PROGRAMMER'S TOOLBOX

FITTING THE TOOLBOX

PLEASE READ AND UNDERSTAND THESE INSTRUCTIONS BEFORE PROCEEDING.

Your programmer's toolbox is supplied in a 4K EPROM which is inserted in socket 24 of your Acorn Atom. To fit the chip, turn the Acorn 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 1NS8255 I/O chip. If a colour board is fitted this will have to be removed as it partly obscures socket 24. Insert the Toolbox 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.

USING THE TOOLBOX

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

PROGRAMMER'S TOOLBOX
(C) program power

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

CASSETTE INTERFACE

The Toolbox 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 #AF00 (LINK 44800). Each block of data can be recorded and read in one quarter of the time taken at 300 Baud. Because the Atom inserts a long delay between blocks, you will get the maximum advantage from the increased speed by using unnamed files. You may find that the level setting of your recorder may need to be adjusted to operate at 1200 Baud. It is also important to keep the tape head clean to ensure reliable operation.

Of course, data must be read at the same Baud rate at which it was recorded. You can switch between Baud rates in the Toolbox by using the command VECTOR; VECTOR 0 sets the rate to 300, VECTOR 1 to 1200. Alternatively, you can retain the Baud rate at the standard value when you initialise the Toolbox by using the command LINK #AF04 (LINK 44804).

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

THE COMMANDS

Your Toolbox contains the following commands:-

Command	Page	Command	Page	Command	Page	Command	Page
AUTO	3	FIND	2	OFF	4	STOP	5
BEEP	4	HEX	3	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.

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 must not be longer than 32 characters. This command can only be used in 'Direct' mode, that is, it can not be used within a program.

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 the 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 numbers 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. You will now find that a single screen of line numbers is produced; the routine then pauses until a key is pressed, when renumbering will continue.

You should not start a program with a line number greater

than 255. When you recover a program by entering 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. E.g., 1000, which is stored as #03,#E8, is changed to 232, stored as #00, #E8

AU(TO) x, y Direct

This initialises 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 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 before the start and step have been initialised 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.,y deletes all lines from the start of the program to y. An error message is produced if there is no line x or line y in the program.

V(AR) [#] Direct

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

LV(AR) [#] Direct

Ouputs the values of variables A - Z to a printer connected to the Atom's Centronic interface. If followed by a # 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 stored in medmory, starting at address yyyy, are tabulated in hexadecimal and in ASCII. The format of each line is:- address (in hexadecimal), four bytes of data (hexadecimal) and the ASCII characters corresponding to those bytes. Eight lines are printed, the routine then waits for a key 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 hexadecimal or decimal.

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, depending 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-hand corner 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 the 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 is 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 two seconds.

LT(RACE) VIA Direct

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

O(FF) Direct

Turns off the TRACE, LTRACE, and STEP commands. These commands should be turned off before editing - 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) x, y

This generates a note 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 note will be heard - this can be used to give 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
130 NEXT:NEXT
```

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. Or the durations and pitches of the notes of a tune can be placed in a series of DATA statements, and then you can READ them as you play the tune.

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 is line 0, the bottom line is 15; only the least significant 4 bits are used. If the comma and the second value are omitted, the cursor is reposition to the xth column on the current line. The CURSOR command will work even if the cursor has been turned off by executing ?#E1=0. If the screen display has been disabled, for example, by PRINT $21, CURSOR will turn it on again.

KEY X

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 during the scan 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 $#8200 are valid. If no key has been pressed a null string is returned. Note that the LOCK, COPY, and CURSOR CONTROL keys produce a null string, because there is no ASCII equivalent for these control codes. The RETURN key will also produce a null string, because the ATOM treats a string which consists of only a carriage return as a null string.

ST(OP)

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, followed by the current 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 values of variables with VAR. Once the program has been debugged, the STOP statements can be removed.

POP

This removes references to a current subroutine from the stack, so that you can jump directly from 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 under the same conditions 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=" PRINT "CORRECT"
110 ELSE PRINT"THAT'S WRONG"
```

The ELSE must not be placed on the same line as the 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 statement is true. This loop differs 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, whereas both DO ... UNTIL and FOR ... NEXT loops are always executed at least once.

(b) The loop is repeated while the condition is true, i.e., until it is false; with DO ... UNTIL the loop is executed while the condition is false, i.e., until it is true.

WHILE can be followed by any testable integer expression (complete with the conjuctions 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 incorporating data (numerical or character strings) in a program. A data pointer is intialised to the start of the current text space by the RESTORE command. Each READ statement searches through the program for the next piece of DATA.

DA(TA)

This can be followed by any string of 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 READ 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, ordinarily any spaces in the string will also be read. The exception is that spaces immediately after the keyword DATA are ignored. If the first data string has leading spaces, place a comma immediately 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 data string will be stored), or an array element (the subscript to the array can be any arbitrarily complicated expression, though of course the array must have previously been 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 commmand must be used before the firs t READ, to intialise the data pointer to the start of the current text space. It can be used at any other point in the program to reset the pointer.

ON(ERROR)

This command is used for error trapping. When it is encountered in a program the normal Basic error-handler vector (locations 16 and 17) is changed to the start of the staement immediatel following ON ERROR, and the rest of the line, including any multiple statements, is then skipped. Every time an error occurs the normal error message is suppressed and execution will recommence after the ON ERROR command. 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; 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, for example:-

```
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 a result of the incorrect use of a Toolbox 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
```

will be produced (number too large). Note that if the line number in an error message corresponds 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 !#A8&#FFFF
```

ADDRESSES USED BY THE TOOLBOX

The Toolbox makes use of a series of addresses in the free area on zero page (#80 - #AF), and these should not be used by a

program. If the program does not contain any Toolbox functions, then only location #86 should be avoided (this contains a flag for the AUTO command). The STEP and TRACE commands use locations #80, #81, #82 and #85; problems will occur if you use these commands with programs which also access these addresses. Programs that contain Toolbox commands should confine their zero-page workspace to the region from #90 to #A5. The function of the following addresses may be of interest:-

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

#8B Contains a number which is one ~~greater~~ less than the position in the command table of the current Toolbox command. For example, HEX is the 23rd command in the table; when this command is executed the value at #8B is #16 (22 in decimal).

The order of the commands in the table is:-

1 FIND	2 KEY	3 READ	4 DATA	5 RESTORE
6 WHILE	7 ENDW.	8 TRACE	9 LTRACE	10 STEP
11 OFF	12 VAR	13 LVAR	14 AUTO	15 RENUM.
16 STOP	17 BEEP	18 VECTOR	19 DUMP	20 ON ERR.
21 POP	22 ZERO	23 HEX	24 IHEX	25 CURSOR
26 INKEY	27 DELETE	28 ELSE	29 XIF	

#A6 The number of WHILE loops curently 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 from #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.

* *

BOTH THE TOOLBOX PROGRAM AND THESE NOTES ARE
COPYRIGHT (C) PROGRAM POWER, 1981

* - * - * - * - * - * - * - * - * - * - * - *

PROGRAM POWER

5, WENSLEY ROAD, LEEDS, LS7 2LX

* o * o * o * o * o * o * o * o * o * o * o *