3 The Keyboard.

The Key Manager is the pack associated with the keyboard. All the attributes of the keyboard are generated and controlled by the Key Manager. These attributes include repeat speed, shift and control keys, function keys and key translation. The joysticks are also scanned by the Key Manager.

The Key Manager has three levels of operation. The lowest level scans the keyboard, the middle level converts the key pressings into key values and the top level converts the key values into characters. The user may access the Key Manager at whichever level is most appropriate for a given program. It is usually unwise, however, for a program to mix accesses at different levels.

3.1 Keyboard Scanning.

The keyboard is completely software scanned. This scan occurs automatically every fiftieth of a second (see KM SCAN KEYS). The keyboard hardware is read and a bit map noting which keys are pressed is constructed. This bit map is available for testing if specific keys are pressed (see KM TEST KEY). As the bit map is constructed keys that are newly pressed are noted and markers are stored in a buffer until needed. If no newly pressed keys are found then the last key pressed may be allowed to repeat if it is still down (see section 3.5). The keyboard is 'debounced' in that a key must be released for two consecutive scans before it is marked as released in the bit map. This 'debouncing' hides multiple operations of the key switch as it opens or closes.

At this stage only four keys are treated specially. The two shift keys and the control key are not stored in the key buffer themselves. Instead, when any other marker is stored the states of the shift and control keys are noted and put into the buffer as well. The escape key generates a marker as normal but may also have other effects depending on whether the break mechanism is armed (see section 3.6).

There is a problem with scanning the keyboard. If three keys at the corners of a rectangle in the key matrix are all pressed at the same time then the key at the fourth corner appears to be pressed as well. There is no way to avoid this problem as it is a feature of the keyboard hardware. All key combinations used by the firmware (and the BASIC) have been especially designed to avoid this effect.

3.2 Key Translation.

When the user asks for a key (KM WAIT KEY or KM READ KEY) the next key pressed marker is read from the key buffer. The marker is converted to a key number and this is looked up in one of three translation tables.

Which table is used depends on whether the shift and control keys were pressed when the key was pressed. One table is used if the control key was pressed, another is used if either shift key was pressed but control was not, the third is used if neither shift nor control keys were pressed. The contents of these tables can be altered by the user as required (by calling KM SET CONTROL, KM SET SHIFT and KM SET TRANSLATE respectively).

The value extracted from the table may be a system token, an expansion token or a character. Expansion tokens and characters are used by the top level of the Key Manager (see 3.3 below) and are passed up from the middle level when they are found in a table. There are three system tokens, which are obeyed immediately they are found in a table. After obeying the token the next marker is read from the buffer and translated.

The default translation tables are described in Appendix II.

The immediately obeyed System tokens are:

a. Ignore (#FF)
The key pressed is to be ignored.

b. Shift lock (#FE)
The shift lock is to be toggled (turned on if it is currently off and turned off if it is on).

c. Caps lock (#FD)
The caps lock is to be toggled (turned on if it is off and off if it is on).

3.3 Characters from the Keyboard.

When the user asks the top level for a character (KM WAIT CHAR or KM READ CHAR) a key is fetched from the middle level. If this is a character (#00..#7F or #A0.#FC) then it is passed to the user. If it is one of the 32 expansion tokens (# 80.. # 9F) then the string associated with the token is looked up. The characters in this string are passed to the user one at a time with each request for a character until the end of the string is reached.

There is only one character with a special meaning at this level. This is character #EF which is produced when pressing the escape key generates a break event (see section 3.6). It has no effects, it is merely a marker for the place in the buffer where a break event was generated. It is intended to be used to allow all characters before the break to be discarded. This character is not generated by the translation tables and thus cannot be changed by altering them.

A single 'put back' character is supported. When the user puts back a character this character will be returned by the next call to the top level of the Key Manager. This is intended for use by programs that need to test the next character to be read from the keyboard without losing it (when processing breaks perhaps).

In V1.1 firmware it is possible to call KM FLUSH to discard any unused or unwanted characters so that subsequent calls to KM READ CHAR or KM READ KEY will not return values from a previous input. The same effect can be achieved in V1.0 Firmware by repeatedly calling KM READ CHAR until it returns with carry false to indicate that there are no more characters available.

3.4 Shift and Caps Lock.

a. Shift lock

When shift lock is engaged then the keys pressed are translated as if a shift key is pressed.

The shift lock is toggled by a system token (see 3.2 above) which is normally generated by pressing CTRL and CAPS LOCK.

b. Caps lock

When caps lock is engaged then alphabetic characters read from the keyboard are converted to their upper case equivalents. This case conversion is applied before expansion tokens are expanded and so expansions are not capitalised.

The caps lock is toggled by a system token (see 3.2 above) which is normally generated by pressing CAPS LOCK (without control).

In V1.1 firmware it is possible to set the state of the locks as if the SHIFT or CAPS LOCK keys had been pressed by calling KM SET LOCKS.

3.5 Repeating keys.

There is a table, which the user can alter as desired, that specifies which keys are allowed to repeat when held down (see KM SET REPEAT). The default setting for this table is described in Appendix 111. Briefly, the default is to allow all keys to repeat except the ESC, TAB, CAPS LOCK, SHIFT, ENTER and CTRL keys and the 12 keys in the numeric keypad (the function keys).

The speed at which keys repeat and the delay before the first repeat can be set by the user (see KM SET DELAY). The default speed produces up to 25 characters a second with a 0.6 second start up delay.

A key is allowed to repeat if the following conditions are satisfied:

1/ The appropriate time has passed since the key was first pressed or it last repeated.

2/ The key is still pressed.

3/ No other key has been pressed since the key was first pressed.

4/ The key is marked as allowed to repeat in the repeat table.

5/ There are no keys stored in the key buffer.

Condition 5 above means that the repeat speed and start up delay set the maximum speed at which a key is allowed to repeat. If a program is slow about removing keys from the buffer then the generation of keys will adjust itself to this. Thus it is impossible to get a large number of keys stored in the buffer simply by holding a key pressed.

3.6 Breaks.

Breaks can occur when the keyboard scanner detects that the ESC key is pressed. When the escape key is found to be pressed the indirection KM TEST BREAK is called to deal with the break. The default setting for this routine tests whether the SHIFT, CTRL and ESC keys and no others are pressed. If so then the system is reset (by executing an RST 0), otherwise the break mechanism is invoked.

If the break mechanism is disarmed then no action is taken other than the normal insertion of the marker for the escape key into the key buffer. If the break mechanism is armed then two additional operations take place. Firstly, a special marker is placed into the key buffer that will generate character #EF when it is found (irrespective of the translation tables). This is intended to be used to allow the characters which were in the buffer before the break occurred to be discarded. Secondly, the synchronous break event is 'kicked'.

The break mechanism can be armed or disarmed at any time (by calling KM ARM BREAK or KM DISARM BREAK). The default state is disarmed. When a break is detected the mechanism is disarmed automatically which prevents multiple breaks from occurring.

The method BASIC uses to handle breaks should serve as a model for other programs. BASIC's actions are as follows:

> The break mechanism is armed. After each BASIC instruction the synchronous event queue is polled and if a break event is found (because it has been kicked as explained above) the break event routine is run.
>
> The break event routine stops sound generation (SOUND HOLD) and then it discards all characters typed before the break occurred by reading characters from the keyboard (KM READ CHAR) until either the buffer is empty or the break event marker (character #EF) is found. BASIC then turns the cursor on (TXT CUR ON) and waits for the next character to be typed (KM WAIT CHAR).
>
> If the next character is the escape token (character #FC - the default value generated by the ESC key) then a flag is set to make BASIC abandon execution (or run the user's ON BREAK GOSUB subroutine) and the break event routine returns.
>
> If the next character is any character other than escape then the break will be ignored. If it is any character other than space then this is 'put back' (KM CHAR RETURN). Before the event routine returns the cursor is turned off (TXT CUR OFF), sound generation is restarted (SOUND CONTINUE) and the break mechanism is rearmed. BASIC then continues as if nothing had happened.

When reading or writing from the cassette the ESC key is handled in a different manner which is described in section 8.12.

3.7 Function Keys and Expansion Tokens.

The Key Manager allows for 32 expansion tokens (values #80.A9F) which may be placed in the key translation tables. Each token is associated with a string which is stored in the expansion buffer.

When the user asks the top level for a character a key is fetched from the middle level. If this key is a character it is passed straight back. However, if it is an expansion token then the string associated with the token is looked up. The characters in this string are passed out one at a time with each request for a character until the end of the string is reached. Values # 80.. # 9F and # EF, # FD.. # FF in the expansion string are treated as characters and are not expanded or obeyed.

The user may set the string associated with an expansion token (see KM SET EXPAND) and may cause any key on the keyboard to generate an expansion token. The default settings for the expansion tokens and the keys with which they are normally associated are given in Appendix IV. The user may also set the size and location of the expansion buffer (see KM EXP BUFFER); the default buffer is at least 100 bytes long.

3.8 Joysticks.

There may be two joysticks connected to the system. These are both scanned in the same way as keys on the keyboard. Indeed, the second joystick occupies the same locations in the key matrix as certain other keys and is indistinguishable from them. The state of the joysticks can be determined by calling the routine KM GET JOYSTICK.

Because the joysticks are scanned like keys the pressing of joystick buttons can be detected like any other key. Firstly, individual direction or buttons can be tested in the key bit map (see section 3.1) by calling KM TEST KEY. Secondly, the joystick buttons generate characters when they are pressed (providing the translation tables are set suitably) and these characters can be detected. The major problem with this latter method is that the rate of generation of characters depends on how fast the keyboard is set to repeat. If the repeat speed is increased to make the joystick more responsive then the keyboard may become impossible to use.

See Appendix I for the numbering of the keys and joystick buttons and see Appendix II for the default translation tables.