13 Firmware Jumpblocks.

There are a number of jumpblocks provided by the firmware. The largest of these is the main firmware jumpblock. This is intended to be used by programs to access the firmware routines in the lower ROM. BASIC, for instance, uses these jumps. Note, however that the firmware does not use this jumpblock for internal communication with itself. This means that altering the jumpblock will cause BASIC to behave differently but will not cause the firmware to behave differently.

The most important jumpblock is the indirections jumpblock. The indirections are jumps that are used by the firmware at key points. This allows the user to alter the action of firmware routines. The entries in this jumpblock are not intended for the user to call, only for the firmware to call. Altering an indirection is the method to make the firmware behave differently.

The remaining two jumpblocks are associated with the Kernel. One is a jumpblock to allow the user to call various useful Kernel routines to do with changing ROM states and the like. The other is not a jumpblock as such, just an area where the routines are at published addresses. These are general utility routines and restarts. In general neither of these areas should be altered by the user.

The routines in these jumpblocks are briefly listed below. More complex descriptions of the routines can be found in sections 14, 15 and 16.

13.1 The Main Jumpblock.

The main firmware jumpblock lies in RAM between addresses #BB00 and #BD39. Each entry in the jumpblock occupies three bytes and is initialized to use LOW JUMP restarts (RST 1) that cause the lower ROM to be enabled, so that the firmware routines can be run, and the upper ROM to be disabled, so that the screen memory is accessible while the firmware is running.

After the jumpblock has been set up at EMS it is not altered by the firmware until the system is reinitialised. If any entries are changed then it is the user's responsibility to undo the reinitialised. This can be achieved by calling JUMP RESTORE which completely initializes the jumpblock.

13.1.1 Entries to the Key Manager.

The Key Manager deals with the keyboard and the joysticks.

INITIALISATION

0	#BB00	KM INITIALIZE	Initialize the Key Manager.

1	#BB03	KM RESET	Reset the Key Manager - clear all buffers, restore standard key expansions and indirections.
CHARACTERS			
2	#BB06	KM WAIT CHAR	Wait for the next character from the keyboard.
3	#BB09	KM READ CHAR	Test if a character is available from the keyboard.
4	#BB0C	KM CHAR RETURN	Return a single character to the keyboard for next time.
5	#BB0F	KM SET EXPAND	Set an expansion string.
6	#BB12	KM GET EXPAND	Get a character from an expansion string.
7	#BB15	KM EXP BUFFER	Allocate a buffer for expansion strings.
KEYS			
8	#BB18	KM WAIT KEY	Wait for the next key from the keyboard.
9	#BB1B	KM READ KEY	Test of a key is available from the keyboard.
10	#BB1E	KM TEST KEY	Test if a key is pressed.
11	#BB21	KM GET STATE	Fetch Caps Lock and Shift Lock states.
12	#BB24	KM GET JOYSTICK	Fetch current state of the joystick(s).
TRANSLATION TABLES			
13	#BB27	KM SET TRANSLATE	Set entry in key translation table without shift or control.
14	#BB2A	KM GET TRANSLATE	Get entry from key translation table without shift or control.
15	#BB2D	KM SET SHIFT	Set entry in key translation table when shift key is pressed.
16	#BD30	KM GET SHIFT	Get entry from key translation table when shift key is pressed.
17	#BB33	KM SET CONTROL	Set entry in key translation table when control key is pressed.

18	#BB36	KM GET CONTROL	Get entry form key translation table when control key is pressed
REPEATING			
19	#BB39	KM SET REPEAT	Set whether a key may repeat.
20	#BB3C	KM GET REPEAT	Ask if a key is allowed to repeat.
21	#BB3F	KM SET DELAY	Set start up delay and repeat speed.
22	#BB42	KM GET DELAY	Get start up delay and repeat speed.
BREAKS			
23	#BB45	KM ARM BREAK	Allow break events to be generated.
24	#BB48	KM DISARM BREAK	Prevent break event from being generated.
25	#BB4B	KM BREAK EVENT	Generate a break event (if armed).

14.1.2 Entries to the Text VDU.

The Text VDU is a character based screen driver.

INITIALIZATION			
26	#BB4E	TXT INITIALISE	Initialize the Text VDU.
27	#BB51	TXT RESET	Reset the Text VDU - restore default indirections and control code functions.
28	#BB54	TXT VDU ENABLE	Allow characters to be placed on the screen.
29	#BB57	TXT VDU DISABLE	Prevent characters from being placed on the screen.
CHARACTERS			
30	#BB5A	TXT OUTPUT	Output a character or control code to the Text VDU.
31	#BB5D	TXT WR CHAR	Write a character onto the screen.
32	#BB60	TXT RD CHAR	Read a character from the screen.

33	#BB63	TXT SET GRAPHIC	Turn on or off the Graphics VDU character writing option.
WINDOWS			
34	#BB66	TXT WIN ENABLE	Set size of the current text window.
35	#BB69	TXT GET WINDOW	Get the size of the current text window.
36	#BB6C	TXT CLEAR WINDOW	Clear current window.
CURSOR			
37	#BB6F	TXT SET COLUMN	Set cursor horizontal position.
38	#BB72	TXT SET ROW	Set cursor vertical position.
39	#BB75	TXT SET CURSOR	Set cursor position.
40	#BB78	TXT GET CURSOR	Ask current cursor position.
41	#BB7B	TXT CUR ENABLE	Allow cursor display - user.
42	#BB7E	TXT CUR DISABLE	Dissallow cursor display - user.
43	#BB81	TXT CUR ON	Allow cursor display - system.
44	#BB84	TXT CUR OFF	Dissallow cursor display -system.
45	#BB87	TXT VALIDATE	Check if a cursor position is within the window.
46	#BB8A	TXT PLACE CURSOR	Put a cursor blob on the screen.
47	#BB8D	TXT REMOVE CURSOR	Take a cursor blob off the screen.
INKS			
48	#BB90	TXT SET PEN	Set ink for writing characters.
49	#BB93	TXT GET PEN	Get ink for writing characters.
50	#BB96	TXT SET PAPER	Set ink for writing text background.
51	#BB99	TXT GET PAPER	Get ink for writing text background.
52	#BB9C	TXT INVERSE	Swap current pen and paper inks.
53	#BB9F	TXT SET BACK	Allow or dissallow background being written.

54	#BBA2	TXT GET BACK	Ask if background is being written.
MATRICES			
55	#BBA5	TXT GET MATRIX	Get the address of a character matrix.
56	#BBA8	TXT SET MATRIX	Set a character matrix.
57	#BBAB	TXT SET M TABLE	Set the user defined matrix table address.
58	#BBAE	TXT GET M TABLE	Get user defined matrix table address.
CONTROL CODES			
59	#BBB1	TXT GET CONTROLS	Fetch address of control code table.
STREAMS			
60	#BBB4	TXT STR SELECT	Select Text VDU stream.
61	#BBB7	TXT SWAP STREAMS	Swap the states of two streams.

14.1.3 Entries to the Graphics VDU

The Graphics VDU deals with individual pixels.

INITIALIZATION			
62	#BBBA	GRA INITIALISE	Initialize the Graphics VDU.
63	#BBBD	GRA RESET	Reset the Graphics VDU -restore standard indirections.
CURRENT POSITION			
64	#BBC0	GRA MOVE ABSOLUTE	Move to an absolute position.
65	#BBC3	GRA MOVE RELATIVE	Move relative to current position.
66	#BBC6	GRA ASK CURSOR	Get the current position.
67	#BBC9	GRA SET ORIGIN	Set the origin of the user coordinates.
68	#BBCC	GRA GET ORIGIN	Get the origin of the user coordinates.

WINDOW

69	#BBCF	GRA WIN WIDTH	Set left and right edges of the graphics window.
70	#BBD2	GRA WIN HEIGHT	Set top and bottom edges of the graphics window.
71	#BBD5	GRA GET W WIDTH	Get the left and right edges of the graphics window.
72	#BBD8	GRA GET W HEIGHT	Get the top and bottom edges of the graphics window.
73	#BBDB	GRA CLEAR WINDOW	Clear the graphics window.

INKS

74	#BBDE	GRA SET PEN	Set the graphics plotting ink.
75	#BBE1	GRA GET PEN	Get the current graphics plotting ink.
76	#BBE4	GRA SET PAPER	Set the graphics background ink.
77	#BBE7	GRA GET PAPER	Get the current graphics background ink.

PLOTTING

78	#BBEA	GRA PLOT ABSOLUTE	Plot a point at an absolute position.
79	#BBED	GRA PLOT RELATIVE	Plot a point relative to the current position.

TESTING

80	#BBF0	GRA TEST ABSOLUTE	Test a point at an absolute position.
81	#BBF3	GRA TEST RELATIVE	Test a point relative to the current position.

LINE DRAWING

82	#BBF6	GRA LINE ABSOLUTE	Draw a line to an absolute position.
83	#BBF9	GRA LINE RELATIVE	Draw a line relative to the current position.

CHARACTER DRAWING

84	#BBFC	GRA WR CHAR	Put a character on the screen at the current graphics position.

14.1.4 Entries to the Screen Pack

The Screen Pack interfaces the Text and Graphics VDUs to the screen hardware. Screen functions that affect both text and graphics (e.g. ink colours) are located in the Screen Pack.

INITIALIZATION

85	#BBFF	SCR INITIALISE	Initialize the Screen Pack.
86	#BC02	SCR RESET	Reset the Screen Pack – restore standard indirections, ink colours and flash rates.

SCREEN HARDWARE

87	#BC05	SCR SET OFFSET	Set the offset of the start of the screen.
88	#BC08	SCR SET BASE	Set the area of RAM to use for the screen memory.
89	#BC0B	SCR GET LOCATION	Fetch current base and offset settings.

MODE

90	#BC0E	SCR SET MODE	Set screen into new mode.
91	#BC11	SCR GET MODE	Ask the current screen mode.
92	#BC14	SCR CLEAR	Clear the screen (to ink zero).
93	#BC17	SCR CHAR LIMITS	Ask size of the screen in characters.

SCREEN ADDRESSES

94	#BC1A	SCR CHAR POSITION	Convert physical coordinates to a screen position.
95	#BC1D	SCR DOT POSITION	Convert base coordinates to a screen position.
92	#BC20	SCR NEXT BYTE	Step a screen address right one byte.

97	#BC23	SCR PREV BYTE	Step a screen address left one byte.
98	#BC26	SCR NEXT LINE	Step a screen address down one line.
99	#BC29	SCR PREV LINE	Step a screen address up one line.
INKS			
100	#BC2C	SCR INK ENCODE	Encode an ink to cover all pixels in a byte.
101	#BC2F	SCR INK DECODE	Decode an encoded ink.
102	#BC32	SCR SET INK	Set the colours in which to display an ink.
103	#BC35	SCR GET INK	Ask the colours an ink is currently displayed in.
104	#BC38	SCR SET BORDER	Set the colours in which to display the border.
105	#BC3B	SCR GET BORDER	Ask the colours the border is currently displayed in.
106	#BC3E	SCR SET FLASHING	Set the flash periods.
107	#BC41	SCR GET FLASHING	Ask the current flash periods.
MISCELLANEOUS			
108	#BC44	SCR FILL BOX	Fill a character area of the screen with an ink.
109	#BC47	SCR FLOOD BOX	Fill a byte area of the screen with an ink.
110	#BC4A	SCR CHAR INVERT	Invert a character position.
111	#BC4D	SCR HW ROLL	Move the whole screen up or down eight pixel lines (one character).
112	#BC50	SCR SW ROLL	Move an area of the screen up or down eight pixel lines (one character).
113	#BC53	SCR UNPACK	Expand a character matrix for the current screen mode.
114	#BC56	SCR REPACK	Compress a character matrix to the standard form.

115	#BC59	SCR ACCESS	Set the screen write mode for the Graphics VDU
116	#BC62	SCR PIXELS	Write a pixel to the screen ignoring the Graphic VDU write mode.

LINE DRAWING

117	#BC5F	SCR HORIZONTAL	Plot a purely horizontal line.
118	#BC62	SCR VERTICAL	Plot a purely vertical line.

13.1.5 Entries to the Cassette Manager

The Cassette Manager handles reading files from tape and writing files to tape.

INITIALIZATION

119	#BC65	CAS INITIALISE	Initialize the Cassette Manager - close all streams, set default speed and enable messages.
120	#BC68	CAS SET SPEED	Set the write speed.
121	#BC6B	CAS NOISY	Enable or disable prompt messages.

READING FILES

122	#BC6E	CAS START MOTOR	Start the cassette motor.
123	#BC71	CAS STOP MOTOR	Stop the cassette motor.
124	#BC74	CAS RESTORE MOTOR	Restore previous state of cassette motor.

READING FILES

125	#BC8C	CAS IN OPEN	Open a file for input.
126	#BC8F	CAS IN CLOSE	Close the input file properly.
127	#BC7D	CAS IN ABANDON	Close the input file immediately.
128	#BC80	CAS IN CHAR	Read a character from the input file.
129	#BC83	CAS IN DIRECT	Read the input file into store.

130	#BC86	CAS RETURN	Put the last character read back.
131	#BC89	CAS TEST EOF	Have we reached the end of the file yet?

WRITING FILES

132	#BC8C	CAS OUT OPEN	Open a file for output.
133	#BC8F	CAS OUT CLOSE	Close the output file properly.
134	#BC92	CAS OUT ABANDON	Close the output file immediately.
135	#BC95	CAS OUT CHAR	Write a character to the output file.
136	#BC98	CAS OUT DIRECT	Write the output file directly from store.

CATALOGUING

137	#BC9B	CAS CATALOG	Generate a catalogue from the tape.

RECORDS

138	#BC9E	CAS WRITE	Write a record to tape.
139	#BCA1	CAS READ	Read a record from tape.
140	#BCA4	CAS CHECK	Compare a record on tape with the contents of store.

14.1.6 Entries to the Sound Manager.

The Sound Manager controls the sound chip.

INITIALIZATION

141	#BCA7	SOUND RESET	Reset the Sound Manager - shut the sound chip up and clear all sound queues.

SOUND QUEUES

142	#BCAA	SOUND QUEUE	Add a sound to a sound queue.
143	#BCAD	SOUND CHECK	Ask if there is space in a sound queue.

144	#BCB0	SOUND ARM EVENT	Set up an event to be run when a sound queue becomes not full.

SOUNDS

145	#BCB3	SOUND RELEASE	Allows sounds to happen.
146	#BCB6	SOUND HOLD	Stop all sound in mid flight.
147	#BCB9	SOUND CONTINUE	Restart sound after they have been stopped.

ENVELOPES

148	#BCBC	SOUND AMPL ENVELOPE	Set up an amplitude envelope.
149	#BCBF	SOUND TONE ENVELOPE	Set up a tone envelope.
150	#BCC2	SOUND A ADDRESS	Get the address of an amplitude envelope.
151	#BCC5	SOUND T ADDRESS	Get the address of a tone envelope.

13.1.7 Entries to the Kernel

The Kernel handles synchronous and asynchronous events. It is also in charge of the store map and switching ROMs on and off. Apart from the entries listed below, the Kernel has its own jumpblock and a number of routines whose addresses are published. These extra entries are listed in sections 13.3 and 13.4 below.

INITIALIZATION

152	#BCC8	KL CHOKE OFF	Reset the Kernel - clears all event queues etc.
153	#BCCB	KL ROM WALK	Find and initialize all background ROMs.
154	#BCCE	KL INIT BACK	Initialize a particular background ROM.
155	#BCD1	KL LOG EXT	Introduce an RSX to the firmware.
156	#BCD4	KL FIND COMMAND	Search for an RSX or background ROM or foreground ROM to process a command.

FRAME FLYBACK LIST

157	#BCD7	KL NEW FRAME FLY	Initialize and put a block onto the frame flyback list.
158	#BCDA	KL ADD FRAME FLY	Put a block onto the frame flyback list.
159	#BCDD	KL DEL FRAME FLY	Remove a block from the frame flyback list.

FAST TICKER LIST

160	#BCE0	KL NEW FAST TICKER	Initialize and put a block onto the fast tick list.
161	#BCE3	KL ADD FAST TICKER	Put a block onto the fast tick list.
162	#BCE6	KL DEL FAST TICKER	Remove a block from the fast tick list.

TICK LIST

163	#BCE9	KL ADD TICKER	Put a block onto the tick list.
164	#BCEC	KL DEL TICKER	Remove a block from the tick list.

EVENTS

165	#BCEF	KL INIT EVENT	Initialize an event block.
166	#BCF2	KL EVENT	'Kick' an event block.
167	#BCF5	KL SYNC RESET	Clear synchronous event queue.
168	#BCF8	KL DEL SYNCHRONOUS	Remove a synchronous event from the event queue.
169	#BCFB	KL NEXT SYNC	Get the next event from the queue.
170	#BCFE	KL DO SYNC	Perform an event routine.
171	#BD01	KL DONE SYNC	Finish processing an event.
172	#BD04	KL EVENT DISABLE	Disable normal synchronous events.
173	#BD07	KL EVENT ENABLE	Enable normal synchronous events.
174	#BD0A	KL DISARM EVENT	Prevent an event from occurring.

ELAPSED TIME

175	#BD0D	KL TIME PLEASE	Ask the elapsed time.
176	#BD10	KL TIME SET	Set the elapsed time.

13.1.8 Entries to the Machine Pack

The Machine Pack provides an interface to the machine hardware. Most packs use Machine to access any hardware they use. The major exception is the Cassette Manager which, for speed reasons, performs its own hardware access.

PROGRAMS

177	#BD13	MC BOOT PROGRAM	Load and run a foreground program.
178	#BD16	MC START PROGRAM	Run a foreground program.

SCREEN

179	#BD19	MC WAIT FLYBACK	Wait for frame flyback.
180	#BD1C	MC SET MODE	Set the screen mode.
181	#BD1F	MC SCREEN OFFSET	Set the screen offset.
182	#BD22	MC CLEAR INKS	Set all inks to one colour.
183	#BD25	MC SET INKS	Set colours of all the inks.

PRINTER

184	#BD28	MC RESET PRINTER	Reset the printer indirection.
185	#BD2B	MC PRINT CHAR	Translate a character to the Centronics port.
186	#BD2E	MC BUSY PRINTER	Test if the Centronics port is busy.
187	#BD31	MC SEND PRINTER	Send a character to the Centronics port.

SOUND CHIP

188	#BD34	MC SOUND REGISTER	Send data to a sound chip register.

13.1.9 Entries to Jumper

Jumper sets up the main jumpblock.

INITIALISATION

189	#BD37	JUMP RESTORE	Restore the standard jumpblock.

13.2 Firmware Indirections

The firmware indirections listed here are taken at key points in the firmware thus allowing the user to provide substitute routines for many firmware actions, without having to replace a complete firmware package. These indirections are not intended for the user to call - there is usually a higher level routine in the main firmware jumpblock that is more suitable.

The indirections are set up by the pack to whom they apply whenever its reset (or initialize) routine is called and during EMS; they are not otherwise altered by the firmware.

The indirections are all three bytes long and use standard jump instructions (#C3). If a ROM state other than upper ROMs disabled and lower ROM enabled is required then the appropriate restart instruction might be substituted (see section 2.3). The indirections are to be found between #BDCD and #BDF3.

At this level of operation very little validation is carried out. If incorrect parameters are passed or a substitute routine corrupts a register in defiance of the documented interface then the firmware will probably cease to function as expected.

More detailed descriptions of these routines can be found in section 15.

13.2.1 Text VDU Indirections

0	#BDCD	TXT DRAW CURSOR	Place the cursor blob on the screen (if enabled).
1	#BDD0	TXT UNDRAW CURSOR	Remove the cursor blob from the screen (if enabled).
2	#BDD3	TXT WRITE CHAR	Write a character onto the screen.
3	#BDD6	TXT UNWRITE	Read a character from the screen.
4	#BDD9	TXT OUT ACTION	Output a character or control code.

13.2.2 Graphics VDU Indirections

5	#BDDC	GRA PLOT	Plot a point
6	#BDDF	GRA TEST	Test a point
7	#BDE2	GRA LINE	Draw a line

13.2.3 Screen Pack Indirections

8	#BDE5	SCR READ	Read a pixel from the screen.
9	#BDE8	SCR WRITE	Write a pixel(s) to the screen using the current graphics write mode.
10	#BDEB	SCR MODE CLEAR	Clear the screen to ink 0.

13.2.4 Keyboard Manager Indirections

11	#BDEE	KM TEST BREAK	Test for break (or reset).
13	#BDF4	KM SCAN KEYS	Scan the keyboard.

13.2.5 Machine Pack Indirections

12	#BDF1	MC WAIT PRINTER	Print a character or time out.

14.3 The High Kernel Jumpblock

The high Kernel jumpblock is provided to allow the user to turn ROMs on and off and to access memory underneath ROMs while they are enabled. The entries in this jumpblock are not all jump instructions, some entries are the start of routines, thus the user should not alter any of the entries in this jumpblock. The high Kernel jumpblock occupies store from #B900 upwards. More detailed descriptions of the routines in it can be found in section 16.

0	#B900	KL U ROM ENABLE	Turn on the current upper ROM.
1	#B903	KL U ROM DISABLE	Turn off the upper ROM.
2	#B906	KL L ROM ENABLE	Turn on the lower ROM.
3	#B909	KL L ROM DISABLE	Turn off the lower ROM.
4	#B90C	KL ROM RESTORE	Restore the previous ROM state.
5	#B90F	KL ROM SELECT	Select a particular upper ROM.

6	#B912	KL CURR SELECTION	Ask which upper ROM is currently selected.
7	#B915	KL PROBE ROM	Ask class and version of a ROM.
8	#B918	KL ROM DESELECT	Restore the previous upper ROM selection.
9	#B91B	KL LDIR	Move store (LDIR) with ROMs disabled.
10	#B91E	KL LDDR	Move store (LDDR) with ROMs disabled.
11	#B921	KL POLL SYNCHRONOUS	Check if an event with higher priority than the current event is pending.

13.4 The Low Kernel Jumpblock.

The Kernel provides a number of useful routines in the area of memory between #0000 and #003F. These are available, in some cases, both as a published routine address and as a restart instruction. In general the routines are available both in ROM and RAM so whether the lower ROM is enabled does not matter. There are also a couple of areas available for the user to patch to trap RST 6s and interrupts from external hardware.

The low Kernel jumpblock is not intended for the user to alter. However, it may be necessary to alter it under certain circumstances. In particular the INTERRUPT ENTRY (by patching the jump at #0038) or the RESET ENTRY (by patching the bytes from #0000..#0007). If a program does change any locations in the jumpblock (other than those in the USER RESTART or EXT INTERRUPT areas) then it is the program's responsibility to ensure that the lower ROM is enabled or the original contents are restored when any other programs runs. In particular the program must sort out the state when interrupts occur (hence the need to patch the INTERRUPT ENTRY).

More detailed descriptions of the routines in this jumpblock can be found in section 17.

#0000	**RST 0**	RESET ENTRY	Completely reset the machine as if powered up.
#0008	**RST 1**	LOW JUMP	Jump to lower ROM or RAM, takes an inline 'low address' to jump to.
#000B		KL LOW PCHL	Jump to lower ROM or RAM, HL contains the 'low address' to jump to.
#000E		PCBC INSTRUCTION	Jump to address in BC.

#0010	**RST 2**	SIDE CALL	Call to a sideways ROM, takes inline 'side address' to call.
#0013		KL SIDE PCHL	Call to a sideways ROM, HL contains 'side address' to call.
#0016		PCDE INSTRUCTION	Jump to address in DE.
#0018	**RST 3**	FAR CALL	Call a routine in any ROM or RAM, takes an inline address of the 'far address' to call.
#001B		KL FAR PCHL	Call a routine in any ROM or RAM, C and HL contain the 'far address' to call.
#001E		PCHL INSTRUCTION	Jump to address in HL.
#0020	**RST 4**	RAM LAM	LD A,(HL) with all ROMs disabled.
#0023		KL FAR ICALL	Call a routine in any ROM or RAM, HL points at the 'far address' to call.
#0028	**RST 5**	FIRM JUMP	Jump to lower ROM, takes an inline address to jump to.
#0030	**RST 6**	USER RESTART	ROM version saves current ROM state in #002B, turns the lower ROM off and jumps to the RAM version. RAM version may be patched by the user between #0030 and #0037 inclusively.
#0038	**RST 7**	INTERRUPT ENTRY	This restart is not available as it is used for interrupts (Z80 interrupt mode 1).
#003B		EXT INTERRUPT	When an interrupt occurs on the expansion port the firmware calls location #003B in RAM. The user may patch between #003B and #003F inclusive to trap this occurrence.