baud rate the Toolkit provides visible indication of the operation of the cassette interface. Each byte of data being sent to or received fram the cassette is displayed at the top right corner of the screen. You can thus always see that a 'load' is proceeding correctly.

When using the Toolkit cassette routines it is essential that you respond to the prompts 'PLAY TAPE' and 'RECORD TAPE' by pressing the Space Bar, as specified in the Atom manual, and not the Return key.

THE COMMANDS

COMMAND	PAGE	COMMAND	PAGE	COMMAND	PAGE	COMMAND	PAGE
AUTO	3	FIND	2	OFF.	4	STOP	5
BEEP	4	HEX.	5	ON ERROR	7	TRACE.	4
CURSOR	4	IHEX	3	POP.	5	VAR,	3
DATA	6	INKEY.	5	READ	6	VECTOR	4
DELETE	3	KEY.	5	RESTORE.	6	WHILE. ,	6
DUMP	4	LTRACE	4	RENUMBER	2	XIF.	5
ELSE	5	LVAR	3	STEP	3	ZERO	3
ENDWHILE	6						

In the command descriptions below, if any part of the command name is enclosed in brackets the command may be abbreviated by terminating it with a full stop anywhere after the start of the brackets. Thus REN(UMBER) can be replaced by REN., RENU., etc. Commands marked 'Direct' can only be used in the

direct mode, that is , they cannot be used within programs.

FI(ND) "string" Direct

This command lists the numbers of all lines in which the string of characters within the quotation marks occurs. The string may be up to 32 characters long.

REN(UMBER) x, y Direct

This command renumbers a program, starting with x in steps of y. If only one number is entered, this will be used for the start and step. If no number is entered the default value is 10. The program is first checked to see that renumbering will not produce a line number greater than 32767, which is not permitted in Atom Basic. All GOTOs and GOSUBs are changed to match the line to which they refer, with the exception of 'indirect' jumps, such as GOS. (A+3*B). The RENUMBER routine lists the new number of all lines containing such indirection, so that you can edit the program after renumbering. If a program contains many indirect GOTOs and GOSUBs, you may find that some of the line numbers scroll off the screen. Enter CONTROL/N and do another RENUMBER. The indirect lines uill now be listed one screenful at a time.

You should not start a program with a line number greater

than 255. When you recover a program after a 'Break' by the use of OLD, the Atom assumes that the first byte of the first line number is zero; a number greater than 255 is therefore changed by OLD. Far example, 1000, which is stored as #03, #E8, is changed to 232, Stared as #00, #E8.

AU(TO) x, y Direct

This initialisas the automatic generation of line numbers for use in Basic programs, starting at x in steps of y. If only one number is entered after AUTO, this will be used for the start and step; if no number is entered. the start and step will be 10. Generation of the line numbers is turned on and off with CONTROL/A; if CONTROL/A is entered beware the start and step have been initialisad line numbers will be produced, but the start and step will be unpredictable. When the routine is turned on, a new line number is produced whenever RETURN is pressed.

DE(LETE) x, y Direct

All lines in a program from x to y inclusive are deleted. DE. x deletes line x; DE. x, deletes all lines from x to the end of the program; DE., x deletes all lines from the start of the program to x. An error message is produced if there is no

line x or line y in the program.

V(AR) [#] Direct

Prints the vaues of variables A - Z in two columns on the screen; if followed by the symbol #, it prints the values in hexadecimal.

LV(AR) [#] VIA Direct

Outputs the values of the variables A - Z to a printer connected to the Atom's Centronics interface. If fallowed by the symbol # the values are printed in hexadecimal.

ZE(RO)

This command can be used directly from the keyboard or in programs to set variables A - Z to zero.

H(EX) yyyy Direct

The bytes stared in memnry, starting at address yyyy, are tabulated in hexadecimal and in ASCII. The format of each line is:- address (hexadecimal), four bytes of data (hexadecimal), the four ASCII characters corresponding to those bytes. Eight lines are printed, the routine then waits for a Icy to be pressed. The SPACE bar causes a further block of eight lines to appear, while the RETURN key terminates the command. The starting address can be specified in decimal or hexadecimal.

IH(EX) YYYY Direct

Tabulates the hexadecimal codes in memory starting at address yyyy in instruction format; that is, one, two or three bytes of data appear on a line, d╗pending on the number of bytes used by the instruction represented by the first byte. As in the case of HEX, eight lines are tabulated at a time, and yyyy may be specified in decimal or hexadecimal.

S(TEP) VIA Direct

For this command to operate, the Atom must be fitted with

the VIA chip (6522), and link 2 (the IRQ link) must be in place. The command allows line-by-line single stepping when a program is RUN. The current line number is displayed .at the top left of the screen (unless a graphics mode is selected). The program halts at each line and waits for a key to be pressed. You can quit the program at any point with the escape key, when you can, for example, list the values of the variables with the VAR command.

TR(ACE) x VIA Direct

Can only be used with the VIA chip and IRQ link in place. Operates in the same way as the step command, except that it pauses for a predetermined period on each line. The delay is set by x, which should be a number, variable or expression in the range 0 - 255. If x is not entered, a value of #55 is used, which gives approximately 3 steps in tw0 seconds.

LT(RACE) VIA Direct

Prints the numbers of the lines as they are executed on a printer connected the the Atom's Centronics interface.

O(FF)

Turns off the TRACE, LTRACE, and STEP commands. These commands must be turned off before you edit a program - editing a program with any of these three commands still in operation could result in corruption of the program.

DU(MP)

The printer is enabled, the contents of the screen are printed out line-by line (unless a graphics mode is selected), and the printer is turned off.

VEC(TOR) x

If x is zero the command sets the cassette Baud rate to 300; if x is one, the Baud rate is 1200. If other values are entered, the command is ignored.

BE(EP) v, y

This generates a nate whose pitch depends on x and duration on y. Both x and y may be numbers, variables or expressions between 0 and 255 (if they are outside this range, only the least significant byte will be used). The lower the value of x the higher the note will be; if x is less than 8 no nate will be heard - this can be used to qive a programmable delay. The duration can lie between about 20 milliseconds (y=1) and six seconds (y=255).

'Space invader' sound effects can be produced with the BEEP command, e.g.,

```
100 FOR J=1 TO 5
110 FOR K=40 TO 80 STEP 4
120 BEEP K, 1
130 NEXT; NEXT; END
```

Music can be produced by first reading the pitches of the notes of the scale into an array (using READ and DATA statements, see below), and then specifying the desired array element as the

pitch in a series of BEEP statements. Alternatively, the durations and pitches of the notes of a tune can be placed in a

series of DATA statements, and you can then READ them as you play the tune. The note values are given in the table below.

N0TE	VALUE	VALUE	VALUE	NOTE	VALUE	VALUE	VALUE
A	246	121	60	D#	173	86	42
Bb	231	114	57	E	163	81	39
B	217	108	54	F	154	76	37
C	205	102	50	F#	145	72	35
C#	194	96	47	G	137	68	33
D	183	91	45	G#	129	64	51

CU(RSOR) x, y

Repositions the cursor to the xth column on line y, where x and y can be numbers, variables or expressions. The leftmost position corresponds to x=0, the rightmost to x=31; if x lies outside this range, only the least significant 5 bits are used. The top line of the screen corresponds to y=O, the bottom line to y=15; only the least significant 4 bits are used. If only the first value is entered, the cursor is repositioned to the xth column on the current line.

KEY A

The keyboard is scanned once and the value of the ASCII code of any key pressed is returned in the variable specified after the command. If no key was pressed, a zero is returned.

INK(EY) $A

This command operates in a similar way to KEY, but returns the character corresponding to the key pressed in the designated string. Normally one of the string variables A - Z will be used (this must have been previously dimensioned), but forms such as INKEY $TOP or INKEY $#B200 are valid. If no key has been pressed a null string is recurned. The LOCK, COPY, CURSOR CONTROL and RETURN keys will also give a null string.

STOP

If a program is misbehaving, you can insert STOP commands at several points throughout the program. When the STOP is reached the computer prints STOP AT, fallowed by the line number, and then waits for a key to be pressed before continuing. If

ESCAPE is pressed control returns to the Basic monitor, when you can list the variables with VAR. Once the program has been debugged, the STOP statements can be removed.

POP

This removes references to a current subroutine free the stack, so that you can jump directly fram the subroutine to any point in the main program, rather than going back to the command after the GOSUB which called the subroutine.

XIF ... THEN ... EL(SE)

The XIF command has the same action and the same syntax as the normal IF command in Atom Basic, except that it sets a flag (in location #A7) to indicate the result of the IF condition. THEN is used with XIF in the same way and with the same syntax as with IF. The ELSE command tests the flag set by the preceeding

XIF. If the condition in the XIF line was true, the whole line following ELSE will be skipped; if false, however, everything after the ELSE will be executed. For example:-

```
100 XIF A=1 AND B=0 THEN PRINT "CORRECT"
110 ELSE PRINT "THAT'S WRONG"
```

The ELSE must not be placed on the same line as XIF, because it would then be skipped if the condition were false. It can be placed anywhere after that line, and in fact the XIF could be followed by several lines containing ELSE, as they would all use the flag value set by the last XIF command. If an ELSE is encountered before an XIF it will not produce an error message. but the results will be unpredictable.

W(HILE) ... END(WHILE)

The statements between these two commands are executed repeatedly for as long as the condition specified in the WHILE command is true. This loop differes from the DO ... UNTIL loop in two repects:-

(a) The condition is tested at the start of the loop, not at the end. Thus if the condition is false on entry, the entire loop will be skipped.
(b) The loop is repeated while the condition is true; the DO ... UNTIL loop is executed whiie the condition is false.

WHILE can be followed by any testable integer expression (with the conjunctions AND and OR if desired). ENDWHILE takes no argument - it just serves as a marker for the end of the loop. WHILE/ENDWHILE loops can be nested inside each other up to a maximum depth of 18.

Please note that WHILE/ENDWHILE is a structured loop, and the structure should be adhered to. Don't try jumping in and out of the loop with GOTO commands.

READ .. DA(TA) ... RES(TORE)

These statements provide a convenient way of incorpocating data (numerical or character strings) in a program. A data painter is initialised to the start of the current text space by the RESTORE command. Each READ statement searches through the program for the next DATA.

DA(TA)

This can be followed by any string at characters, or by several strings separated by commas. Each time data is to be read, the next string will be taken. Depending on the form of the statement, either the data string will be taken exactly as it stands (i.e., all the characters between the commas), or it will be evaluated as an integer expression. Where numerical data is to be evaluated in this manner, ordinarily it will be simple numbers, such as:-

```
DATA 1, 10, 250, #FF
```

but there is no reason why a complicated expression such as ABSRND%6+4 should not be evaluated.

Where data is read as a string rather than as a numerical expression, any spaces in the string will also be read, except

that spaces immediately following DATA are ignored. If the first data string has leading spaces, place a comma after the word DATA. DATA statements must not be included in multiple statement lines; they must have a line to themselves and can not be preceeded by a label.

READ

This can take three different types of arguments - a variable, a string ($ followed by anything which evaluates to a valid address where the string will be stored), or an array element (the array must of course have been previausly dimensioned). Each READ statement may have more than one argument

if desired. Each argument will be read in turn e.g., READ $A, X, AA(X).

RE(STORE)

This command must be used before the first READ, to initialize the data pointer. It can be used at any other point in the program to reset the pointer.

ON(ERROR)

When this is encountered in a program the normal Basic error-handler vector (locations 16 and 17) is changed to the start of the statement immediately following ON ERROR, and the rest of the line, including any multiple statements, is skipped. Every time an error occurs the normal error message is suppressed and execution will recommence after the ON ERROR comand. All the interpreter stacks are cleared to ensure correct operation if the error occurs in a FOR, DO or WHILE loop or in a subroutine. This means that you should not jump into the middle of a loop or subroutine with an ON ERROR command; if you do, the end of the loop or subroutine will itself cause an error, sending control back to the ON ERROR statement, and the program will go round in circles.

A program can contain more than one ON ERROR statement. For example, if a program has a series of input statements, each one can be supplied with its own error handling routine of the type:-

```
100 ON ERROR PRINT "NUMBERS ONLY"
110 INPUT "TYPE A NUMBER",A
```

ERROR MESSAGES

Any programming errors will still produce the normal Atom error messages, even if the error occurred as the result of the incorrect use of a toolkit command. For example, if you try to renumber a program with too large a step, so that the line number would become greater than 32767, the message ERROR 109 <number too large) will be displayed. Nate that if the line number in an error message corresponds to to a READ statement, the error could be in the DATA statement currently being read. The line number of this DATA statement can be obtained by printing the value stored at #A8 and #A9, by entering PRINT !#A85#FFFF.

ADDRESSES USED BY THE TOOLKIT

The Toolkit makes use of a series of addresses in the free area on page zero (#80 - #AF), and these should not be used by programs. If a program does not contain any Toolkit commands, only location #86 need be avoided. Programs which contain Toolkit commands should confine their zero-page workspace to #90 - #A5. The function of the following addreses may be of interest:-

#80-#85 These locations are used by STEP and TRACE; problems will occur if yau use these commands with programs which access these addresses.

#86 If this location is zero, auto line numbering is turned off; if it contains a 1, it is turned on; any other value at this address prevents line numbering being turned off.

#8B Contains a number which is one less than the position in the command table of the current Toolbox command. Thus the numbers corresponding to the commands are:-

0 FIND	1 kEY	2 READ	3 DATA	4 RESTORE
5 WHILE	6 ENDW.	7 TRACE	8 LTRACE	9 STEP
10 OFF	11 VAR	12 LVAR	13 AUTO	14 RENUM.
15 STOP	16 BEEP	17 VECTOR	18 DUMP	19 ON E.
20 POP	21 ZERO	22 HEX	23 IHEX	24 CURSOR
25 INKEY	26 DELETE	27 ELSE	28 XIF	

#A6 The number of WHILE loops currently active is stored here

#A7 The XIF flag, zero if the condition in the last XIF statement was false.

#A8,#A9 The line number of the current DATA statement.

#AC,#AD The data pointer.

#AE Normally zero, this location is set to -1, when a Toolbox command is being executed.

The memory locations form #21C to #23F are used for the WHILE/ENDWHILE stack. This region is available for other purposes if a program does not contain these commands.