MILES GORDON TECHNOLOGY plc

SAM COUPE TECHNICAL MANUAL

version 3.0

The purpose of this manual is to introduce hardware and software developers to the internal and external workings of the SAM Coupe computer. To this end it is a technical document assuming a certain amount of technical knowledge.

It is MGT's policy that the SAM Coupe and associated MGT products should be fully documented to help third party software and hardware developers take full advantage of all the Coupe's advanced features. If you are developing a product for the Coupe and require further clarification or explanation MGT technical staff will be pleased to assist.

All production machines contain ROM 1.0. This manual details information for ROM 1.0 and ROM 1.2. In the period April/May 1990 all Coupe owners will be issued with an enhanced ROM, issue 2.0, free of charge. The ROM 1.2 referred to in this manual is a pre-production version of ROM 2.0.

written by
Bruce Gordon a Andy Wright

With additional information by
Gary Thomas & Adrian Parker

Supplementary information on the use of the SAA1099 sound chip
by Andy Graharn

With special Thanks to
Bob Brenchley, Simon Goodwin, W.Ettrick Thomson, and Nev Young

While every effort has been made to ensure that the contents of this manual are correct in every particular, it must be appreciated that MGT cannot guarantee that it is definitive since we have a policy of continual product enhancement.

Published by Miles Gordon Technology plc
Century Park Valley Way Swansea SA6 8QP
Tel: 0792 791100 Fax: 0792 791175

TECHNICAL SUPPORT FOR PROFESSIONAL DEVELOPERS

More and more hardware and software developers are working on SAM Coupe products.

HGT is keen to support the efforts of developers. To this end we have created the MGT SAM Coupe Developers Group. For a fee of £50.00 we offer members of the Developers Group the following benefits:

Unlimited telephone support from our Software Engineers.

Unlimited telephone support from our Hardware Engineers.

Free and first-off-the-press updates to the Technical Manual.

Free copies of supplementary information provided by developers, (where permission to distribute such information has been given).

Hardware port address allocation for hardware developers. This will avoid the possibility of two devices Using the same port address(es).

SAM Compatible: if a product, hardware or software, works with the Coupe we can issue 'SAM Compatible' stickers for the product case or packaging.

Developers Group registration number to ensure priority service.

If you require further information, or wish to join the Developers Group, please either *write* to, or phone, our Technical Department whose staff will be pleased to help you.

CONTENTS

THE SAM COUPE HARDWARE

CONTENTS

THE SAM COUPE SOFTWARE

CONTENTS

THE SAM COUPE DISK OPERATING SYSTEM

THE SAM COUPE HARDWARE

The SAM Coupe is a modern micro-computer using a Z80B microprocessor running at 6MHz. It has 256K RAM as standard, with an expansion socket to allow the memory to be doubled to 512K with a small expansion board.

The machine's main controller is a VLSI VGT-200 gate array (ASIC), customised to carry out the main tasks of processor/video contention, paging and management of all memory, video memory mapping, colour palette table allocation, and all input/output port control. There are four modes of video operation with a hardware capacity to show 16 colours from a palette of 128 on each screen line.

The sound for the machine is generated by a Philips SAA1099, with a capability in stereo of six channels over eight octaves with 256 tones per octave, two white noise generators, six amplitude controllers and two envelope wave shapers.

GENERAL SPECIFICATIONS

CPU	Z80B microprocessor running at 6MHz
Special	Customised VLSI 10,000-gate ASIC chip
ROM	32K x 8 ROM, 150nS, containing SAM BASIC, disk bootstrap, BIOS
RAM	256K upgradeable to 512K (256K x 4 100ns DRAM)
Sound	Philips SM 1099 Synthesizer: 6 channels, 8 octaves, stereo with amplitude and envelope control, plus choice of wave form.
Graphics	Motorola MC 1377P video Chip. ASIC serves as graphics processor, offering four modes:

Mode	1	32 x 24 character cells per screen, each cell capable of 2 colours; 16 colours selectable from 12e; Spectrum-attribute compatible.
Mode	2	As mode 1, but with 32 x 192 cells, each cell capable of 2 colours: 16 colours selectable from 128.
Mode	3	80 column text display - 512 x 192 pixel screen; each pixel selectable for colour; 4 colours per line selectable from 128.
Mode	4	256 x 192 pixel graphics screen; each pixel selectable for colour; 16 colours selectable per line from 128.

In all modes, colours may be redefined at line interrupt, allowing all 128 colours to be displayed on-screen.

UHF (TV channel 36), through power supply unit. Colour composite video, digital and linear RGB, all through SCART.

Atari-standard joystick (dual capability with splitter cable).

Mouse - Coupe standard.

Light-pen, Light-gun - Coupe standard. Audio output socket.

Domestic cassette recorder.

MIDI In, MIDI Out (MIDI Through, via software switch). Network - screened microphone cable with 7 pin DIN connectors.

RS232 and parallel printer via external }4GT interface connected at expansion port.

64-pin Euroconnector for further peripherals.

Disk Drives 1 or 2 removable and internally mounted 3.5” ultra-slim Citizen drives, 1 Mb unformatted, 780K formatted.

Keyboard 72 full travel keys, membrane type, including 10 function keys (software defined).

Power Consumption 11.2 Watts.

Shock	- Operating 3 G	
	- Non-operating 60 0	
Vibration	- Operating	5 - 500Hz / 0.5 G
	- Non-operating	5 - 500Hz / 2 G
Environmental: Ambient Temperature		Operating 5 - 45 C
		Storage -20 - 50 C
Relative Humidity		Operating 20 - 80%
	Wet Bulb Maximum 29,4 C, No Condensation	
Reliability:	MTBF: 10,000 POH; MTTR: 30 Mins;	
	Component Life: >5 years	
Weight	2.26Kg (4.97 lb)	
Power Supply	Primary Input voltage	240v AC 50Hz
	Secondary Output voltage	5v DC 2A Maximum Rating
		12v DC 2OOmA Maximum Rating

INTERFACES -

All connections to the Coupe except the removable internal disk drives, and the RAM expansion socket are made via the rear panel. please refer to the Input/Output Port map where reference is made to bits and registers.

- MIDI IN -

Standard 7 pin DIN type connector. This serial input, working at 31.25 Kbaud is fed via an opto-isolator to a serial/parallel converter, which interrupts the CPU when a data byte has been assembled in the MIDI-IN register (253 dec). This INPUT can also be read by bit 7 of the Video Memory Page Register VMPR (252 dec). This connector is also used by the network.

PIN	SIGNAL
1	NET - LOOP
2	N.C.
3	NET + LOOP
4	MIDI + IN
5	MIDI - IN
6	NET - LOOP
7	NET + LOOP

- MIDI OUT -

Standard 7 pin DIN type connector. By writing a data byte to the MIDI-OUT register (253 dec), the MIDI outputs a 7.5 mA current at 31.25 Kbaud. when transmitting, bit 1 of the PEN register (TXFMST of register 248 dec), is set. This OUTPUT can also be driven by bit 7 of the VMPR (252 dec). An internal through connection is made from MIDI IN to OUT by setting bit 6 THROM - through MIDI) of the BORDER register (254 dec). This connector is also used by the network, An interrupt is given on completion of the transmission of MIDI data.

PIN	SIGNAL
1	NET - LOOP
2	GND
3	NET + LOOP
4	NIDI + OUT
5	MIDI - OUT
6	NET - LOOP
7	NET + LOOP

- BREAK (NMI) BUTTON -

When pressed, this button forces the CPU to address (0066H) where it is vectored to any assignable address in the memory map. A versatile function for programmers to assign a vector, to enable its use as an ESC, BREAK or CRASH key. When the Disk Operating System Spectrum Emulator is loaded this button is used to activate the snapshot facility.

- JOYSTICK -

Standard 9 pin 'D' type plug. The joystick interface has standard ATARI connections except that it has an extra strobe line for a second joystick together with a line carrying 5 volts. MGT will allow for dual joystick control with the second joystick having a special plug-socket connector. The joystick is read by the Key-board port (254 dec) and overlays numeric keys 1 to 5 for second joystick and 6 to 0 for the first joystick.

PIN	SIGNAL
1	UP
2	DOWN
3	LEFT
4	RIGHT
5	0 VOLTS
6	FIRE
7	+5 VOLTS
8	STROBEL 1
9	STROBEL 2

- MOUSE PORT -

Proprietary 5 pin DIN type connector. The MGT mouse interrupts the CPU to request a read of its X and Y signal lines. The software driver is integral to the ROM and overlays the cursor control keys. The mouse input can be read by Keyboard port (254 dec) with address lines A8 to A15 set high. (RDMSEL}.

PIN	SIGNAL
1	DOWN
2	UP
3	CTRL
4	LEFT
5	RIGHT
6	MSE INT.
7	RDMSEL
8	+5 VOLTS
SCREEN	GROUND.

- RESET BUTTON -

When pressed this button causes the CPU to clear and restart at memory location (0000H). It also resets the following:

1. Floppy Disk Controller (Disk 1 and 2)
2. Memory page registers
3. MIDI receive and transmit registers
4. Border register

It does not reset the colour look up table, the sound chip registers or the LINE INTerupt register (249 dec) value. The ROM normally initialises LINE INT as part of its initialisation.

- CASSETTE JACK -

Standard 3.5mm mono jack socket. This is a bi-directional line. when outputting it should be connected to the MIC socket of a standard cassette tape recorder; when inputting it should be connected to the EAR socket of the recorder. It makes no difference to the Coupe if EAR and MIC are connected together, but only some tape recorders allow this, because of positive feedback.

- LIGHT PEN & AUDIO PORT -

Standard 5-pin, 180 degree, DIN type connector. This connector has a dual function of light-pen/gun input as well as stereo sound output capable at driving a HI-FI system on the AUX input.

PIN	SIGNAL
1	+5 VOLTS
2	AUDIO LEFT OUTPUT
3	0 VOLTS
4	SPEN INPUT
5	AUDIO RIGHT OUTPUT

- POWER INPUT -

Regulated DC input of 12 Volts at 200 milliamps and 5 Volts at 2 Amps. A fully loaded machine will have a power consumption of approximately 15 watts.

PIN	SIGNAL
1	+5 VOLTS
2	o VOLTS (SIGNAL GROUND)
3	o VOLTS (DIGITAL GROUND)
4	COMPOSITE VIDEO
5	+12 VOLTS
6	SOUND OUTPUT (MONO)

- EUROCONNECTOR (EXPANSION CONNECTOR) -

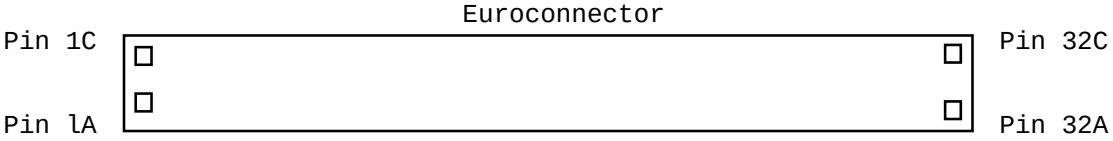
The Coupe's Euroconnector is a standard 64 pin type with rows A-C fitted. It has all the typical hardware signals found on other computer expansion ports. In addition, we include many extra hardware signals on the connector. This allows the hardware designer a hitherto unknown flexibility and control of a microcomputer from an external device.

SIGNALS

(NB row A is at the bottom of the Euroconnector, row C at the top).

PIN	SIGNAL	PIN	SIGNAL
lA	DBDIR	1C	IORQL
2A	RDL	2C	MREQL
3A	WRL	3C	HALTL
4A	BUSAKL	4C	NMIL
5A	WAITL	5C	INTL
6A	BUSREOL	6C	CD1
7A	RESETL	7C	CD0
8A	CM1L	8C	CD7
9A	REFRESHL	9C	CD2
1OA	0 VOLTS	10C	+5 VOLTS
1lA	AO	l1C	CD6
1ZA	Al	12C	CD5
13A	A2	13C	CD3
14A	A3	14C	CD4
15A	A4	15C	CPU CLK
16A	A5	16C	Al5
17A	A6	17C	A14
l8A	A7	18C	Al3
19A	A8	19C	A12
20A	A9	20C	All
21A	Al0	21C	DISK 2L
22A	MSEINTL	22C	ROMCSL
23A	XMEML	23C	EARMIC
24A	8 MHz	24C	DISK 1L
25A	RED 1	25C	PRINTL
26A	GREEN 1	26C	BLUE 1
27A	C SYNC	27C	ROMCSRL
28A	SPEN	28C	AUDIO RIGHT OUTPUT
29A	BLUE 0	29C	AUDIO LEFT OUTPUT
30A	RED 0	30C	COMP VIDEO
31A	BRIGHT	31C	GREEN 0
32A	+5 VOLTS	32C	0 VOLTS

when looking at the back of the Coupe's expansion connector, the pins are numbered as follows:



For details of Z80B timing signals refer to any standard Z80 text. Please note that we will be releasing the SAM Coupe Multi Expansion (SME) in the near future. (This product has been referred to as the Card Cage in some of our literature). Some of the signals described here must be included in any designs to ensure their correct operation when used with the SME.

DBDIR (lA) — This signal is not used by the computer but must be driven by all external devices to allow their use with the SME. The SME has buffers present to prevent undue loading of the machine. However, it is the responsibility of the hardware designer to ensure that the data bus transceiver has its direction bit set according to the operation (read or write) being performed. The operation should be as follows:

1. Processor sends out an address.
2. External device decodes address.
3. External device drives the DBDIR line, from an open collector output, LOW for a processor READ.
   It is configured as a WIRED-OR internally to the computer.
4. Normal handshaking is continued with the Z80B for data transfer.

RDL (2A} — Read signal from the processor.

WRL (3A} — Write signal from the processor.

BUSAKL (4A) — Bus acknowledge signal from the processor.

WAITL (5A) — Wait signal to processor.

BUSREQL (6A) — Bus request signal to external device.

RESETL (7A) — This is a system reset generated by a 10 Kilohm resistor charging a 47 microfarad capacitor. This gives a pulse of approximately 330ms duration. It is suggested that if a sharp reset edge is required, this signal is passed through a schmitt inverter.

CM1L (6A) — Op-code fetch signal from the processor

REFRESHL (9A) — Refresh signal from the processor.

0 V (lOA) — Zero Volt: not to be used as a supply rail.

A0 (11A) — Address lines A0 to Al0 from the processor are on pins 11A to 21A.

MSEINTL (22A) — Mouse interrupt signal also present on the mouse port. The signal is pulled low by an external device to notify the processor that the mouse co-ordinates have changed.

XMEML (23A) External memory signal which will be driven low when the processor requires to switch out the internal memory and 8ccess the external memory. Please note that only memory above address 8000H in each block of 64K will be switched out. It is the responsibility of the peripheral designer to ensure that the DIEM signal is included in the address decoding circuitry of the design, to avoid contention between the internal RAM and the external device.

8MHZ (24A) 8 MegaHertz clock signal.

RED1 (25A) Each colour is generated by three colour driver signals, the intensity of which is determined by three bits each. RED1 is the MSB (Nest significant Bit) of the red colour signal at the current pixel position.

GREEN1 (26A) GREEN1 is the MSB of the green colour signal (see pin 25A for a full description).

CSYNC (27A) Composite Sync for video circuitry as on the SCART connector.

SPEN (28A) Light pen signal (active high) which is normally low. When the light pen is touched to the screen and the raster passes the point of the pen, a positive going edge is received on this signal from the light pen socket. This causes the two registers LPEN and HPEN to contain the current x and y co-ordinates of the light pen, respectively.

BLUE0 (29A) BLUE0 is the second bit of the blue colour signal (see pin 25A for a full description).

RED0 (30A) REDO is the second bit of the red colour signal (see pin 25A for a full description).

BRIGHT (31A) BRIGHT is the LSB (Least significant Bit) of all colour signals (see pin 25A for a full description).

+5V (32A) +5 volt supply rail, up to 250 mA available.

IORQL (1C) Input/output Request becomes active when the address bus contains a current port address from the processor.

MREQL (2C) Memory Request becomes active when the address bus contains a current memory address front the processor.

HALTL (3C) Halt indicates from the processor that it has executed a HALT instruction and is waiting for an interrupt.

NMIL (4C) Non Maskable Interrupt causes the processor to execute a jump, from which it may be vectored anywhere by the user. It is used as a BREAK button by BASIC, and a SNAPSHOT button by the DOS. It is pulled up by a 10 Kilohm resistor internally, and should therefore be driven low by an open collector circuit.

INTL (5C} Maskable Interrupt is used to interrupt the processor from its current task if the interrupts are currently enabled.

CD1 (6C) Data lines Dl,D0,D7 and D2 are on pins 6C to 9C.

+5V (l0C) +5 Volt reference only.

CD6 (l1C) Data lines D6,D5,D3 and D4 are on pins l1C to 14C.

CPUCLK (15C) The 6 MegaHertz clock signal to the processor.

A15 (16C) Address lines Al5 to All are on pins 16C to 20C.

DISK2L (21C) DISK2 is the decoded base address for the eight ports of drive 2.

ROMCSL (22C) ROM chip select is pulled high to disable the internal ROM (Used in conjunction with ROMCSR on pin 27C)$_4$ It indicates when the ROM is being accessed.

EARMIC (23C) EARMIC is connected to the logic level side of the cassette port circuitry.

DISKiL (24C) DISK1 is the decoded base address for the eight ports of drive 1.

PRINTL (25C) PRINT is the decoded address for either of the two possible printer addresses (246 and 248).

BLUEl (26C) BLUE1 is the MSB of the blue colour signal (see pin 25A for a full description).

ROMCSRL (27C) ROMCSR is connected to the ROMCS (22C) pin via a 1 Kilohm resistor. If ROMCS is pulled high by an external circuit, then ROMCSR could be used as a chip select for an external ROM which would then take the place on the internal one.

AUDIORIGHT(28C The right audio signal, as on the light pen socket, is brought out here.(5ornV pp as per auxiliary standard input) It must be amplified before being applied to a speaker.

AUDIOLEFT(29C) The left audio signal is present on this pin. The same precautions and signal levels as for AUDIORIGHT apply.

CVID (30C) Composite Video, as sent to the modulator and SCART socket is present on this pin.

GREENO (31C} GREENO is the second bit of the green colour signal (see pin 25A for a full description).

0 V (32C) Zero Volt supply rail.

- SCART SOCKET -

SCART socket to handle all video and sound outputs. Please note that not all connections are standard.

PIN	SIGNAL	PIN	SIGNAL
1	AUDIO OUT RH	2	SPEN
3	AUDIO OUT LH	4	AUDIO EARTH
5	BLUE EARTH	6	BLUE TTL OUT
7	BLUE LIN. OUT	8	RED TTL OUT
9	GREEN EARTH	10	GREEN TTL OUT
11	GREEN LIN. OUT	12	+5V POWER IN
13	RED EARTH	14	CSYNC EARTH
15	RED LIN. OUT	16	CSYNC
17	C.VID EARTH	18	+12V POWER IN
19	C.VID OUT	20	BRIGHT TTL OUT
21	GND.		

- UHF OUTPUT -

UHF television standard video and mono sound are output on channel 36; the UHF output is in the power supply box.

- SLIMLINE SLOT-IN DISK DRIVES -

The disk drives used in the SAM Coupe are Citizen Slim Line OSDC type 3.5', 1 megabyte (formatted 780 kilobytes).

The Coupe can control up to two disk drives. Connections to the two drives are made via two 32-pin Euroconnectors, with rows A and B connected. Drive 1 is on the left of the machine and drive 2 on the right. A list of signals available at these sockets are shown below.

PIN	SIGNAL	PIN	SIGNAL
lA	0 VOLTS	1B	WR
2A	0 VOLTS	2B	AO
3A	0 VOLTS	3B	Al
4A	0 VOLTS	4B	D0
5A	0 VOLTS	5B	Dl
6A	0 VOLTS	6B	D2
7A	0 VOLTS	7B	D3
8A	0 VOLTS	8B	D4
9A	0 VOLTS	9B	D5
10A	0 VOLTS	10B	D6
11A	5 VOLTS	l1B	D7
12A	5 VOLTS	12B	8 MHz
13A	5 VOLTS	13B	RST
14A	5 VOLTS	14B	No connection
15A	5 VOLTS	15B	A2
166A	5 VOLTS	16B	DISK 1 OR DISK 2

- DISK DRIVE INTERFACE -

The SAM Coupe can be connected to one or two disk drives. These would normally be internal SAM disk drives. However, we have made provision for people to use an external disk drive, (Shugart 400 type 5.25" or 3.5"), if they add the SAM External Drive Interface (SDI). Both the internal drives and the SDI use the VL-1772-02 disk controller chip manufactured by VLSI.

NB All values are given in Decimal.

The SAM Coupe controls each disk drive via 8 I/O mapped ports, as listed below for both drives:

```
      Disk 1   224 to 231 inclusive.
      Disk 2   240 to 247 inclusive.
```

In each case, the address can be given by:

```
      Disk1base (224) + offset (0 to 7)
      Disk2base (240) + offset (0 to 7)
```

Both sets of offsets perform the same functions on their respective sides of the disk. The first four offsets (0-3) refer to the first side of the disk, and the second four offsets, (4-7) refer to the second side of the disk. Offsets 4-7 can be ignored for single sided disk drives.

The offset determines which register of the 1772 will be accessed as shown below:

OFFSET	READ	WRITE
0	Status(disk side 1)	Command (disk side 1)
1	Track 1	Track 1
2	Sector 1	Sector 1
3	Data 1	Data 1
4	status(disk side 2)	Command (disk side 2)
5	Track 2	Track 2
6	Sector 2	Sector 2
7	Data 2	Data 2

For example, to read the current contents of the data register on side 2 of Disk 1:

```
      IN A,(231)
```

where 231 = Disk1base (224) + offset (7)

For complete information on controlling the 1772 chip refer to the data sheets contained in Appendix H.

MEMORY

- INTRODUCTION -

The Coupe can take 512 Kbytes of RAM on board (with extra memory able to be added via the expansion connector using signal XMEML). The basic machine has 256Kb fitted, with an internal expansion connector for an additional 256Kb. The memory used is Dynamic Random Access Memory 256K x 4 bits - 100ns access time, 20 pin dual in-line plastic package.

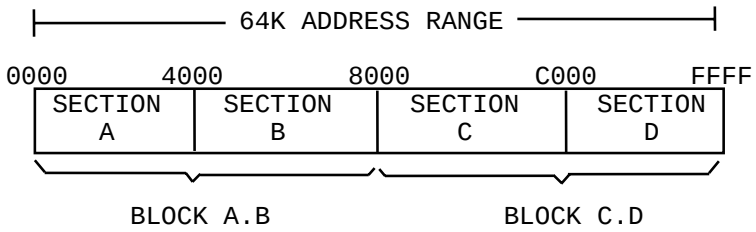
Because the Z80B CPU (central Processing Unit) is limited to a 64K addressing range, one of the functions of the ASIC (Application Specific Integrated Circuit) in the Coupe is to manage the addressing of this memory by splitting it into 32 pages of 16K.

The ASIC controls the paging via two 8-bit read/write registers.

LMPR (Low Memory Page Register)...I/O address 250dec
HMPR (High Memory Page Register}...I/O address 25ldec

The lower 5 bits of each register represent pages 0 to 31. (of course on the basic machine with 256K of memory fitted, the paging is from 0 to 15).

To illustrate the paging system used by the Coupe, it's best to envisage the 64K addressing range of the Z80 as 2 blocks 2 of sections of 16K, represented by the letters A.B and C.D:

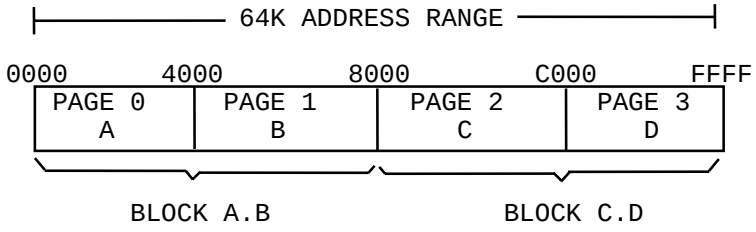


LMPR manages the block A.B, and the HMPR manages the block C.D

If we write 00H to the LMPR then page 0 of the memory is allocated to section A of the CPU address range. Section B is always automatically allocated one page above section A, in this case to page 1.

If we write 02H to the HMPR then page 2 of the memory is allocated to section C of the CPU address range. Section D is always automatically allocated one page above section C, in this case to page 3.

In our example the CPU address map would look like this:



SCREEN MODES

- INTRODUCTION -

There are four screen modes used in the Coupe, each using varying amounts of memory and each having different attributes for use by the programmer.

The hardware pointers which display the contents of the memory are controlled by the VMPR, Video Memory Page Register (252 dec).

By using the lower 5 bits of this register we can access up to 32 pages of video screen memory. It must be noted that modes 3 and 4 use 24 kilo-bytes, which go over the page boundary of 16 kilobybtes. Where this happens, the video addressing hardware wraps into the next page within the same block. For example:

By entering page 12 to the VMPR, video wraps to page 13

The same applies to any page and the principal of even to odd always applies.

- MODE 1 -

32 cells x 24 lines in 2 colours from 16 out of a palette of 128 colours, giving 768 character cells (8 x 8) using 6 kilobytes of bit-mapped memory and 0.75 kilo-bytes of attribute memory. This mode emulates Spectrum memory mapping.

- MODE 2 -

32 cells x 192 lines in 2 colours from 16 out of a palette of 128 colours, giving 6144 character cells (8 x 1) using 6 kilobytes of bit-mapped memory and 6 kilobytes of attribute memory. This mode has contiguous memory addressing in two blocks.

- MODE 3 -

512 pixels x 192 lines in 4 colours out of a palette of 128 colours, giving 98304 dots using 24 kilo-bytes of memory. This mode, when used with a character set 6 pixels wide, will give 85 characters per line.

- MODE 4 -

256 pixels x 192 lines in 16 colours out of a palette of 128 colours, giving 49252 dots using 24 kilo-bytes of memory. This mode is ideal for graphic display, and when used in conjunction with LINE INTerrupt register can display the full 128 colours on screen.

- LOCATING THE CURRENT SCREEN -

The SAM Coupe provides a port called Video Memory Page Register (address 252). To locate the base address of the currently displayed screen use the following BASIC program:

```
10 LET A=IN 252 BAND 31            REM Input the current page
20 LET BASE=(A+1) * 16384          REM BASE now equals start
                                   REM of screen area.
```

KEYBOARD

The Coupe has a full-travel 72 key keyboard. It is addressed as a 9 x 8 matrix, using two ports, KEYBOARD (254 dec) and STATUS (249 dec).

The 8 input lines are made up by the KEYBOARD port inputting the lower 5 bits represented by K1, K2, K3, K4 and K5, while the STATUS port inputs the upper 3 bits represented by K6, K7 and K8.

The 9 output scan lines are made up by the CPU address lines AD8, AD9, ADl0, ADl1, AD12, AD13, AD14, AD15 and the ASIC line RDMSEL.

An example of a typical keyboard scan:

```
      LD    HL,SCAN              ;set HL with buffer start
      LD    B,lllllll0bin        ;set upper address lines

LOOP  LD    C,HIKEY              ;port address K6 - K8
      IN    A,(C)
      AND   lll00000bin          ;strip unwanted bits
      LD    (HL),A               ;hold it

      LD    C, LOKEY             ;port address K1 - K5
      IN    A, (C)
      AND   000llllllbin         ;strip unwanted bits
      OR    (HL)                 ;make K1 - K8
      LD    (HL),A               ;save it

      INC   HL                   ;next location in buffer
      SCF                        ;set carry flag for
      RLC   B                    ;rotate next address line
      JR    C,LOOP               ;jump if not done

      IN    A,(C)                ;input RDMSEL line
      AND   00011111bin          ;strip unwanted bits
      LD    (HL),A               ;save it
      RET                        ;return to analyse scan
```

INPUT/OUTPUT PORTS

The SAM Coupe can address 64k of ports. Addresses 224 (EOH) to 255(FFH) are allocated to the Coupe itself. Refer to the diagram provided for an overview of these ports and their contents.

- WRITE (OUTPUT) PORTS -

SOUND ports (address port 511 dec) and (data port 255 dec)

The sound chip, a Philips SAA1099, is controlled through these two port addresses. The sound chip is controlled in BASIC through using the SOUND a,d command. See the attached Philips application notes and the SAA1099 supplements included elsewhere for further information.

VMPR - Video Memory Page Register (252 dec)

This read/ write register mainly controls the page addressing for the screen display memory.

Bit 0 R/W	BCD 1	of video page control.
Bit 1 R/W	RCD 2	of video page control.
Bit 2 R/W	BCD 4	of video page control.
Bit 3 R/W	BCD 8	of video page control.
Bit 4 R/W	BCD 16	of video bank control, used to switch between the banks of 256 kilobytes.
Bit 5 R/W	MDEO	first bit of screen mode control.
Bit 6 R/W	MDE1	second bit of screen mode control.
Bit 7 -/W	TXMIDI	output bit to directly drive the MIDI OUT channel.
Bit 7 R/-	RXMIDI	input bit from MIDI IN channel.

HMPR - High Memory Page Register (251 dec)

This read/write register is used mainly for the control of paging memory in the CPU's addressing range.

Bit 0 R/W	BCD	1	of high memory page control.
Bit 1 R/W	BCD	2	of high memory page control.
Bit 2 R/W	BCD	3	of high memory page control.
Bit 3 R/W	BCD	4	of high memory page control.
Bit 4 R/W	BCD	16	of high memory page control.
Bit 5 R/W	MD3S0		BCD 4 of the colour look-up address available only in mode 3.
Bit 6 R/W	MD3S1		BCD 8 of the colour look-up address available only in mode 3.
Bit 7 R/W	MCNTRL		If this bit is set when the CPU addresses high memory, then the external signal XMEM goes low and the Coupe looks on its expansion connector for memory sections C and D (addresses 32768 to 65536).

(See section entitled CLUT IN MODE 4 & MODE 3 in the text ahead)

LMPR - Low Memory Page Register (250 dec)

This read/write register is used mainly for the control of paging low memory in the CPU's addressing range.

Bit 0	R/W	BCD 1	of low memory page control.
Bit 1	R/W	BCD 2	of low memory page control.
Bit 2	R/W	BCD 4	of low memory page control.
Bit 3	R/W	BCD 8	of low memory page control.
Bit 4	R/W	BCD 16	of low memory bank control.
Bit 5	R/W	RAM0	when bit set high, RAM replaces the first half of the ROM (ie ROM0) in section A of the CPU address map.
Bit 6	R/W	ROM1	when bit set high, the second half of the ROM (ie ROM1) replaces the RAM in section D of the CPU address map
Bit 7	R/W	WPRAM	Write Protection of the RAM in section A of the CPU address map is enabled when this bit is set high.

MIDI OUT port (253 dec)

By writing a data byte to this port, the hardware automatically transmits through the MIDI out channel at 31.25 kbaud, the standard for MIDI protocol. Bit 4 of the STATUS register (249 dec) is set high when this register is full.

BORDER port (254 dec)

This output port mainly controls the border colour of the screen by supplying a 4-bit address to the Colour Look Up Table (CLUT), to enable a colour to be displayed during border time.

Bit 0	BCD 1	of CLUT address for border colour.
Bit 1	BCD 2	of CLUT address for border colour.
Bit 2	BCD 4	of CLUT address for border colour.
Bit 3	MIC	output control bit, normally set high.
Bit 4	BEEP	output control bit, normally set low.
Bit 5	BCD 8	of CLUT address for border colour.
Bit 6	THROM	bit set high to allow through MIDI operation
Bit 7	SOFF	bit set high to disable screen display, only active in screen modes 3 and 4, also re moves memory contention during off period.

LINE INTerrupt register (249 Dec)

This write-only register will cause an interrupt to the CPU at the end of the scan-line before the one matching its contents, (ie at the start of the right hand border). This works even for the first scan line. Line numbers are from 0 to 191. This function is always enabled, so to inhibit its operation a false line number from 192 to 255 can be entered. A useful function of this register can be to switch video modes or change colour lookup table values.

CLUT - Colour Look Up Table (base port 248 dec)

There are 16 write only 7 bit registers in the CLUT:

```
      colour       0    addresses register 0    on port      248
      colour       1    addresses register 1    on port      504
      colour       2    addresses register 2    on port      760
      colour       3    addresses register 3    on port     1016
      colour       4    addresses register 4    on port     1272
      colour       5    addresses register 5    on port     1528
      colour       6    addresses register 6    on port     1784
      colour       7    addresses register 7    on port     2040
      colour       8    addresses register 8    on port     2296
      colour       9    addresses register 9    on port     2552
      colour      10    addresses register A    on port     2808
      colour      11    addresses register B    on port     3064
      colour      12    addresses register C    on port     3320
      colour      13    addresses register D    on port     3576
      colour      14    addresses register E    on port     3832
      colour      15    addresses register F    on port     4088
```

Each register has 7 bits to represent 1 of 128 possible colours.

```
      Bit 0 BLUO        least significant bit of blue.
      Bit 1 REDO        least significant bit of red.
      Bit 2 GRNO        least significant bit of green.
      Bit 3 BRIGHT      half  bit intensity on all colours.
      Bit 4 BLU1        most  significant bit of blue.
      Bit S RED1        most  significant bit of red.
      Bit 6 GRN1        most  significant bit of green.
```

The registers at switch on will be loaded from the ROM with a default set of values corresponding to:

```
register 0  black  ( 0) register 8  black              (0)
register 1  blue   (16) register 9  bright blue       (17)
register 2  red    (32) register A  bright red        (34)
register 3  magenta(48) register B  bright magenta    (51)
register 4  green  (64) register C  bright green      (68)
register 5  cyan   (18) register D  bright cyan       (85)
register 6  yellow (96) register E  bright yellow    (102)
register 7  white (120) register F  bright white     (127)
```

The colour numbers are found by:

```
FOR c=0 TO 15 : PRINT PEEK (&55D8+c) : NEXT C
```

A typical routine for loading the CLUT from a memory table:

```
      LD    HL,TABLE    ;load HL with top of table
      LD    B,16        ;load B with size of table
      LD    C,248       ;load C with port address

      OTDR              ;execute a decrement B,
                        ;output to port (C) and
                        ;decrement HL until B=O
      RET               ;return to calling routine
```

CLUT IN MODE 4 & MODE 3

MODE 4

In screen mode 4, 4 bits are used to address the CLUT for the colour of a pixel. Within a byte, the most significant nibble refers to the first pixel and the least significant nibble refers to the second pixel.

MODE 3

The situation is similar in screen mode 3, however this time there are only 2 bits per pixel which address the CLUT. Out of an 8-bit data byte, the first two most significant bits are used as the address for the first pixel.

Normally, only 4 of the sixteen possible registers would be available. This is overcome by using an extra two bits from the high memory page register (HMPR-251 dec). Bit 5 of HMPR is used to access BCD 4 of the colour look up address and bit 6 of HMPR is used to access BCD 8 of the colour look-up address.

In this way, we can still access the 16 colours specified in the colour look-up table whilst having high resolution graphics.

- READ (INPUT) PORTS -

ATTRIBUTES register (255 dec)

This register enables the programmer to read the attributes of the currently displayed character cell in modes 1 and 2, and the third byte in every four displayed in modes 3 and 4.

KEYBOARD register (254 dec)

This read only register is mainly used for inputting the lower 5 bits of the keyboard matrix. It is also the input for the MOUSE when the address lines AD8 - AD15 are high.

Bit 0	K1	keyboard matrix line 1, Mouse Control.
Bit 1	K2	keyboard matrix line 2, Mouse Up.
Bit 2	K3	keyboard matrix line 3, Mouse Down/Button 2.
Bit 3	K4	keyboard matrix line 4, Mouse Left/Button 1.
Bit 4	K5	keyboard matrix line 5, Mouse Right/Button 3.
Bit 5	SPEN	light pen strobe/serial input bit.
Bit 6	EAR	serial input from EAR of cassette recorder.
Bit 7	SOFF	status bit show if external memory is set.

MIDI IN register (253 dec)

This read-only register generates an interrupt to the CPU when a data byte has been read from the MIDI serial input interface. The data must be read by the CPU before the next interrupt (typically 320 μS).

STATUS register (249 dec)

This read-only register is mainly used for reading interrupt status. Although all five interrupts go to the CPU operating under mode 1 interrupt, there is no way of knowing which one is requesting, therefore a read of this register is necessary. Approximate interrupt times are 20 μs long.

Bit 0	LINE	int	when low, signals the line interrupt register is requesting.
Bit 1	MOUSE	int	when low, signals the mouse requests the interrupt.
Bit 2	MIDIIN	int	when low, signals the MIDI channel has a data byte.
Bit 3	FRAME	int	when low, signals the frame scan has been completed (50 per/second).
Bit 4	MIDOUT	int	when low, indicates the NrDr out register has just completed data output.
Bit 5	K6		keyboard matrix line 6.
Bit 6	K7		keyboard matrix line 7.
Bit 7	K8		keyboard matrix line 8.

PEN registers (LPEN - 248 dec) (HPEN - 504 dec)

If the light pen is not connected, these read-only registers are continuously updating with the current position of the scan. The LPEN register is connected to the horizontal or X co-ordinate of the scan function and HPEN is connected to the vertical or Y coordinate of the scan function.

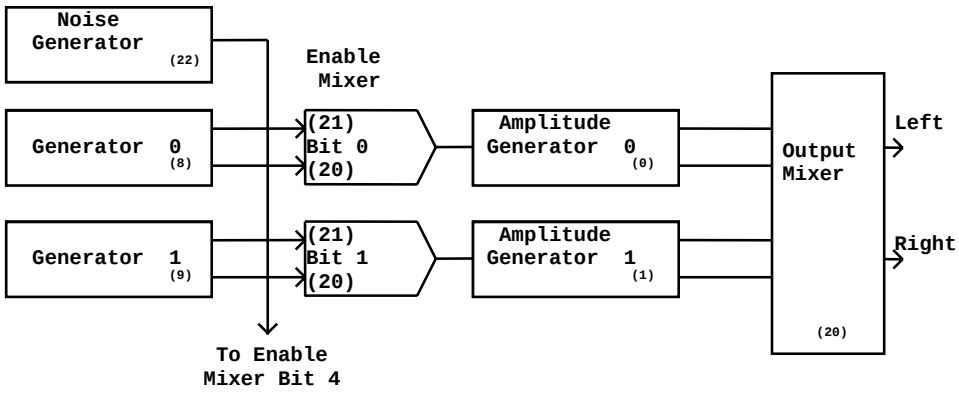
The first two bits of the LPEN register are:

Bit 0 - Inputs BCD 1 of the current colour look up table address.
Bit 1 - TXFMST is the status bit for MIDI OUT. When this is high
it shows that a byte is being transmitted.

PHILIPS SAA1099 SOUND CHIP

- INTRODUCTION -

The sound chip has six tone registers or oscillators and two noise generators. These can all be panned individually from left to right across the stereo field using one of the 256 possible positions, {ie. 16 volumes per channel, thus 16 x 16 combinations gives 256 positions). The position in the stereo field, relating to that particular oscillator, is controlled by its amplitude register. The overall sound can be enabled or disabled using output mixer (28). To hear sound, this register should be '1', to kill the sound, it should be 0.

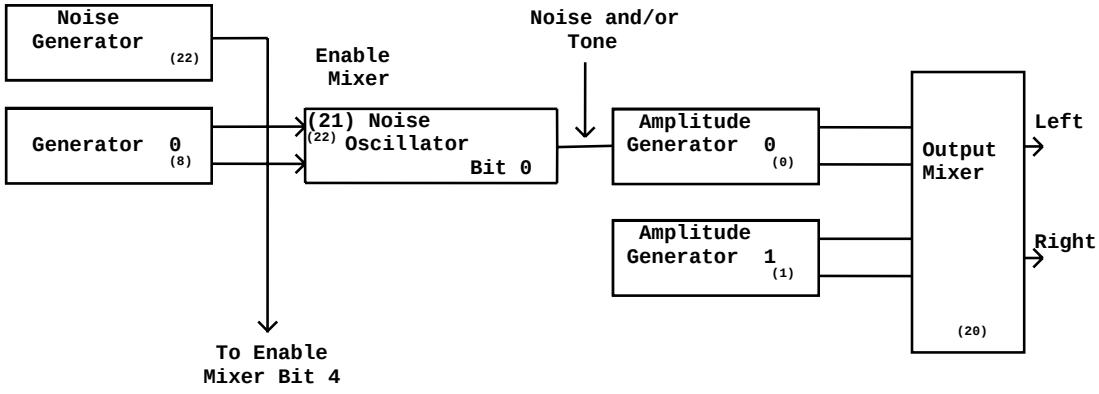


The six oscillators are split, three per noise generator. enables the two envelope registers to be used independently.

- NOISE & SOUND -

Mixer registers 20 and 21 mix the noise with each of the oscillators. Register 20 controls whether an oscillator's sound is passed or stopped, whilst register 21 controls the noise.

Example:



Using this mixer allows either noise or tone or both or nothing to be passed to a single amplitude register and so to a precise position in the stereo field.

- PITCH -

The pitch of each complete oscillator is controlled by two parameters. These are eight possible octaves per generator and 256 possible tones per octave. By carefully choosing octave and tone register data, a smooth transition is possible from the lowest frequency on this sound of 31Hz to the highest frequency of 7.81KHz.

- MUSIC -

Chromatic scales are possible using the table below. Please note also, that the tone numbers are also valid for the other octaves of the same note although tuning may be difficult towards the lower octaves.

- THE CHROMATIC SCALE

NB: All numbers given in decimal.

Note	Tone Number (decimal)	Octave Number	Required Frequency (Hz)	Actual Frequency (Hz)
Middle C	33	03	261.626	261.506
C#	60	03	277.183	277.162
D	85	03	293,665	293.427
D#	109	03	311.127	310.945
E	132	03	329.628	329.815
F	153	03	349.228	349.162
F#	173	03	369.994	369.B22
G	192	03	391.995	391.850
G#	210	03	415.305	415.282
A	227	03	440.000	440.141
A#	243	03	466.164	466.418
B	5	04	493.883	494.071
C	33	04	523.251	523.013

- OCTAVE REGISTERS -

Register 16 is an octave register altering the octave of tone registers 8 and 9. This is made possible by only using three bits per tone register to select the required octave.

Thus 2x2x2 gives us eight possible octaves as previously mentioned. So, to access the octave register for tone register 8, we must supply a number (0-7) to the lower 3 bits of register 16. If we needed to access the octave for tone register 9, we would then need to supply a number between 16 and 64 making sure that the first four bits DO-D3 contain zero, or the octave number for tone register 8 if in use.
Example:

Required Octave 5 For Tone Register 8 only

X = Don't Care

B7	B6	B5	B4	B3	B2	B1	B0
X	0	0	0	X	1	0	1

Number output to Register 16 = 5

Required Octave 6 For Tone Register 9 only

B7	B6	B5	B4	B3	B2	B1	B0
X	1	1	0	X	0	0	0

Number output to Register 16 = 96

- AMPLITUDE REGISTERS -

Any data sent to a particular amplitude register can precisely control the position of a sound in the stereo field.

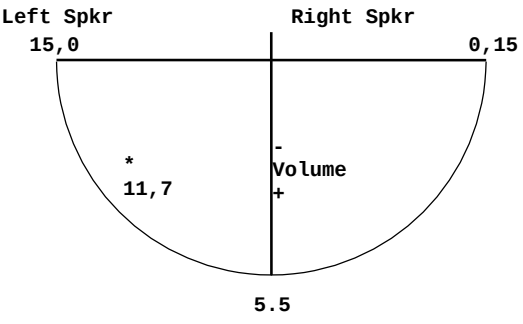
This is made possible by using the same technique as for the octave register. The only differences being that this time 4 bits are used to convey the volume of each channel, and the two 'halves' of the binary number refer to two channels (left and right) instead of two separate tone registers.

Volume 11 required on left channel. Volume 7 required on right channel for amplitude register 2

B7	B6	B5	B4	B3	B2	B1	B0
X	1	1	0	X	0	0	0

==== Right === ==== Left ====

This would place the sound towards the left side of the stereo field.



Imaginary Position Of Sound Emanating From Speakers in Stereo Field

- NOISE GENERATORS -

We have two noise generators again controlled by one register. The operation is very similar to that of the octave registers, but this time only 2 bits are used per noise generator. This gives 4 possible noise clock frequencies. Three of these options are preset - namely 0=31.25KHz, 1=15.6KHz and 2=7.8KHz. The fourth option (External) is controlled by the frequency of Generator 0 in the case of Noise generator 0 or Generator 3 in the case of Noise Generator 1. Using these generators, we are able to vary the 'pitch' of the noise and/or tone. If only continuously controllable noise is required, the relative tone mixer bit is disabled, but the relative noise mixer bit enabled. This technique is called using a tone generator to "Modulate" the frequency of the noise.

- ENVELOPE REGISTERS -

The envelope controllers can be found on registers 24 and 25.

Bit 0-Controls whether the envelope that is set up is mirrored in the left or right channel. (This can be useful to generate a sense of movement without writing to the amplitude registers).

Bits 3,2,1-Set envelope shape. These three bits select an envelope shape from the table found in Fig 3. in the Philips sound chip data sheet.

Bit 4-Controls resolution of envelope if repeated very quickly. 16 levels of resolution controlling envelope modulation are available for repetition up to 977Hz and 8 levels for repetition rates above 977Hz.

Bit 5-This bit controls whether the envelope registers are controlled by generators 1 or 4. If this bit is reset(0), then the internal clock is used to control the shape of the envelope. If this bit is set(1), then the external clock is used(AO) to control the envelope.

Bit 6-is not used.

Bit 7-is used to enable or disable the operation of the envelope controller. Note that this bit should be set (1), the enable envelope control.

THE SAM COUPE SOFTWARE

MEMORY

Several memory arrangements are possible with the Coupe, with various advantages and disadvantages, and each of these sections is discussed in detail. The screen mode and the location of screen memory are perhaps the first factor to consider.

- SAM COUPE MEMORY MAP -

0FFFFH	ROM1 OR PROGRAM, VARIABLES OR SCREEN	
0C000H	PROGRAM, VARIABLES OR SCREEN	
0B000H	START OF CHANNELS	CHANS
05CB6H	SYSTEM VARIABLES	SVARS
05A00H	KEYBOARD TABLE	
058E0H		KTAB
	DEF KEY BUFFER	DKLIN DKBU
05800H	LINE INTERRUPT COLOUR TABLE	
05600H		LINICOLS
	PALETTE TABLE	
055D8H		PALTAB
	UDG PATTERNS CHR$ (127 - 168)	UDGS
05490E	CHARACTER PATTERNS CHR$(32 - 127)	
05190H		CHARS

```
05120H          PAGE ALLOCATION
                OF 33 BYTES.
                (1 per 16K PAGE+FFH)

05l00H                          ALLOCT

050FFH

                GENERAL-PURPOSE
                BUFFER SPACE

04F00H          MACHINE
                STACK           SP

                FLOATING POINT STKEND
                CALCULATOR
                STACK
04D00H                          STKBQT
04C00H          FARLDIR BUFFER

04B50H                          HDL
                TAPE HEADERS
04B00H                          HDR
                BASIC STACK FOR STKBQT
                GOSUB/DO/PROCED BASSTK
                URES
                                BSTKEND
                SPARE FOR HEAP
                OR BASIC STACK


                SYSTEM HEAP.    HEAPEND
                2.7K AVAILABLE
                DOESN'T PAGE
04000H                          HEAPST

                ROM 0

00000H
```

- SCREEN MEMORY LOCATION -

A complete screen takes 6.75K in mode 1 and 12K in mode 2 (not including a 2K gap between pixel and attribute data). In these modes the memory containing the screen can be switched in at any 16K boundary. The paging system means that the other 16K RAM page in the same half of memory will page at the same time as the page containing the screen. Modes 3 and 4 require 24K and thus two 16K pages. You can either leave the screen memory permanently paged in or just page it in when you want to read it or write to it, providing your screen handling code is paged in at the same time! The ROM switches in the screen as it requires, in sections C and D of the memory map (8000H).

- THE STACK -

The ROM normally keeps the machine stack and system variables in section B at 4000H-5CD0H, so that they are not paged out when the screen, Basic program or variables area are paged in in the top half of memory.

ROM

ROM0 is normally switched into section A of the memory map. ROM1 is normally switched out until it is needed; when in use it is located in section D. Jump table routines in ROM0 that need ROM1 will page it in automatically, use it, and then restore its previous status.

- USE OF ROM SUBROUTINES -

The ROMs usually expect the stack to be in section B, and many routines require the system variables in the same area to be intact. However, if you wish to use this area for your own code, you can page out the system variables and use the ROM routines indirectly via JSVIN (see later). This routine will switch the system variables back in temporarily while the ROM routine of your choice is executed. The screen will be temporarily switched into section C and D if required, but your code can usually be located in the same area. Naturally, ROM0 must be enabled if you want to use the jump table, but you could page it out most of the time.

For example, you could load your machine code, disable interrupts, switch out ROM0, switch new RAM into section A/B, and use it for your program. You could provide some code to handle maskable and/or non-maskable interrupts, if desired. Before using a ROM routine, enable ROM0 from somewhere outside section A, and use JSVIN to call the routine you want. If you switch the page your code is currently executing in, remember that the next instruction will come from the new page!

- USING SCREEN MODES 3 AND 4 -

The arrangement of screen memory in modes 3 and 4 makes plotting a pixel very simple. A method similar to that used by the ROM for mode 4 (256*192, 16 colour) is shown below. On entry, L and H are the X and *Y* coordinates (0,0 at top left) and D has the colour to use in both its nibbles (e.g. D=77H for colour 7). H and L are returned unchanged (convenient, for example, during line drawing).

```
PLOTLH:     SCF
            RR  H
            RR  L
            LD  A,(HL)
            JR  C,PLOTODD   ;jr if odd pixel

            XOR D
            AND 0FH
            XOR D           ;mix left nibble of D with right nibble
                            ;tread from screen
            LD  (HL),A
            ADD HL,HL       ;restore entry HL
            RET

PLOTODD:    XOR D
            AND 0F0H
            XOR D
            LD  (HL),A
            ADD HL,HL
            INC L           ;restore entry HL
            RET
```

The address of the next scan line can always be obtained by adding 128 to the current screen address.

The screen can be easily moved left or right by 1 pixel using the RLD and RRD instructions provided by the Z80. LDIR and LDDR can be used for 2, 4, 6 etc. pixel moves. The ROM subroutines support most possibilities.

- PALETTE SWITCHING -

The ROM interrupt routine sets up the palette registers (CLUT) at the start of every frame, using one of two tables of 16 bytes at PALTAB, and switching between them to give flashing colours. It is a good idea to keep these tables up-to-date with the current colours, even if you do not want to use the ROM routines, because the palette registers themselves cannot be read. Software entered via MNI would find an accurate PALTAB useful when doing screen dumps or palette alterations.

The ROM can also change a specified palette memory to a given value at a specific scan line, allowing many more than 16 colours to be displayed at once. If only a single change is made per scan line, it will occur during the scan flyback time and no flicker will occur in the display. In the Coupe's striped start-up screen all the stripes are actually palette colour 0 - but palette memory 0 is being altered.

If more than one palette change is programmed to occur at a given scan line, the later changes will occur as the picture is displayed, and this can cause flicker if the colour being altered is on-screen at that particular location. Up to 127 changes can be made per frame. If you wish to provide your own routine this can be added using the line interrupt vector address (LINIV), or by switching out the ROM.

- THE PAGING SYSTEM -

The paging system makes it possible to page in any address so that there are at least 16K bytes above it before FFFFH is passed. This means that when dealing with data structures less than 16K long, once the start of the data is paged in there is no need to take special precautions to ensure pointers do not "fall off" the end of memory. The ROM normally uses memory sections C and D as a “rotating window" onto memory and keeps system variables and the stack in section B. For example when the next Basic line is pointed to by adding a number to HL (adding current line length to current line start) code similar to that below checks to see if HL has moved out of section C into section D. If so, the page value is increased and HL is adjusted to point to the same actual byte, at an address 4000H lower, in section C. The whole line can then be dealt with fairly simply since no further checks of HL need be made until the next line is looked at.

```
      BIT 6,H
      JR Z,LAB1

      IN A,(HMPAGE)
      INC A
      OUT (HMPAGE)
      RES 6,H

LAB1: ETC.
```

Here there is no danger of the lower 5 bits of HMPAGE being incremented past 1FH to corrupt the upper bits, since the last line of program has a terminator. This will be the case with most other data structures you are likely to want to use.

The ROM should be used via the documented fixed locations or the jump table to avoid problems with new ROM issues. References are made to ROM 1.0 and ROM 1.2. All production machines contain ROM 1.0, however a free upgrade will allow users to change this to ROM 2.0, which is the production version of ROM 1.2. Production machines will be fitted with ROM 2.0 by mid 1990. The ROM version installed can be determined by: PRINT PEEK 15, and dividing the value displayed by 10.

FIXED ROUTINES NEAR THE START OF LOWER ROM

```
0000  MACHINE INITIALISATION
0004  POP HL      Call here to return your own address in HL.
0005  JP (HL)
```

The two instructions above provide the ability to (effectively) use CALL within relocatable code. For example.

```
            CALL 0004H              ;get address of HERE in HL
HERE        LD   DE,THERE-HERE
            ADD  HL,DE              ;add displacement to get
                                    ;address of THERE
            CALL 0005H              ;effectively, CALL THERE
            etc.

THERE       RET
```

The method above corrupts HL and DE, but it can be adapted to use the alternate register set or IX or lY if desired.

```
      0006  JP (JY)
      0008  ERROR HANDLER
      000F  ROM VERSION NUMBER
      0010  Print char in 'A' register to current stream. The 'K'
            and 'S' channels respond to control codes for PEN,
            PAPER, OVER, INVERSE, FLASH, delete, left, right, up,
            down, AT, TAB and carriage return.

      0013  Print BC bytes starting from DE, using current stream.
      0016  JP (BC)
      0018  Get current character from Basic line.
      0020  Get next character front Basic line.
```

0028 Call floating point calculator. The example multiplies the top two numbers on the floating point calculator stack, then divides the result by 10 and swops top and third no.s.

```
      RST 28H     ;a,b,c,d
      DB MULT     ;a,b,c*d
      DB STKTEN   ;a,b,c*d,10
      DB DIVN     ;a,b,c*d/10
      DB SWOPl3   ;c*d/10,b,a
      DB EXIT
```

The operation codes used are different from the Spectrum's, and more numerous; see the separate section on the floating-point calculator. Note: The vector address RST28V makes it possible to translate any operation code into another before the operation is performed.

```
002B        DJNZ 002B               Useful for uncontended timings.
002D        JP (IX)

0030        USER  RST.  Intended  for debugging etc.  JPs  to  the
            address in "RST3OV".
0033        JP (DE)
0035        DJNZ 0035H               Useful for timing.
0037        RET
0038        MODE 1 INTERRUPTs
005C        OUT (HMPR),A
            JP (HL)
005F        LD A,B                  Useful for timing.
            OR C
            DEC BC
            JR NZ,005F
            RET

0066        NON-MASKABLE INTERRUPTS
```

RAM contention means that ROM routines usually run slightly faster than identical RAM routines. If you are executing the Z80's block instructions on large blocks, and speed is important, it might be worthwhile to call the routines below:

```
008F        LDIR:   RET
0092        LDDR:   RET   (block moves run about 8% faster in ROM)
0095        CPIR:   RET
0098        CPDR:   RET
009B        OTIR:   RET
009E        OTDR:   RET
```

JUMP TABLE AT 0100H

Sometimes conditions for entry a few bytes past the start of a routine are given, as well as the usual entry conditions to the routine; where this is done, you can read the routine address from the jump table, add a displacement, and enter part-way through (for example using CALL HLJUMP). The documented entry conditions will remain constant.

When routines are described as "generating an error" this means that error handling (using RST 08H) will clear the machine stack back to the value held in ERRSP and then do a RET. Normally this leads to an error message being given in the lower screen window, but this can be avoided by PUSHing the address of your own error handler onto the stack, and altering ERRSP (possibly temporarily) to hold the current Stack Pointer. When you gain control, ERRNR will hold the error number (see the list in the Basic manual).

JSCRN (0100H) Select screen held in the C register (1-16). Equivalent to Basic's SCREEN command. An error is generated if the screen is not open. This Jump Table entry is not available in version 1.0 of the ROM

JSVIN (0103H) Call parameter word with system variables page switched in. HL', DE', BC' and F are corrupted, and interrupts are enabled, before the parameter address is called. All other registers are passed intact. On exit, all registers except AF' have the values left in them by the called routine. The original status of HMPR is restored. For example:

```
CALL JSVIN
DW JGETINT
LD (STORE),HL etc.
```

JHEAPROOM (0106H) Reserve BC bytes in the system heap. This is always paged in; it is in the same page as the stack and the system variables, paged into memory area B. On entry, BC=bytes to reserve (if BC is positive) or bytes to release (if BC is negative). On exit, if the carry flag is set the call was successful, and DE points to the old HEAP END (the start of the reserved space), HL points to the new HEAP END (one byte past the end of the reserved memory), and BC is unchanged. If the carry flag is reset, not enough space was available, and HL holds the amount that the request exceeded the available space by.

JWKROOM (0109H) Open BC bytes in workspace (a temporary work area, cleared before each Basic statement is dealt with). On exit, DE points to the reserved space, starting in section C of the memory map, paged in. HL points one byte past the end of the space, provided it is less than 16K long. A and BC are unchanged.

JMKRBIG (010CH) Open A 16K pages and BC bytes at HL in section C of the memory map. BC must be 0-3FFFH. This routine can be used to open space before the Basic program, inside it, in the variables area, or the workspace. If there is not sufficient space, an error will be generated. On exit, HL points to the space and DE points to the end of the space (if the space <16K).

JCALLBAS (010FH)
Call BASIC subroutine at line HL. When RETURN is used, or any error occurs (for example, STOP, variable not found, etc. - even "OK" counts, in this case), your machine code will be re-entered. On return, A holds an error code, or zero if there was no error (and the zero flag will be set if there was no error). When your machine code finally does a RET to return to Basic, the statement that called the code in the first place, such as CALL 54000 or PRINT USR 54000, will be completed, but then the program will carry on at the statement after the RETURN in the Basic subroutine, or after the point where an error occurred, because Basic's notion of its current line and statement have been altered during the subroutine execution.

To return to the statement after your CALL, or anywhere else in the program for that matter, read the values from SUBPPC and PPC before using any Basic subroutines, then INC the SUBPPC value and place it in NSPPC, and place the old PPC into NEWPPC. This will cause a GO TO of the specified line and statement when your machine code RETs.

JSETSTRM (0112H)
Set the stream specified by the A register (FBH-10H).

FBH=channel 'B' (Always)	Printer binary output.
FCH=channel '$' (Always)	Output to a string.
FDH=channel 'K' (Always)	Lower screen I/O.
FEH=channel 'S' (Always)	Upper screen I/O.
FFE=channel 'R' (Always)	Output to edit line.
00H=channel 'K' (Usually)	
01H=channel 'K' (Usually)	
02H=channel 'S' (Usually)	
03H=channel 'P' (Usually)	Printer text output.

JPOMSG (0115H)
Output message number A from table at DE. Zero gives the first message. Each message should have bit 7 set on the last character. Characters 0-31 should not be included. The current output stream is used.

JEXPT1NUM (0118H)
Syntax check/evaluate a numeric expression. During syntax checking an invisible 5-byte form of any literal numbers is inserted. During run time the result of the expression is left on the floating point calculator stack.

JEXPTSTR (0116H)
Syntax check/evaluate a string expression. See JEXPT1NUM.

JEXPTEXPR (0llEH)
Syntax check/evaluate an expression. See JEXPT1NUM. These three routines can be useful in extending the Basic interpreter.

JGETINT (0121H)
Unstack number from calculator stack into HL. BC holds a copy of HL, and A holds a copy of L. An error is generated if the rounded number is not in the range 0-65535.

JSTKFETCH (0124H) Unstack last value from calculator stack to AEDCB. If the value is a floating-point number, the 5 bytes are in CPC/Spectrum format. If the value is a whole number between 0 and 65535 (which may be negative) it may be in a special form where A=O (showing special form), E=SGN (0=positive, FFH=negative} and D and C are less and more significant bytes. If the value is a string, A holds the page the string text starts in, DE holds the start address of the text start within the page (8000-BFFF), and BC holds the string length.

JSTKSTORE (0127H) Stack AEDCB registers on the calculator stack. On exit, HL holds the new STKEND, and AEDCB are unchanged. No other registers are altered.

JSBUFFET (012AH) Unstack the details of a string from the calculator stack and copy the text to a buffer in the system page in section B of the memory map. Paging is undisturbed on exit, no matter where the string is copied from. Generates an error if the string is more than 255 characters long, or the null string. On exit, DE points to the start of the buffer and BC holds the string length. The A register holds a copy of C.

JFARLDIR (012DH) Copies bytes from page A, offset HL to page C, offset DE. Copies (PAGCOUNT) 16K pages and (MODCOUNT) bytes (MODCOUNT=0000-3FFF), paging as needed. The paging state on entry is unimportant, and is unchanged on exit. Since the routine copies via a buffer, it is usually possible to write a faster, more specialised routine if the required paging can be arranged.

JFARLDDR (0130H) A LDDR version of FARLDIR.

JPUT (0133H) — Screen modes 3 and 4 only. Place a block of data on the screen at given coordinates.

On entry, C holds the x coordinate (bit 0 is not significant, since the data can only be aligned to byte accuracy). The x-range is 0-255, whatever the MODE.

The B register holds the y coordinate, with 0 at the top of the screen and 191 at the bottom.

The HL register points to the data to use; the first and second bytes of the data specify width in bytes and height in pixels, followed by 811 bytes for the top scan of the block, all bytes of the second scan, etc.

The A register determines the method of placing the data on the screen. 0=overwrite with INVERSE option, 1=XOR, 2=OR, 3=AND, 4=overwrite (with no INVERSE option - faster), 5=mask. The mask option requires a second block of data the same length as the first (excluding the first 2 bytes of the first block) pointed to by HL', with bits set for "unmasked" and reset for "masked".

This allows a complex shape, perhaps with holes in it, to be placed on a background without border effects. If the shape hangs off the bottom of the screen, it will be "trimmed' to fit. If the data overlaps the right border of the screen, it will wrap round to the left, but 1 scan lower.

Both HL and HL' (if used) must point to data that will be available for use; since JPUT will temporarily switch in the current screen (using CUSCRNP, or at least the contents of that address) at 8OOOH, the data must be located at 4000-7FFFH, or EOOO-FFFFH in the screen page. JPUT also needs to use the system variable CURP (or a byte at that address) as a temporary store, and it will read INVERT to determine INVERSE status, where this is an option. JPUT should run rather faster than Basic's PUT, since Basic has to move data from the variables area.

Unlike attribute colour systems, the colours to be used are implicit in the block data, although methods 0-3 allow an INVERSE option. The restriction that blocks must be put down at even pixel boundaries is to keep the speed high. If you wish to place blocks at 1-pixel intervals, it is possible to prepare two versions of the same graphic, with the data displaced by 1 pixel in one of them, in order to get the required effect.

In version 1.0 ROMs there is an error in JPUT, but you can get the same effect by using: PUSH AF: CALL 3A7BH: POP AF: CALL 2DACH instead of CALL 0133H. If the ROM version number (PEEK 15) is not 10, you must use 0133H.

JGRAB (0136H) — Screen modes 3 and 4 only. Store a block of screen data from given coordinates to a buffer.

On entry, C holds the x coordinate (bit 0 is not significant) and the x-range is 0-255 whatever the screen MODE. B holds the y coordinate, with 0 at the top of the screen and 191 at the bottom. E holds the block width in bytes, and D holds the length in pixels. On exit, DE points to the start of the block in the current screen page (specified by CUSCRNP); the first byte is always zero, the next byte is the width in bytes, the next byte is the length in pixels, and the following bytes store the block, scan line after scan line. BC holds the block length, including the a header bytes. Since the area is stored in the "spare" 8K following a MODE 3/4 screen, the length of the data must be a little under 8K, at the most. JGRAB requires temporary storage at the variable CURP (or storage at that address) and it will read CUSCRNP in order to temporarily switch the screen in. If the GRABed area hangs off the bottom of the screen, it will be trimmed as needed.

JPLOT (0139H) — Plot pixel at x coordinate in C and y coordinate in B, except when the MODE is 3 and FATPIX 0 is in force (THFATT will be zero); in which case use HL as the x coordinate (0-511) and plot a "thin" pixel. The y-axis has 0 at the top and 191 at the bottom in all cases. There is no control to prevent you plotting beyond the bottom of the screen.

JDRAW (013CH) — Draw a line from the current position C (or HL if "thin" pixels are in Use, according to THFATT) pixels horizontally and B pixels vertically. D=1 for down (or for no y movement) and FF for up, E=1 for right (or for no x movement) and FF for left. The colour and OVER and INVERSE state come from the temporary graphics variables. The screen used comes from CUSCRNP. An error is generated if the line is drawn over the screen edge.

JDRAWTO (013FH) — Draw a line from current position to point C,B (or point HL, B if "thin" pixels are in use). The y axis has zero at the top and 191 at the bottom. Colours etc. come from the temporary graphics variables.

JCIRCLE (0142H) — Draw circle at C,B radius A. If "thin" pixels are set, HL should hold an offset from the left-hand screen border (in pixels). This allows a circle to be placed anywhere on a mode 3 screen even though the x coordinate is limited to 1 byte. The y axis has zero at the top. Colours etc. come from the temporary graphics variables.

JFILL (0145H)

Modes 3 and 4 only. Uses the 16*16 pixel pattern at DE to fill an area starting at coordinates C,B. The y axis has zero at the top. If DE holds zero, a "solid" fill with the current PEN (from M23INKT) is performed. Otherwise DE can have any value (but values 000l-3FFFH will use the ROM as a pattern!). If A=0 then a preliminary transfer of the screen memory to a scratchpad area (6K long, immediately after the 24K screen memory) is made; you will need to make A=0 for at least the first fill you use. Later fills on the same screen over the same colour can make A non-zero for greater speed.

If you are using MODE 3 with "thin" pixels, HL should hold a byte offset from the left-hand screen edge for the point to start filling at. This allows any part of the screen to be filled, despite the x coordinate being limited to one byte.

The pattern data pointed to by DE has 8 bytes for the top row, 8 bytes for the next, etc. so 128 bytes are needed. Each nibble controls colour for 1 pixel in mode 4 or 2 pixels in mode 3.

JBLITZ (0148H)

Execute a string of graphic commands BC long at DE. DE can be anything from 4000H to FFFFH, and BC anything from 0 to FFFFH (JBLITZ will *page* if required). PLOT, DRAW, DRAW TO, CIRCLE, OVER, INK, CLS and PAUSE commands can be included in the string. They are handled at a higher level than JPLOT, JDRAW and JCIRCLE; the floating point calculator and special Basic variables can be used for offset and scaling. The use of numerous system variables makes having the system variables paged in hard to avoid. x coordinates and moves of 0-255 only are allowed; i.e. only half the screen is easily usable in MODE 3 with FATPIX 0 selected. The coordinate system used is the same as that used by Basic; e.g. bytes 01 00 00 would act like PLOT 0,0 in Basic and be modified if XOS or YOS were altered. To perform a RELATIVE DRAW:

00 or FF give the SGN of an x movement (00=positive, FF=negative), followed by the x movement, followed by the SON of the y movement and the y movement. If movements are negative, they are stored as 256-move.

PLOT:	0l	followed by x,y
DRAW TO:	02	followed by x,y
CIRCLE:	03	followed by x,y,r
OVER:	04	followed by 0-3
PEN:	05	followed by 0-17
CLS:	OG	followed by 0-1
PAUSE:	07	followed by 0-255

The easiest way to prepare the data for JBLITZ is to use RECORD in Basic and then POKE the string where you need it.

JROLL (O14BH) Modes 3 and 4 only. Move part of screen left, right, up or down. The screen area must be an even number of pixels across. A horizontal move by 1 pixel or any even number of pixels is possible. A vertical move by any number of pixels is possible. Wrap-round is optional; it uses the area immediately after screen memory as a buffer. On entry, A is 00 if a SCROLL (no wrap) is wanted, or FFH for a ROLL (with wrap). B holds the pixels to move by, C is a direction code (l=left, 2=up, 3=right, 4=down), D holds length of block in pixels and E holds width of block in pixels (bit 0 is non-significant). HL holds the coordinates of the top left-hand corner of the block to be moved, L holding the x coordinate (range 0-255, whatever the MODE) and H holding the y coordinate with 0 at the top of the screen. JROLL checks the MODE system variable, copies the permanent graphics variables onto the temporary ones, and uses M23PAPT as the background colour for new screen generated by SCROLL. The routine uses CURP and TEMPPB3 in the system variables, and up to 0181H bytes at 4D00H to store "unrolled" code loops.

JCLSBL (Ol4EH) Clear entire screen if A=0, else clear upper screen.

JCLSLOWER (0151H) Clear lower screen. Also selects channel K (lower window input/output).

JPALET (0154H) If A=FFH, put colours BC in PALTAB as main and flashing colours for palette colour E. (Make B=C for no flashing). Colours should be 0-127, palette entry 0-15. If A is non-zero, it is a y-coordinate to alter the palette at. PALTAB is not altered; instead an entry is made in the line interrupt colour table.

JOPSCR (0157H) OPEN SCREEN number C in mode B. The pages required are located in the Page Allocation Table and reserved, and an entry is made in SCLIST. The screen number must be 2-16, and it must not be already open, or an error is generated.

JMODE (015AH) Set screen MODE that is in the A register (0-3 gives MODEs 1-4).

JTDUMP (Ol5DH) Do a text screen dump (if dump utility installed).

JGDUN? (0160H) Do a graphic screen dump (if dump utility installed). The system variables documented in the Basic manual control the area dumped and its magnification.

JRECLAIM (0163H) Reclaim (close up) BC bytes at HL (8000H-BFFFH). Can be used to close up memory before a Basic program, within it, or in the variables area or workspace. BC must be less than 16K; if you want to reclaim a bigger space, call one byte from the start of the routine, with A holding number of 16K pages to reclaim, in addition to the space specified in BC.

JKBFLUSH (0166H) Flush keyboard buffer.

JREADKEY (0169H) Read keyboard, flush butter (like INKEYS). Z/NC if no key pressed, else NZ/CY and A=key value (ASCII).

JWAITKEY (016CR) Read next key into A from keyboard buffer. Wait for a key to be pressed if needed.

JBEEP (0l6FH) Make a beep DE-1 cycles long at period HL*8 T states.

JSAVE (0172H) Save a block CDE bytes long starting at HL (switched into section C). C holds the number of 16K pages, and DE holds the length MOD 16K. Make A register first byte saved, (FFH for 'data block', 01 for SAM 'header', 00 for spectrum header). If SLDEVT (see system variables) holds "T", the SAVE will be to tape-if it holds "N" the SAVE will be to the Network. SLNUMT controls the save speed to tape (112 is standard speed).

JLOAD (0175H) If the carry flag is set, LOAD CDE bytes to HL (switched into section C of the memory map). If the carry flag is reset, do a VERIFY. The A register holds the type of data to deal with (FFH=data block, O1=SAM or Spectrum header, 00=spectrun header). If SLDEVT holds "T", tape will be used; if "N", the Network. Varying input tape speeds are handled automatically. The routine returns with carry set if the operation was successful; otherwise there has been an error.

JLDVD (0178H) If the carry flag is set, LOAD CDE bytes of data (not header) to HL. If the carry flag is reset, do a VERIFY. Use disk, tape or Net according to the contents of SLDEVT. An error is generated if the operation fails.

JEDGE2 (017BH) Waits for 2 signal transitions on EAR, returninq in A and C the number of 47-T state units taken. On entry, B should hold the current EAR status in bit 6, bit 3 should be set, and the other bits should reflect what you want sent to the BORDER port when there is a signal transition. Bits 2-0 will be reversed on every transition. The D' register should hold a value that determines the length of a pause between finding one transition and looking for another (this avoids double counts when the tape signal has buzzy edges). For standard speed tapes, D' should be about 46H, JLOAD sets D' according to tape speed. Interrupts should be disabled for accurate results.

The routine returns Z if it had to wait too long (256 units), NC if ESC was pressed, and CY if okay. Entry 2 bytes from the start avoids the initial zeroing of the C register which is used as a counter; entry 6 bytes from the start waits for only 1 signal transition on EAR. On exit the B register holds a new value suitable for another call to the routine, and A and C both hold the counter value. Note: In version 1.0 ROMs this jump table entry incorrectly has bit 15 of the address set. You should read the address at O17CH, reset bit 15, and CALL that address.

JSTRS (017EH) Create ASCII version of number on floating-point calculator stack in buffer at 5BA0H. On exit, DE holds 5BA0H and BC holds the number of characters in the buffer.

JSENDA (0181H) Send the byte in the A register to the parallel printer. The system variable at 5A10H is read to obtain the port to use, and the routine loops until the printer is ready before sending the byte and then returning. If ESC is pressed, an error will be generated. Uses A and BC only.

JUMP TABLE ENTRIES INTRODUCED AFTER ROM 1.0

JNCHAR (0184H) Call SCREEN$ subroutine. Try to match the character at line D, column E on the screen with any character in the main character set (pointed to by CHARS) or any character in the set pointed to by UDG. On exit, the carry flag is set if the search was successful, in which case A holds a character code of 20H-7FH (32-127) if the character was found in the main set, or 80H-7FH (128-168) if the character was found in UDG set. If the carry flag was reset, A=0 and the search failed.

JGRCOMP (0187H) Subroutine used by graphic dump utility.

JGTTOK (Ol8AH) Try to match potential spelled-out keyword pointed to by DE in a list of keywords at HL+1. The list should be in upper case, with bit 7 set on the last letter of each keyword. The A register should hold the number of words in the list, plus 1. On exit no match was found if the zero flag is set. If a the zero flag is reset, the A register holds the entry number in the list (1 or more), HL points to the start of the original word and DE points just past the end of it.

JCLSCR (018DH) Close screen number held in the C register.

- CHARACTER SET -

The character set from CHR$ 32 to CHR$ 130 is stored in compressed form in ROM1 and loaded into RAM in the system variables page when the machine is turned on. CHR$ 131-168 are initially defined as blanks, although CHR$ 128-143 are displayed as block graphics unless "BLOCKS 0" has been set. (The block graphic shapes are generated as needed - the character set itself does not contain them). Characters apart from block graphics occupy a 6*8 matrix, making then suitable for display at 6-pixel or 8-pixel intervals.

Character definitions occupy 8 bytes, from top row of pixels to bottom row. The left and right columns of pixels of this 8*8 block are not used when printing is at 6-pixel intervals. The character set base address is 5190H, and space is available for CHR$ 32 to 168 in one continuous block. An additional system variable, HUDG, can be set up to point to definitions for CHR$ 169-254, but standard designs for characters 128 to 168 are supplied (file 'font') which includes a full set of foreign characters which match the accented characters on most printers. Their use is recommended so that software can easily be converted for use abroad.

- VECTORS -

These allow important ROM routines to be intercepted by the user. They contain addresses which will be called if (and only if) the most significant byte is non-zero. This gives the option of RETurning (perhaps you decide not to handle a particular case), or discarding the return address and turning the vector into effectively a JUMP.

Several programmers might want to use the same vector. If your programs simply overwrite vectors they are likely to be incompatible with future utilities from MGT and other software suppliers. We recommend that your programs save the old vector value. Unfortunately, an automatic linking mechanism that made this very easy had to be omitted front the ROM, but the solution is still fairly simple: for example, to add code to provide some action for the STEP token:

```
;Initialise CMDV but store any existing entry first.
INIT        LD  HL,(CMDV)
            LD  (STORE),HL
            LD  HL,MYPATCH
            LD  (CMDV),HL
            RET
MYPATCH     CP  STEPTOKEN
            JR  Z,MYP2        ;JR if we are interested
            LD  HL,(STORE)
            INC H
            DEC H
            RET Z             ;RET if no address was in CMDV at INIT

            JP (HL)           ;let another routine have a go

MYP2        (Code to provide some action for STEP)
            RET
```

NB All examples are program fragments, not complete programs, and do not, for example, include the above code, which should be included in actual applications.

LIST OF VECTORS

DMPV (5ADAH) — Called if the DUMP keyword is used in run-time. CHAD points to 0DH or 3AH if just DUMP was used; otherwise DUMP CHR$ was used.

SETIYV (5ADCH) — SETTY is called when DRAW, PLOT and CIRCLE "decide" what pixel setting routine to use, and itself calls SETIYV if its most significant byte is non-zero. SETIY points the IY register to a suitable routine, and the provision of a vector is intended to allow extra plotting modes, such as dotted lines, to be implemented without all the graphics commands being rewritten. On entry to SETIYV, A holds the current screen mode-1.

On return to PLOT, DRAW and CIRCLE the IY register should point to a pixel setting routine, and only A and HL should be altered otherwise. In mode 4, and mode 3 when "fat" pixels are selected, the value of A on exit is passed to D' by the graphics commands, and can be used to hold the PEN colour if desired.

If you alter SETIYV, as with all the vectors, your routine making just RET will cause the normal ROM routine to be executed. Thus your code might start with a check of the mode to see if it is one you want the ROM to deal with, and RET if so. Otherwise, POP the return address so that the ROM routine is never used, and set IY yourself. The next RET will be to DRAW, PLOT or CIRCLE.

The pixel setting routine that you supply the address of in IY should plot the pixel at L,H and return with only AF changed. The coordinate system used has 0,0 at the top left-hand comer of the screen. You may want to take note of the current OVER, INVERSE and PEN settings. (In which case you may find it speeds things up if, like the ROM, you set IY to a different routine for some options). The routine should end in JP (IX), not RET.

PRTOKV (5ADEH) Print a token. A=token code to expand and print as ASCII.

MNIV (5AE0H) Non-maskable interrupt. Normally holds the address of a routine that returns to Basic when the BREAK button is pressed. NMIV switches in the system page temporarily, and provides its own stack there, whatever the paging state when the interrupt was generated. On entry, the LMPR status when the NNI occurred is stored in NMILRP, the Stack Pointer when NMI occurred is stored in NMISP, and the AF and HL registers have already been PUSHed onto the stack in use when NMI occurred (which may be paged out).

FRAMIV (5AE2H) Called by the frame interrupt 50 times a second. All main registers can be used.

LINIV (5AE4H) Called by the line interrupt, which is generated when the value in the line interrupt register matches the scan about to occur. This vector can be used to alter the palette, change screen or screen mode, etc. in mid-screen. To synchronise such changes with the scan flyback, it is possible to wait for a change at the HPEN port to show that the scan number has just changed. (This will fail if a light-pen is connected, however). All main registers can be used.

COMSV (5AE6H) Called by COMMS interrupt.

MIPV (5AE8H) Called by MIDI input interrupt. On entry, the A register has just been read from the MIDI IN port.

MOPV (5AEAH) Called by MIDI output interrupt.

EDITV (5AECH) Called before the editor is used for editing a line or INPUT.

RSTOBV (5AEEH} Called early in error handling, with A=error code. The alternate registers have been selected, CHAD and CHADP have been copied to XPTR and XPTRP, DE points to the error code that follows RST 08H, and an address 1 byte past that code is on the stack, underneath the return address to the error handling routine. No clearing of stacks has been done at this stage.

RST28V (5AF0H) Called by floating point calculator with A=next code in the list that follows a RST 28H instruction. DE points to the end of the floating point calculator stack. On exit, the IX register should be unchanged, and DE should point to the end of the floating point calculator stack. (This may be different from the entry value, if items have been added, discarded or combined). See example below.

RST30V (5AF2H) User RST. Jumped to (Not called like the other vectors) by RST 30H. (ROM0 uses RST 30H to provide a "CALL ROM1" facility, but this only applies to RST instructions used inside ROM0). All registers are passed intact.

CMDV (5AF4H) Called with the A register holding the code of the character about to be syntax checked or executed (normally a command code). Can be used to add commands, or modify the action of the existing ones.

EVALUW (5AF6H) Expression evaluator. Called with A=current character in expression; can be used to add new functions.

LPRTV (5AF8H) Called as each character code above 127 is handled by the printer driver, when channel "P" is being used.(More complete control can be obtained by altering the "P" or "B" channel output address - see below).

On entry to the routine the character code is in the register. For example to transform CHR $ 130 to "A":

```
INIT:    LD  HL,PATCH
         LD  (5AF8H),HL
         RET

PATCH:   CP  130
         RET NZ

         POP BC            ;JUNK RETURN ADDRESS

         LD  A,65
         RST 1OH           ;PRINT CHARACTER
         LD  HL,5A70H      ;PRINTER COLUMN VARIABLE
                           ;(SVAR &70)
         INC (HL)          ;1 COLUMN RIGHT
         RET
```

Instead of using RST l0H to print the new character(s), you might prefer to use either CALL 0181H (send A to parallel printer) or probably better, call the following code to send the A register to channel "B":

```
CHBOP:      LD  HL,(5C4FH)    ;CHANNELS BASE ADDRESS
            LD  DE,25         ;(use 15 to point to  "p"
                              ; channel)
            ADD HL,DE
            LD  E,(HL)
            INC HL
            LD  D,(HL)        ;DE=channel B output address
            EX  DE,HL
            JP (HL)
```

Channel "B" handles all output without fiddling with it; channel "p'. responds to control codes, adds auto-LF etc. and then sends its output to channel "B". something like ESC ;"R"; CHR$ 0 should be sent via the **"B"** channel. Channel "B" normally outputs using 0181H, but if you use CHBOP rather than O181H directly, it will be easier for others to divert printer output to another device, such as a serial interface. (By altering the output address in the channel).

MTOKV (5AFAH) Called if a potential spelled-out keyword is not recognised by the ROM. On entry, DE points to the potential keyword. On exit, conditions should match those of jump table entry JGTTOK.

MOUSV (5AFCH) Called every frame. when this vector is used, the ROM's own scan of the mouse is bypassed.

KURV (5AFEH) Called before the editing cursor is printed (by calling the address in MNOP). ROM1 is on. Could be used to supply a "transparent" or other cursor.

Additional example: Suppose you have a lot of machine code graphics subroutines which require the screen to be switched in at 4000H. It would be nice to modify USR so that any use of the subroutines switched in the required page temporarily; the routine below does this, for machine code located in the top half of memory. INIT is called to alter the floating point calculator ~ vector to point to USRPTCH; whenever the calculator handles a USR, we intercept it, juggle the pages to put the screen at 4000H (taking care with the stack) and call the USR address, stacking BC back on the floating point calculator stack as the result of USR, and returning DE pointing to STKEND and IX unchanged. (All as expected by the calculator).

USRPTCH must be in the system page to guarantee it is resident when the vector is called; the best place is in the system Heap. PART2 must be in the top half of memory, but in any case it is a goad idea to use a short routine in the Heap to call a large routine in another page, since this keeps the Heap train being filled up.

```
INIT        LD  HL,USRPTCH
            LD  (RST28V),HL
            RET

USRPTCH     CP  45H                 ;USR code
            RET NZ                  ;keep things simple and ignore
                                    ;anyone else who might want to use
                                    ;this vector

            POP BC                  ;junk RET address
            IN  A,(HMPR)
            PUSH AF                 ;save current HMPR status
            LD  A,1                 ;OK if PART2 in page 1 or 2
            OUT (HMPR),A      ;page in PART2

            IN  A,(LMPR)
            AND 0BFH
            OUT (LMPR),A            ;ROM1 off
            CALL PART2
            IN  A,(LMPR}
            OR  40H
            OUT (LMPR),A            ;ROM1 on again for RET to calculator
            POP AF
            OUT (HMPR),A            ;original HMPR status restored
            RET


PART2       LD  (STORE),SP          ;we will be switching out the stack,
                                    ;so save current SP
            LD  SP,TSTACK           ;put a new stack elsewhere (>=8000H)
            PUSH AF                 ;LMPR status
            PUSH IX                 ;calculator needs IX unchanged
            CALL JGETINT            ;get USR address in BC
            IN  A,(VPAGE)
            AND 1FH                 ;A=screen page
            DEC A
            OUT (LMPR),A            ;page it in at 4000H
            CALL BCJUMP             ;CALL USR address
            POP IX
            POP AF
            OUT (LMPR),A            ;original LNPR status
            LD  SP,(STORE)          ;original SP
            XOR A
            LD  E,A
            LD  D,C
            LD  C,B
            CALL JSTKSTORE
            EX  DE,HL
            RET

STORE       DW 0
```

FLOATING POINT CALCULATOR COMMAND CODES

The floating-point calculator performs all the mathematical operations of the Coupe, and many of the string operations. Since it provides a simple way of programming quite complex tasks in a sort of sub-language, it can be very convenient. The calculator is called using RST 28H, followed by a series of command codes (which may have parameters). A special code terminates the list. The data that the calculator manipulates is placed on a special stack that we will refer to as the FPCS. All items are five bytes long. The jump table allows registers to be passed to and from this stack, and the B register is passed into and out of the calculator (see below).

Numbers are held in two forms; the integer form stores whole numbers between -65535 and +65535 as zero, SGN (0 for positive, FFH for negative), less significant and more significant bytes, and a final zero byte. E.g. 80H is stored as 00 00 80 00 00, 4001H as 00 00 01 40 00, -1 as 00 EFF FF FF 00 and -80H as 00 FF 80 FF 00. (Negative values are stored in negated form (65536 minus the number).)

If numbers are in integer form the calculator is often able to use specialised, faster forms of operations such as multiply, subtraction and addition. Numbers outside the -65535 to +65535 have to be held in a different form, as 2 to the power of a number called the exponent, multiplied by another number called a mantissa. The first byte is the exponent +80H, and the other four bytes are the mantissa. This is always between 0.5 and just less than 1, which means that the first bit is always 1, allowing it to be actually used as a SGN bit (0 for positive, 1 for negative).

This scheme allows values between about 1E38 and 1.7E-39 to be stored to an accuracy of 9 or 10 digits. It is important to note that some fractional numbers (such as 1/10) cannot be stored exactly in this binary form (just as, for example 1/3 cannot be represented exactly in decimal form). This means that if you subtract 0.1 from 1 ten times, the result is not exactly zero. This is not a fault - just a limitation of number systems in general, which you should be aware of.

Strings are held as 16K page of string text start, offset of string text start (8000H-BFFFH, LSB/MSB) and string length (LSB/MSB).

In the list below, the action of many of the calculator codes is shown symbolically. Where a command code deals with two numbers (e.g. multiply) the last number on the FPCS is referred to as N2 and the number below that is referred to as N1. If the values dealt with are strings, $2 and $1 are used.

```
00H MULT        N1*N2
01H ADDN        Ni+N2
02H CONCAT      $1+$2 Prepare the concatenated text of $1+$2 in the
                Workspace, and stack the details of this temporary
                string.
03H SUBN        N1-N2
04H POWER       N1^N2
05H DIVN        Nl/N2
06H SWOP        Swap V1,V2
07H DROP        Discard last value
08H MOD         N1 MOD N2
09H IDIV        N1 DIV N2
0AH BOR         N1 BOR N2
0BH Reserved
0CH BAND        N1 BAND N2
0DH NUOR        N1 OR N2
0EH NUAND       N1 AND N2
0FH NOTE        N1<>N2 Comparisons like <> delete the two compared
                values from the FPCS and replace them with 1 or 0
                (true or false).
10H NLESE       N1<=N2
11h NGRTE       N1>=x2
12H NLESS       N1<N2
13H NEQUAL      N1=N2
14H NGRTR       N1>N2
15H SAND        $1 AND N2
16H SNOTE       $1<>$2
17H SLESE       $1<=$2
18H SGRTE       $1>=$2
19H SLESS       $1<$2
1AH SEQUAL      $1=$2
1BH SGRTR       $1>$2
1CH SWOp13      Swop last value and value 2 below.
1DH SWOp23      Swop second-to-last value and value below it.
```

The three command codes below, along with DECB, use a displacement byte to skip backwards or forwards in the list of commands (like GO TO in Basic, or JP in assembly language). The displacement is the distance between the displacement byte and where you want to jump to; if it is greater than 7FH it treated as a backwards jump, for example, FFH is backwards by 1.

```
1EH JPTRUE      Jump by displacement given in next byte, if the last
                value is one. (The last value is discarded).
1FH JFFALSE     Jump by displacement given in next byte, if the last
                value is zero. (The last value is discarded).
20H JUMP        Jump by displacement given in next byte.
```

The calculator has a special store called BREG that initially holds whatever the Z80's B register did when the calculator was called. The four command codes below use BREG.

```
21H LDBREG      Load BREG with next byte.
22H DECB        Decrement BREG and jump by the displacement given in the
                next byte if BREG is non-zero.
23H STKBREG     Put BREG on FPCS as a number between 0 and 255.
24H USEB        Take the next calculator code from BREG.
```

25H DUP Duplicate top FPCS entry.
26H ONELIT Stack next byte on FPCS (as a number between 0 and 255).
27H FIVELIT Stack next 5 bytes on FPCS (as any number).
28H SOMELIT Stack number of bytes specified by next byte, using (next byte+1} as the first byte. (This allows many 5-byte values to be stacked at once).
29H LKADDRB Use the next 2 bytes as an address; stack the byte found at that address on the FPCS.
2AH LKADDRW Use the next 2 bytes as an address; stack the word (2 bytes) found at that address on the FPCS.
2BH REDARG Fiddle with the argument of SIN.
2CH LESS0 Stack 1 (TRUE) or 0 (FALSE) according to whether the last value is <0. The last value is discarded.
2DH LESE0 Stack 1 or 0 according to whether the last value is <=0. The last value is discarded.
2EH GRTR0 stack 1 or 0 according to whether the last value is >0. The last value is discarded.
2FH GRTE0 Stack 1 or 0 according to whether the last value is >=0. The last value is discarded.
30H TRUNC Chop off the digits after the decimal point in the last value.
31H RESTACK Transform the last value to full 5-byte form if it isn't already in that form.
32H POWR2 Calculate 2 to the power of the last value.
33H EXIT Finish using floating point calculator.
34H EXIT2 Finish using floating point calculator, and do a RET.
35H Reserved
36H Reserved
37H Reserved
38H Reserved

The command codes below are used to provide most of Basic's functions:

39H SIN	4CH EOF
3AH COS	4DH PTR
3BH TAN	4EH Reserved
3CH ASN	4FH UDG
3DH ACS	50H NUMBER
3EH ATN	51H LEN
3FH LOGN	52H CODE
40H EXP	53H VAL$
41H ABS	54H VAL
42H SQN	55H TRUNC$
43H SOR	56H CHR$
44H INT	57H STR$
45H USR	58H BIN$
46H IN	59H HEX$
47H PEEK	5AH USR$
48H DPEEK	5BH INKEY$ #
49H DVAR	5CH NOT
4AH SVAR	5DH NEGATE
4BH BUTTON	

The calculator has six 5-byte memories for holding temporary results. The following codes store and recall the memories:

```
C8H STOD0       Copy the last value to calculator memory 0-5,
C9H STOD1       then delete it from the FPCS.
CAH STOD2
CBH STOD3
CCH STOD4
CDH STOD5

D0H STO0        As above, but do not delete the last value
D1H STOl        after copying it.
D2H STO2
D3H STO3
D4H STO4
D5H STO5

DBH RCL0        Stack the value in calculator memory 0-5 on.
D9H RCLl        the FPCS
DAH RCL2
DBH RCL3
DCH RCL4
DDH RCL5
```

The codes below provide a compact method of stacking comanonly used constants on the FPCS:

```
E0H STKHALF           Stack 0.5
ElH STKZERO           Stack zero
E2H STX1SK            Stack 16384
EGH STKFONE           Stack 5-byte form of 1.
E9H STKONE            Stack integer form of 1.
ECH STKTEN            Stack 10.
FOE STKRALFPI         Stack PI/2
```

TAPE SYSTEM

There are five types of tape files; Basic program, Machine Code, Numeric array, String array and SCREEN$. File saving occurs at a selectable speed which can be set to match Spectrum tape speeds. The header format is different from the Spectrum's and will not be recognised by a Spectrum as a header. The data block format is the same as on the Spectrum, but it may contain up to 512K. SCREEN$ files normally contain the palette settings (40 bytes) and line interrupt table, (1 to 509 bytes), so even a screen that switches the palette part-way down a screen can be reloaded. The screen mode is set appropriately before loading of the data block begins.

Files are loaded at the correct speed automatically. Spectrum CODE file headers are converted into the SAM equivalent and then used to load the data block. Spectrum CODE files can be loaded into the Coupe as either CODE or SCREEN$ files by specifying this after the LOAD command. (The Coupe distinguishes SCREEN$ files from CODE files because they have to be loaded to screen memory, which can be anywhere in the memory map). Coupe CODE files can be forced to load as SCREEN$ files, and vice versa, simply by using LOAD "name" CODE or SCREEN$. CODE files can have an execution address.

Altering the system variable TPROMPTS (SVAR 50) can turn off the prompt before SAVE, and/or the printing of file as names they are read from tape.

- FILE HEADER FORMAT -

Tape, disk and Net use a common header format as far as the Basic interpreter is concerned. (The DOS holds information in a slightly different format for its own use, but uses information from, and passes information to, the standard header buffers). SAVE uses a single buffer, HDR (Header-requested) at 4B00H where the details of the file to be saved are built up before the header and then the data are saved. LOAD also uses this buffer, plus an extra one, HDL (Header-loaded) at 4850H which is used to load file headers for comparison with HDR (On the Coupe a "tape header reader" program just consists of a program to PEEK the HDL buffer). (NB There is an extra file type in SAMDOS - .SNP).

HEADER BUFFER FORMAT (HDR and HDL):

0	- (1)	Type. 5 is a Spectrum 48K snapshot, 16 is a Basic program, 17 is a Numeric array, 18 is a String or string array, 19 is a CODE file, and 20 is a SCREEN$ file.
1-10	- (10)	File name, padded with spaces.
11-14	- (4)	Allows longer file name if DEVICE is not T, for example 'D2:filenamexx'. SAMDOS will strip off any device identifier, so the maximum length of a filename is still ten characters.

15 - (1) Flags. Bit 0 is set if the name is not to be printed on LOADing, bit 1 is set if the CODE is protected. The flags can be conveniently set during SAVE by:

```
SAVE CHR$ 1+"name" - sets bit 0
SAVE CHR$ 2+"name" - sets bit 1
SAVE CHR$ 3+"name" - sets both bits
```

(Characters <=3 as the first character in a file name just set the flags - they are not saved as part of the name). Flag bit 0 works in addition to the TPROMPTS system variable. Bit 1 protects CODE files; when this bit is set, an auto-running CODE file cannot be stopped by MERGE, or re-directed to another address by supplying one in, for example:
LOAD start,length,newaddress.

16-26 - (11) If type is 17 or 18, holds type/length byte and array name
16 - (1) If type is 20, holds screen MODE.
16-18 - (3) If type is 16, holds program length excluding variables.
19-21 - (3) If type is 16, holds program length plus numeric variables.
22-24 - (3) If type is 16, holds program length plus numeric variables and gap length before string/array variables.

(The extra data for Basic programs allows NVARS, NUMEND and SAVARS to be set up on LOADing).

27-30 - (5) Reserved
31 - (1) 16K page that file starts in (ignored on LOADing, except for CODE files). bits 4-0 hold the page, bits 7-5 are undefined. If type is 17 or 18 and the array being LOADed does not exist in memory, this byte in HDR will be FFH.
32-33 - (2) Offset within page of file start (ignored on LOADIng, except for CODE files). The offset is 8000H-BFFFH, in LSB/MSB form.
34 - (1) File length in 16K pages (in HDL, or HDR during SAVE) or requested start for CODE files in HDR during LOAD (FFH if there was no requested start). During LOAD of an array, "file length" in HDR is the length of the existing array in memory, or FFFFFF if there isn't one.
35-36 - (2) File length MOD 16K, plus 8000H (reset bit 15 of the length word to get the true length).

37	(1)	If CODE file, page of execute address, or FF if there isn't one. If Basic program, 00 if there is an auto-run line number in the next two bytes, or FF if there isn't one.
38-39	- (2)	If CODE file, offset within page of execute address (8000-BFFFH, LSB/MSB) if there is one. If Basic program, auto-run line number if there is one.
40-79	- (40)	Comment field. Not initialised to anything, but SAVEd and LOADed with the program. Could be POKEd or PEEKed to carry extra information about the file. (DOS only SAVEs the first 8 bytes in this area).

SAM BASIC

Numeric variable names must start with a letter. Numbers, letters, underlines and spaces may follow the first character, to a total of up to 32 characters (spaces do not count). (FOR-NEXT variables are treated like normal numeric variables).

String and array variable names are similar but limited to 10 characters (spaces do not count). String and string array names are followed by '$'. strings can be up to 65520 characters long. Arrays can fill all available memory, although no subscript can be over 64K. Program lines can have up to 127 statements, and can be up to 16127 bytes long.

Keywords are normally typed in full, although keys can be programmed to generate complete words. When a keyword is recognised, it is converted to capital letters in the listing A keyword which is followed by a letter will not be recognised, so printx is assumed to be a procedure name, but print x and print1 become PRINT x and PRINT 1.

If you press the EDIT key on its own, the line with '>' will appear in the editing area. If you type a line number and then press EDIT, the desired line will appear for editing. The cursor can be moved left, right, up or down in the edited line, using the cursor keys. The delete key deletes to the left; shift-delete deletes right.

BASIC MACHINE CODE INTERFACE

NB angle brackets enclose optional items.

SAVE "name" CODE start,length<,execute address>

Produces auto-running file if an execute address is used.

LOAD "name" CODE <start><,length><,execute address>

Executes file from execute address if used. Otherwise any execute address saved with the file is used. MERGE will stop CODE a file executing unless it is protected by setting a bit in the file header.

Start, length and execute values, and the addresses used by USR, USR$, CALL, POKE, PEEK etc. can be up to 528K minus 1.

USR n

Calls address n. Enters with BC=n. All registers can be used. USR returns the final value of BC.

USR$ n

As USR, but returns a string BC bytes long starting at DE in page A. (Page is irrelevant if DE<8000H).

CALL n

Calls address n with a list of numeric or string parameters if desired. The values of numbers and the lengths and starts of strings are placed on the floating point calculator stack, with information on type. The A register contains the number of parameters when address n is entered. Using the FPC stack allows numbers to be removed and rounded to 2 bytes with range checking by calling JGETINT in the jump table; floating-point numbers, or strings, can be unstacked using JSTKFETCH. strings shorter than 256 bytes can be copied to a buffer in the system variables area by calling JSBUFFET. (strings can be copied from anywhere in memory, even if they are paged out). JSTKSTORE can be used to place any result on the calculator stack.

CLEAR n	Set upper limit of Basic's RAM use.
OPEN n	Reserve n more 16K pages for Basic to use.
OPEN TO n	Allocate a total of n pages to Basic.
CLOSE n	Free n 16K pages.

LENGTH

This function returns the address and dimensions of arrays and strings, and the address of numbers. Useful for passing data via Basic's CALL.

```
POKE n,x          POKE 1 byte
POKE n,1,2,3      POKE multiple bytes (up to 32)
DPOKE n,x         POKE 2 bytes (LSB/MSB)
POKE n,a$         POKE entire string

PEEK n            PEEK 1 byte
DPEEK n           PEEK 2 bytes

XEM$(n TO m)      PEEK a string, for example:
```

LET a$=MEN$(70000 TO 71000) assigns 1000 bytes to a$ POKE 72000,a$ puts those bytes to 72000.

MEM$S can read any RAM page (16K-528K-l) but will not read the ROM's. Use PEEK or DPEEK if you need to read R0)

Examples of base conversion:

```
BIN 1010=10             HEXS 74="4A"            &FF=255
BINs 128="10000000"     HEX$ 16384="4000"       &10003=65539
                        HEX$ 65539="010003"
VAL("&"+a$)=255 if a$="ff"
```

MAJOR POINTERS TO BASIC'S MEMORY AREA

(ALL TAKE 3 BYTES, IN THE FORM: PAGE/OFFSET)

```
PROC                        START OF PROGRAM
NVARS                       START OF NUMERIC VARIABLES
NUMEND                      END OF NUMERIC VARIABLES GAP (<=512 BYTES)
SAVARS                      START OF STRING/ARRAY VARIABLES
ELINE                       START OF EDIT LINE
WORKSP                      START OF WORKSPACE
WKEND                       END of WORKSPACE
                            SPARE SPACF
RAMTOP                      LAST BYTE ALLOCATED TO BASIC PROGRAM,
```

See the section on system variables for full information.

- PAGE ALLOCATION TABLE -

The table at 5100H-5120H is very important; it holds a byte for every possible l6K RAM page in the Coupe, plus an FFH terminator byte. The first byte corresponds to the first 16K RAM page, the second to the second page, etc. Non-existent pages are also marked by FFH. Unused pages are marked as 00H. Pages used by the first Basic program in the machine are marked 40H; screen pages are marked C0H, and pages used by DOS (usually just one) are marked 60H.
On a 256K Coupe, the table (ALLOCT) will look like this after DOS has loaded:

40 40 40 40 00 00 00 00 00 00 00 00 00 60 C0 C0 FF FF FF etc.

A "Utilities" page is marked 20H. It is divided into 16 1K sections which can be used by assorted short utility programs. The final 16 bytes in a utilities page (SLOTT) show which 'slots" are reserved - a 0 shows the corresponding slot is free, and FFH that it is reserved. (The last slot is 16 bytes short of 1K). The proper method of allocating space for a short program is to look backwards through ALLOCT for 20H. If you find it switch in the indicated page and look backwards through SLOTT for a spare slot. Mark it and use that slot. (Or slots - you could reserve several slots). If you do not find a 20H entry in ALLOCT, look for 00H; report an error if none is found, else mark it 20H, clear the last 16 bytes of that page with zeros, and then reserve yourself some space in the new SLOTT you have just created.

Free pages can be used as temporary workspaces, provided you are sure that nothing is going to overwrite the page while you are using it. (Interrupts do not do this, but the DOS might).

Any keyboard key in NORMAL, CAPS SHIFTED, SYM SHIFTED or CONTROL states can return any code (0-255). The 69 keys and 4 shift states give 276 key in a keyboard 'map'. This map has a seemingly haphazard arrangement that corresponds to the keyboard matrix see Appendix C for diagrams of the keyboard and the map positions, plus initial codes returned by each position. E.g. map position 0 has 98 in it - so CHR$ 98 ('b') is produced when the actual key corresponding to that map position is pressed, without any shifts. The entire map can be altered if desired.

KEY Posn,x

KEY 23,134 makes key position 23 in the key map produce CHR$ 134 hen pressed.
KEY 0,65 would make the normal 'b' into CHR$ 65 ('A').
Posn must be 0-275, x must be 0-255.

When certain codes come from the keyboard, a search is made to see if an expanded definition has been made for them. Codes 192-254 can be defined in this way. If a definition does not exist, CHR$ 192-254 will be used, otherwise the expanded definition is used. This allows, for example, function keys to be defined as RUN (enter) or LIST (enter). See below.

DEF KEYCODE

DEF KEYCODE 195: PRINT 123: PRINT "z" will cause any subsequent code of 195 corning from the keyboard to be expanded to: PRINT 123: PRINT "z"(ENTER). This makes: 123 z appear on the upper screen area. The entire line after DEF KEYCODE 195: is used. If a final colon is added to the line, the automatic (ENTER) is suppressed, for example:

```
DEF KEYCODE 193: PRINT "asd":
would produce:
PRINT "asd" in the editing area.
```

DEF KEYCODE can also be used with a string, rather than the rest of a line, for example:
DEF KEYCODE 192,"TESTING" will cause any subsequent code of 192 coming from the keyboard to be expanded to '[t]TESTING" (ENTER). (This would cause an error report).
DEF KEYCODE 192,"TESTING:" would suppress the (ENTER) because the last character is a colon.

A particular definition can be cleared by:
DEF KEYCODE 195: (followed by nothing)
or DEF KEYCODE 195,""
DEF KEYCODE ERASE removes all definitions.

'Too many definitions' is reported if space runs out in the memory area used by DEF KEYCODE.

CLOSE
OPEN

Like the Spectrum, streams 0 and 1 are normally OPEN to channel 'K'; PRINT #0;"test": PAUSE will print at the bottom of the screen. PRINT #2 will print on the upper part of the screen, as usual (using channel 'S'), and PRINT #3 acts like LPRINT (using channel 'P'). You can also do something like OPEN *#;"P": PRINT #6;"testing" to send data to the printer via stream 6, or OPEN #2;"P" so that the normal screen output goes to the printer instead. INPUT normally uses channel "K" (the lower screen), but INPUT#2 is allowed and uses the upper screen, (channel "S").

For example: INPUT #2;"Value:";v will allow you to type in the value of V on the upper screen, next to the 'Value:' prompt. The line is NOT cleared after you press enter, unlike the lower screen.

The RECORD command works via stream 16 (which did not exist on the Spectrum). Stream 16 prints to the string variable specified by RECORD, and it will still do so after RECORD STOP. This night be useful to some users: ~or example, RECORD TO a$: DIR #16;1 :RECORD STOP, will place a disk catalogue in a$. Other streams apart from 16 can be opened to the string variable using, for example, OPEN #5;"$".

OPEN #s;"b" (or "B") will allow stream S to send anything to the printer - this is useful for sending control codes. (The problem with the 'p' stream is that control codes are acted on, and may not be sent to the printer - the TAB control code for example, will send a number of spaces to the printer). Channel 'p' actually uses channel 'b' for output, after it has modified the data it receives. Therefore all an advanced user need do to divert output to, say, a serial printer driver, is to alter the channel 'b' output address.

SAVE, LOAD, MERGE, VERIFY

Allow programs, strings, arrays, machine code and screens images to be handled.

SAVE "name" CODE start,length,execute-address is allowed, so that
LOAD "name" CODE will start running from the address given.
MERGE "name" CODE will suppress auto-running of code files.

LOAD "name" DATA b$() works, but also, DATA addr$() or DATA abc$ are allowed. Note that SAVE "name" DATA blitS is okay even if blit$ is a simple string, such as a BLITZ string.

LOAD "name" LINE n loads the program and goes to line number n. This overrides any auto-run line number saved with the program.

SAVE "name" SCREEN$ saves the screen image and the current mode and palette information for later reloading. (LOAD automatically switches modes if required).

DEVICE

This new command makes SAVE, LOAD, MERGE and VERIFY work with the disk or Network.

DEVICE d means use disk drive 1.
DEVICE dl ditto
DEVICE Dl ditto
DEVICE N5 use Network station 5.
DEVICE T use tape, SAVE speed similar to the spectrum's.
DEVICE t35 use tape, SAVE speed 35.
(Speed 112 is spectrum speed, 35 is much faster. Faster speeds are less reliable - but 3*spectrum speed should be feasible with many recorders. LOAD automatically adjusts to the tape speed that was used when the file was saved).

```
SOUND r,d
SOUND r,d;r,d;r,d...
```

sends data byte D to sound chip register R. Up to 127 pairs of numbers can be used.

This explanation is based on the program in the SAM Coupe User's Guide. The program is reproduced here for ease of reference

```
10 REM sound effects
20 GO SUB 3000
30 REM steam locomotive
40 LET L=7 : GO SUB 2000
50 LET p=20 : GO SUB 1000
100 REM flying saucer
110 LET L=3: GO SUB 2000
120 LET p=20 : GO SUB 1000
200 REM sonic scooter
210 SOUND 17,4 : SOUND 16,64
220 LET p=5 : GO SUB 1000 :
PAUSE 200
300 REM telephone
310 GO SUB 3000
320 FOR n=0 TO 3
340 SOUND 2,255 : PAUSE 18
350 SOUND 2,0 : PAUSE 8
360 SOUND 2,255 : PAUSE 18
370 SOUND 2,0 : PAUSE 80
380 NEXT n
400 REM cuckoo
410 BEEP 0.2,19   BEEP 0,4,15
420 STOP
1000 REM stereo Left to Right
1010 LET a=0 : LET b=0
1020 LET L=1 : LET R=0 :
GO SUB 1500
1030 LET L=0 : LET R=16 :
GO SUB 1500
1040 LET L=-1 : LET R=0 :
GO SUB 1500
1050 LET L=0 : LET R=-16 :
GO SUB 1500
1060 RETURN
1500 REM output subroutine
1510 SOUND 2,(a+b) : PAUSE p
1520 FOR n=1 TO 15
1530 LET a=a+L : LET b=b+R
1540 SOUND 2,(a+b)
1550 PAUSE p : NEXT n
1560 RETURN
2000 REM output sound data
2010 DATA 9,64,16,16,17,0,24,
138,28,1,21,4,20,0
2020 DATA 9,255,17,1,20,4
2030 DATA 24,142,20,4,10,111,
17,6,9,128,16,32,28,1
2040 FOR n=1 TO L
2050 READ a,d : SOUND a,d :
NEXT n
2060 RETURN
3000 REM clear the sound
3010 FOR n=0 TO 31 : SOUND n,0 :
NEXT n
3020 RETURN
```

Line 20 clears the sound chip's registers by setting them all to zero.

Line 40. L refers to the pairs of data on line 2010. The data pairs refer to setting up the various registers.

Line 50. p refers to the speed at which the sound is moved across the stereo field. If you play the sound demo program, you will find that the sonic scooter moves from left to right faster than either the steam locomotive or the flying saucer. The subroutine at line 1000 tells the computer where to place the sound across the stereo field. The subroutine at line 1500, outputs the sound on both left and right channels.

Both the flying saucer and sonic scooter are similar to the steam locomotive except that the steam locomotive uses white noise.

The sonic scooter changes the current octave, so that register 17's octave is 4 and register 16's octaves are set to 0 for generator 8, and 4 for generator 9. Notice also that the speed of movement of the scooter is faster than either the locomotive or the flying saucer.
The telephone is placed in the centre of the stereo field after clearing the sound chip again. This sound uses envelope register 24 and places a sound from tone register 10 through it using enable mixer 20 with bit 4 set. It also uses tone register 9 to control the speed of the envelope. Finally, register 28 enables the sound. The 'bleep' sound is controlled by switching the volume of amplitude register 2 between 0 and 255. This is repeated 4 times with pauses between each of the rings.

The cuckoo uses the BEEP command. This command, as on the Spectrum, does not use the sound chip at all, but the ASIC to generate the sound.

The subroutine at line 1000 sets up control data for the output subroutine at line 1500. Variables L and R set up which direction to go when 'ramping' the volume up or down on either channel, whist the subroutine at line 1500 does the ramping itself. Register 2 is the amplitude register which controls the sound. It is also here, on line 1550 that P is used to control the speed of the movement across the stereo field.

The subroutine at line 2000 holds the register data, and actually acts upon the data presented to the chip. Its function is to output any information to the various registers as it receives the data.

The subroutine at line 3000 clears the sound chip by setting all registers to zero.

XOS, XRG,
YOS, YRG

Control graphics scale and origin. XOS and YOS are offsets added to x and y coordinates when graphics commands are used. They allow the origin to be moved from the normal position just above the editing area, on the left. RUN or CLEAR resets XOS and YOS to zero; LET alters them. XRG and YRG alter the scale of the graphics system. XRG is normally 256, or 512 if 'thin' pixels are in use. YRG is normally 192. Altering XRG to, say, 1024 would make PLOT 950,100 a legal command by doing 256/1024*950 before plotting the pixel on the screen. (Altering the ranges by powers of 2 gives faster plotting).

Normally, modes 2 and 4, and mode 3 if 'fat' pixels are used, have a graphics coordinate system with 0,0 just above the bottom editing section, at the left, and 255, 173 at the top right.

If thin' pixels are in use, the normal X is expanded so that top right is 511,173. In mode 1, the default character height is 8 pixels, rather than 9, and 2 less scans are used by the editing area, so that the top of the screen is at a Y coordinate of 175.

The bottom 2 lines of the display can be plotted on using:

```
      PLOT 0,-10.
```

If the position of the origin is moved by LET XOS=-18 then 0,0 is at the very bottom left of the screen and 0,191 is the top left.

KEYWORD CODES

The Coupe's functions are stored in the program as FFH (255) followed by 3BH-83H.

Function	Dec	Hex	Function	Dec	Hex	Function	Dec	Hex
PI	59	3B	COS	84	54	VAL$	109	6D
RND	60	3C	TAN	85	55	VAL	110	6E
POINT	61	3D	ASN	86	56	TRUNC$	111	6F
FREE	62	3E	ACS	87	57	CHR$	112	70
LENGTH	63	3F	ATN	88	58	STRS	113	71
ITEM	64	40	LN	89	59	BIN$	114	72
ATTR	65	41	EXP	90	5A	HEX$	115	73
FN	66	42	ABS	91	5B	USR$	116	74
BIN	67	43	SGN	92	5C	Reserved	117	75
XMOUSE	68	44	SQR	93	5D	NOT	118	76
YHOUSE	69	45	INT	94	5E	Reserved	119	77
XPEN	70	46	USR	95	5F	Reserved	120	78
YPEN	71	47	IN	96	60	Reserved	121	79
RAMTOP	72	48	PEEK	97	61	MOD	122	7A
Reserved	73	49	LPEEK	98	62	DIV	123	7B
INSTR	74	4A	DVAR	99	63	BOR	124	7C
INKEY$	75	4B	SVAR	100	64	Reserved	125	7D
SCREEN$	76	4C	BUTTON	101	65	BAND	126	7E
MEM$	77	4D	EOF	102	66	OR	127	7F
Reserved	78	4E	PTR	103	67	AND	128	80
PATH$	79	4F	Reserved	104	68	<>	129	81
STRING$	80	50	UDG	105	69	<=	130	82
Reserved	81	51	Reserved	106	6A	>=	131	83
Reserved	82	52	LEN	107	6B	Reserved	132	84
SIN	83	53	CODE	108	6C			

Codes from 85H to FEH do not need a preceding FFH; they are single-byte compressed forms of keywords. Context allows then to appear as keywords or as UDGs, as appropriate. For example, they will appear as keywords in a listing, unless inside quotes, but will appear as UDGs when PRINTed.

The tokens from 85H to 8FH are qualifiers - they have no result or action on their own, but simply modify the action at other commands. (This does not prevent an action routine being assigned to them. The DOS, for example, provides an action routine for WRITE).

Token	Dec	Hex	Token	Dec	Hex	Token	Dec	Hex
USING	133	85	OFF	137	89	THEN	141	8D
WRITE	134	86	WHILE	138	8A	TO	142	8E
AT	135	87	UNTIL	139	8B	STEP	143	8
TAB	136	88	LINE	140	8C			

The tokens from 90H-FFH are commands. The command address table (see System variables - CMDADDRT) contains a 2-byte address for each of them, starting with DIR. This is the address that handles the syntax checking and run-time action for the command.

Command	Dec	Hex	Command	Dec	Hex	Command	Dec	Hex
DIR	144	90	REM	183	B7	ON	222	DE
FORMAT	145	91	READ	184	B8	GET	223	DF
ERASE	146	92	DATA	185	B9	OUT	224	EO
MOVE	147	93	RESTORE	186	BA	POKE	225	El
SAVE	148	94	PRINT	187	BB	DPOKE	226	E2
LOAD	149	95	LPRINT	l88	BC	RENAME	227	E3
MERGE	150	96	LIST	189	BD	CALL	228	E4
VERIFY	151	97	LLIST	190	BE	ROLL	229	E5
OPEN	152	98	DUMP	191	BF	SCROLL	230	E6
CLOSE	153	99	FOR	192	C0	SCREEN	231	E7
CIRCLE	154	9A	NEXT	193	C1	DISPLAY	232	E8
PLOT	155	9B	PAUSE	194	C2	BOOT	233	E9
LET	156	9C	DRAW	195	C3	LABEL	234	EA
BLITZ	157	9D	DEFAULT	196	C4	FILL	235	EB
BORDER	158	9E	DIM	197	C5	WINDOW	236	EC
CLS	159	9F	INPUT	198	C6	AUTO	237	ED
PALETTE	160	AO	RANDOMIZE	199	C7	POP	238	EE
PEN	161	Al	DEF FN	200	C8	RECORD	239	EF
PAPER	162	A2	DEF KEYCODE	201	C9	DEVICE	240	F0
FLASH	163	A3	DEF PROC	202	CA	PROTECT	241	F1
BRIGHT	164	A4	END PROC	203	CB	HIDE	242	F2
INVERSE	165	A5	RENUM	204	CC	ZAP	243	F3
OVER	166	A6	DELETE	205	CD	POW	244	F4
FATPIX	167	A7	REF	206	CE	BOOM	245	F5
CSIZE	168	A8	COPY	207	CF	ZOOM	246	F6
BLOCKS	169	A9	Reserved	208	D0	Reserved	247	F7
MODE	170	AA	KEYIN	209	Dl	Reserved	248	F8
GRAB	171	AH	LOCAL	210	D2	Reserved	249	F9
PUT	172	AC	LOOP IF	211	D3	Reserved	250	FA
BEEP	173	AD	DO	212	D4	Reserved	251	FB
SOUND	174	AE	LOOP	213	D5	Reserved	252	FC
NEW	175	AF	EXIT IF	214	D6	Reserved	253	FD
RUN	176	B0	long IF	215	D7	Reserved	254	FE
STOP	177	B1	short IF	216	D8	Not usable	255	FF
CONTINUE	178	B2	long ELSE	217	D9	(function prefix)		
CLEAR	179	B3	short ELSE	218	DA			
GO TO	180	B4	END IF	219	DB			
GO SUB	181	B5	KEY	220	DC			
RETURN	182	B6	ON ERROR	221	DD			

Certain new keywords will be implemented in a disk-based extended Basic, and these words should not be used in Basic programs as procedure or variable names, or subsequent editing of the program will be inconvenient (although ALTER is a search-and-replace, so you could change every use of, for example, procedure SORT to SORTER, which would be okay). A provisional list of new keywords follows:

SORT, ALTER, USING$, SHIFT$, INARRAY, NUMBER, CHAR$, JOIN.

SYSTEM VARIABLES

Note: The function SVAR N gives the result 5AOOH+N.

LNCUR	5A00H	Cursor character for current line (usually '>').
LCCUR	5A01H	Cursor character when caps lock is off (usually CHR$ 128).
UCCUR	5A02H	Cursor character when caps lock is on (usually CHR$ 129).
BIN1DIG	5A03H	Character used by BIN$ as "1" (usually "1").
BIN0DIG	5A04H	Character used by BIN$ as "0" (usually "0").
INSTHASH	5A05H	Character used by INSTR as "match anything" character (usually CHR$ 35 "#"
SLDEV	5A06H	Current device letter (usually "T" on a tape system, but can be "D", or "N").
SELNUM	5A07H	Current tape save speed, or default drive number when a a disk drive is in use.
SPEEDINK	5A08H	Time between swaps of alternate PENs, in 50ths.of a second.
LINIPTR	5A09H	(2) Current position in line interupt palette change table.
XCMDP	5A0BH	(3) Page/offset of first external command list, or FFxxxxH.
PRRHS	5A0EH	Printer right-hand column limit. (Usually 79, to give 80 columns; use 255 to give no limit)
AFTERCR	5A0FH	Character code sent to the printer after CHR$ 13 if channel "p" is in use. (Normally 10, to give automatic line teed. Use a value of 0 if no character is to be sent).
LPTPRT1	5A10H	(2) Printer control port/0lH strobe value. The port is usually 233, with port 232 (one less) used for data. The second byte should always be 1.

The following system variables are only used when the screen dump utility has been loaded:

DMPLEN	5A12H	Graphic dump length (in 8-pixel units; normally 22)
DMPWID	5A13H	Graphic dump width (in 8-fat pixel units; normally 32)
DMPWM	5A14H	Graphic dump width multiplier (1 for normal width, 2 or 3 for double or triple width).
DMPHM	5A15H	graphic dump height multiplier (1 for normal height, not 1 for double height).
GCM1	5A17H	Initial message sent to printers before a dump. The first byte is the number of characters to send. The normal values are 6,27,108,8,27,51, 24,0,0 (2 bytes are spare).
GCM2	5A1FH	Message sent to printers before each row is dumped. The first byte is the number of characters to send. The normal values are 5,13,10,27,42,4,0,0 (2 bytes are spare).

GCM3	5A27H	Final message sent to printers after a dump. The first byte is the number of characters to send. The normal values are 4,13,10,27,64,0 (1 byte is spare).
DMPTL	5A2DH	(2) Address in screen of top left corner for graphic dumps. (Usually 8000H).

(End of system variables used by dump utility).

TABVAR	5A2FH	Zero for the print comma to tab by 16 columns (normal) or anything else for an 8-column tab.
M23LSC	5A30H	(2) Mode 3/4 lower screen colours. The first byte is PAPER*16+PAPER, the second is PEN*16+PEN, i.e. both nibbles in each byte match.
SOFE	5A32H	Flag for screen off enable/disable. If it is zero (normal) the screen will go blank in modes 3 and 4 when the keyboard has not been used for 22 minutes. Use a 1 to disable.
TPROMPTS	5A33H	Bit 0=1 to suppress printing of file names during tape loading; bit 1=1 to suppress prompts during tape saving.

This is the start of a block of variables saved with a screen when it is no longer the current screen.

BGFLG	5A34H	Block graphics flag. Zero if BLOCKS 1, 1 if BLOCKS0.
FL6OR8	5A35H	Zero if MODE 3 is using 6-pixel wide characters, otherwise non-zero.
CSIZEH	5A36H	Character height set by CSIZE command.
CSIZEW	5A37H	Character width set by CSIZE command.
UWRHS	5A38H	Upper window right-hand column. starts at 31.
UWLHS	5A39E	Upper window left-hand column. Starts at 0.
UWTOP	5A3AH	Upper window top row. starts at 0.
UWBOT	5A3BH	Upper window bottom row. starts at 18.
LWRHS	5A3CH	Lower window right-hand column. Starts at 31.
LWLHS	5A3DH	Lower window left-hand column. Starts at 0.
LWTOP	5A3EH	Lower window top row. Starts at 19.
LWBOT	5A3FH	Lower window bottom row. Starts at 20.
MODE	5A40H	MODE of current screen. 0-3 for modes 1-4.
YCOORD	5A41H	Current graphics position y coordinate, with 0 at the top of the screen and 191 at the bottom.
XCOORD	5A42H	(2) current graphics position x coordinate, with 0 at the left. The range is 0-255 unless "thin" pixels are in use (in which case the second byte is used and the range is 0-511).

The following are the "permanent" graphics/print variables, set by, for example, PEN 2, PAPER 4, OVER 1.

THFATP	5A44H	Zero if "thin" pixels in use, non-zero for "fat" pixels.
ATTRP	5A45H	Attributes used by modes 1 and 2.

MASKP	5A46H	Mask used by modes 1 and 2. Bits which are 1 make the corresponding attribute bit be taken from the screen, not ATTRP.
PFLAGP	5A47H	Bit 4 is set for paper 9, bit 6 for pen 9
M23PAPP	5A48H	Mode 3/4 PAPER. Nibbles (mode **4)** or double bits (mode 3) match unless striped paper is desired.
M23INKP	5A49H	Mode 3/4 PEN. Nibbles (mode 4) or double bits (mode 3) match unless a striped pen is desired.
OVERP	5A4AH	OVER status for printing. 0 for OVER 0, 1 for OVER 1 (XOR).
INVERP	5A4BH	0 for INVERSE 0, 255 for INVERSE 1.
GOVERP	5A4CH	OVER status for graphics. 0-3 for normal, XOR, OR, AND.

Temporary graphics/print variables, set by, for example:

```
      PRINT PEN 3; PAPER 5;
```

They are temporary versions of the variables listed above.

THFATT	5A4DH	copied *from* thfatp when Mode=2, else set to not zero (fat)
ATTRT	5A4EH	
MASKT	5A4FH	
PFLAGT	5A50H	
M23PAPT	5A51H	
M23INKT	5A52H	
OVERT	5353H	
INVERT	5A54H	
GOVERT	5A55H	

Current window variables, copied from UWRHS etc. or LWRHS etc. according to which window is in use.

WINDRHS	5A56H	
WINDLRS	5A57H	
WXNDTOP	5A58H	
WINDBOT	5A59H	

The next three variables are at 5AB1H, 5B70H and 5B71H in version 1.0 ROMs.

WINDMAX	5A5AH	(2) Upper window lowest bottom row/maximum right column. Used to limit the WINDOW command. Reset by the MODE command.
ORGOFF	5A5CH	Graphics origin offset from the screen bottom, in pixels.
LSOFF	5A5DH	Size of gap between upper and lower windows, in scan lines MOD CSIZEH.
Reserved	5A5EH	(14)
SPOSNU	5A6CH	(2) Upper window position as column/row.
SPOSNL	5A6EH	(2) Lower window position as column/row.

(End of block saved with a deselected screen).

PRPOSN	5A70H	Current printer column.
Reserved	5371H	
OPCHAR	5A72H	Used by LPRINT - character being printed.

DEVICE	5A73H	0=upper window, 1=lower window, 2=printer, 3=other
CLET	5A74H	Current channel letter K/S/P/B/T/$ etc.
IFTYPE	5A75H	Long/short IF status.
REFFLG	5A76H	Zero if a REF variable is being worked on.
CURDISP	5A77H	Current display, set by the DISPLAY command.
CUSCRNP	5A78H	Current screen page. Bit 7=0, bits 6 and 5=MODE (0-3) and bits 4-0=page number. Set by SCREEN command.
CURP	5A79H	Current upper RAM port. (Temporary store during paging).
CLRP	5A7AH	Current lower RAM port. (Temporary store during paging).
CSA	5A7BH	(2) Current statement address. Used by DOS.
FIRST	5A7DH	(2)
LAST	5A7FH	(2) Line numbers in, for example, LIST 20 TO 100.

The following variables point to the main sections of Basic's memory area. They are adjusted when space is made or reclaimed, as needed. In each set of three bytes, the first byte is the RAM page number (0-31) and the other two bytes are the offset in the page, plus 8000H. (In some circumstances the offset is allowed to be greater than 16K, but it is always less than 32K).

SAVARSP	5A81H	
SAVARS	5A82H	(2) String and array variables start
NUMENDP	5A84H	
NUMEND	5A85H	(2) Numeric variables end
NVARSP	5A87H	
NVARS	5A88H	(2) Numeric variables start
DATADDP	5A8AH	
DATADD	5A8BH	(2) Data address used by READ command.
WKENDP	5A8DH	
WKEND	5A8EH	(2) End of workspace. (Last used byte before RAMTOP).
WORKSPP	5A90H	
WORKSP	5A9lH	(2) workspace start.
ELINEP	5A93H	
ELINE	5A94H	(2) Edit line start.
CHADP	5A95H	
CHAD	5A97H	(2) Current character address.
KCURP	5A99H	
KCUR	5A9AH	(2) Address of cursor in the edit line.
NXTLINEP	5A9CH	
NXTLINE	5A9DH	(2) Address of next line in Basic program.
PROGP	5A9FH	

PROG	5AA0H	(2) Program start (address of line number of first line).
XPTRP	5AA2H	
XPTR	5AA3H	(2) Address in the edit line of a syntax error.
DESTP	5AA5H	
DEST	5AA6H	(2) Used in variable assignments.
PRPTRP	5AA8H	
PRPTR	5AA9H	(2) Address of current procedure call.
DPPTRP	5AABH	
DPPTR	5AACH	(2) Address of current DEF PROC statement.
CLAPG	5AAEH	
CIA	5AAFH	(2) Address of start of current line.
Reserved	5AB1H	
STRNO	5AB2H	Current stream number. used by DOS.
LDCO	5AB3H	Offset when loading spectrum machine code (16K pages).
Reserved	5AB4H	
OPSTORE	5AB5H	(2) Temporary store for print output address.
DMPFLG	5AB7H	If non-zero, printed characters are "thrown away".
LISTFLG	5AB8H	0/1/2 for LIST FORMAT 0/1/2.
LSTFT	5AB9H	Temporary version of LISTFLG used by channel 'R'.
INQUFG	5ABAH	"In quotes" flag. Bit 0=1 if character being printed is inside quotes. LIST zeros this bit so initial state is "outside" and tokens are printed, except inside quotes. PRINT sets it to 1 so UDGs are printed instead of tokens.
SPROMPT	5ABBH	If non-Zero no "scroll?" prompts are given.

The next six variables are used by the indented listing routine.

OLDSPCS	5ABCH	Indent status of previous line.
INDOPFG	5ABDE	Flag.
NXTSPCS	5ABEH	
CURSPCS	5ABFH	
NXTHSPCS	5AC0H	
CURTHSPCS	5AC1H	
KPOS	5AC2H	(2) Screen position of editing cursor.
SOFFCT	5AC4H	Counter used by auto-screen off. Changes every 256 frames if the keyboard is not in use.
SOFLG	5AC5H	Flag for screen has been turned off" (if non-zero).
SPEEDIC	5AC6H	Counter for flashing inks.
PALFLAG	5AC7H	Bit 0 shows which palette table in use, main or alternate.

Temporary stores:

TEMPW1	53C8H	(2)
TEMPW2	5ACAH	(2)
TEMPW3	5ACCH	(2)
TEMPB1	5ACEH	
TEMPB2	5ACFH	
TEMPB3	5AD0H	

LASTSTAT	5AD1H	Status port value on last interrupt.
SPSTORE	5AD2H	(2) stack pointer store used by maskable interrupts.
JVSP	5AD5H	(2) JSVIN stack pointer store.
NMISP	5AD7H	(2) Non-maskable interrupt stack pointer store.
NMILRP	5AD9H	(1) LMPR value when NMI occurred.

Vectors (see detailed description elsewhere).

DMPV	5ADAH	(2) Screen dump.
SETIYV	5ADCH	(2) Set IY register to pixel output routine.
PRTOKV	5ADEH	(2) Print a token.
NMIV	5AE0H	(2) Non-maskable interrupt.
FRAMIV	5AE2H	(2) Frame interrupt.
LINIV	5AE4H	(2) Line interrupt.
COMSV	5AE6H	(2) Comms interrupt.
MIPV	5AE8H	(2) MIDI input interrupt.
MOPV	5AEAH	(2) MIDI output interrupt.
EDITV	5AECH	(2) Editor.
RST8V	5AEEH	(2) RST 08H (error handler).
RST28V	5AF0H	(2) RST 28H (floating point calculator).
RST30V	5AF2H	(2) RST 30H (user RST).
CMDV	5AF4H	(2) Command.
EVALUV	5AF6H	(2) Expression evaluator.
LPRTV	5AF8H	(2) Printer.
MTOFCV	5AFAH	(2) Match token.
MOUSV	5AFCH	(2) Mouse.
KURV	5AFEH	(2) Edit cursor.

Tables used by MODE 3 and 4 to expand character pattern data (as stored in UDGs, for example), to a form suitable for placing on the screen. In MODE 3 each table holds 16 bytes (expanded versions of each possible nibble) while in MODE 4 each table holds 16 words (expanded versions of each nibble).

CEXTAB	5B00H	(32) Coloured expansion table - current PEN, PAPER and INVERSE state are used to modify this data.
EXTAB	5B20H	(32) Uncoloured expansion table. E.g. in MODE 3 or 4, the nibble 1010b is expanded to 11001100b or 1111000011110000b.
COMPFLG	5B40H	Flag bits used by LABEL/FN/PROC compiler.
BREARDI	5B41H	Non-zero if ESC between statements is to be disabled.
ERRSTAT	5B42H	Statement to go to ON ERROR.

ERRLN	5B43H	(2) line to go to ON ERROR.
ONERRFLG	5B45H	Bit 7=temporary ON ERROR active flag, bit 0 is permanent.
ONSTORE	5B46H	ON command's statement number.
BCSTORE	5B47H	(2) Used by RST 30H to hold BC register.
M3PAPP	5B49H	(2) used to bold MODE 4 PAPER and PEN when MODE 3 is used
M3LSC	5B4BH	(2) used to hold MODE 4 lower window colours when MODE 3 is in use.
TEMPW4	5B4DH	(2) Used by ADJUST POINTERS routine.
TEMPW5	5B4FH	(2) used by ADJUST POINTERS routine.
Reserved	5B51H	

Table used by the editor showing which screen lines have line numbers.

LPT	5B52H	(30)
Reserved	5B70H	(2)
RNSTKE	5B72H	(2) Rename stack pointer used by procedures with REF.
CURCHD	5B74H	Current Basic command.
LTDFF	5B75H	LET/DEFAULT flag.
STRMX6NN	5B76H	(11) Encoded type/length and name of the variable that stream 16 prints to. Set by RECORD TO command.
GRARF	5B81H	Graphics record flag (O=off).
DHADJ	5B82H	Double height adjust. Zero unless bottom of double-height character is being printed.
PACCOUNT	5B83H	Page counter used by FARLDIR and FARLDDR.
MODCOUNT	5B84H	(2) MOD 16K counter used by FARLDIR and FARLDDR.
ECREG	5B86H	(2) calculator's BC register.
AUTOFLO	5B88H	Zero if AUTO is off, else 1.
AUTOSTEP	5B89H	(2) AUTO command's step value.
LSPTR	5B8EH	(2) Line scan pointer used by the editor.
LNPTR	5B8DH	(1) Used by the editor.
MSEDP	5B8EH	(8) Used by the mouse as a data store.
BUTSTAT	5B8FH	Mouse button status.
MXCRD	5B96H	(2) Mouse x coordinate.
MYORD	5B98H	(2) Mouse y coordinate.

Used by the PRINT A NUMBER routine:

FRACLIM	5B9AH	
NPRPOS	5B9BH	(2)
DIGITS	5B9DR	
EPOWER	5B9EH	
DECPNTED	5B9FH	
PRNBUFF	5BA0H	(16) ASCII form of number.
BCDBUFF	5BBOH	(5)
OTHER	5BB5H	Net destination station number.
Reserved	5BB6H	
SLDEVT	5BB7H	Temporary device letter (usually copied from SLDEV).

SLNUMT	5BB8H	Temporary device number (usually copied from SLNUM).
OVERF	5BB9H	"SAVE OVER" flag used by DOS. Zero if SAVE OVER, else non-zero.
INSLV	BBBAH	(2) Block move vector.
STRLOCN	5BBCH	(2) Used by LOOK FOR A STRING/ARRAY VARIABLE routine.
TVDATA	5BBEH	(2) Used in handling control codes and their parameters.
DOSER	5BC0H	(2) If address is non-zero, DOS jumps there on exit.
DOSFLG	5BC2H	Zero if no DOS loaded, else page number containing DOS.
DOSCNT	5BC3H	Bit 0 is set if DOS is in control of the system.
BSTKEND	5BC4H	(2) End of Basic's stack (used by DO, GOSUB, procedures).
BASSTK	5BC4H	(2) Start of Basic's stack.
HEAPEND	5BC6H	(2) End of system heap.
HPST	5BC7H	(2) Start of system heap.
FPSBOT	5BCAH	(2) Start of floating point calculator stack.
DKDEF	5BDCH	(2) Start of KEYCODE definitions.
DKLIM	5BD0H	(2) Limit address of KEYCODE buffer.
PATOUT	5BD2H	(2) Address of 'printable characters' output routine.
ERRMSGS	5BD4H	(2) Start of error message table.
UMSGS	5BD6H	(2) Start of utility message table.
KBTAB	5BD8H	(2) start of keyboard table.
CMDADDRT	5BDAH	(2) Start of command address table.
MNOP	5BDCH	(2) Address of main output routine.
MNIP	5BDEH	(2) Address of main input routine.
PAGER	5BE0H	(14) Reserved for paging subroutine.
KBUFF	5BEEH	(18) Keyboard scan table - 2 tables of 72 bits each.
LEM1	5C00H	Used by keyscan
LASTH	5C01H	Last key hit. Zero if no key. Stops changing if keys are not being read, when buffer fills. RES 5,(FLAGS} is seen as a read. Used by keyscan.
KDATA	5C02H	Used by keyscan.
LKPB	5C03H	(2)
REPCT	5C05H	Used by keyscan.
LASTKV	5C06H	(2) shift and key codes from last key press.
KBHEAD	5C08H	Key from buffer queue head. Keeps last key value. Needs periodic resets of bit 5,(FLAGS) or buffer fills.
REPDEL	5C09H	Delay before keys auto-repeat (in 50ths. of a second); normally 33.
REPSPD	5C0AH	Delay between key repeats (in 50ths. of a second); normally 3.
Reserved	5C0BH	

STREAMS	5C0CH	(42) For streams -5 to 15, a word gives the displacement from the start of the channels area to the assigned channel. If the word is zero, the stream is closed. Stream 16 is dealt with as stream -4.
CHARS	5C36H	(2) Address 256 bytes below start of main character set.
ERRSOUND	5C38H	Length of error sound in 50ths. of a second; normally 60.
CLICK	5C39H	Length of keyboard click (normally zero).
ERRNR	5C3AH	Error number.
FLAGS	503BH	Main flags byte.
DFLAG	503CH	Display flags.
ERRSP	5C3DH	(2) SP value to use when an error occurs.
LISTSP	5C3FH	(2) SP value to use when an automatic list fills the screen.
Reserved	5C41H	
NEWPPC	5C42H	(2) New line to jump to.
NSPPC	5C44H	New statement to jump to, or FFH.
PPC	5C45H	(2) Current line number during program execution.
SUBPPC	5C47H	Current statement number.
BORDCR	5C48H	Attributes for lower screen in MODEs 1 and 2.
EPPC	5C49H	(2) Number of line with > cursor.
BORDCOL	5C4BH	Value to send to border port.
CHANS	5C4FH	(2) start of channels area.
CURCHL	5C51H	(2) start of current channel.
DEFADDP	5C53H	DEF FN address (page).
DEFADD	5C54H	(2) DEF FN address (offset)
Reserved	5C56H	(15)
STKEND	SC65H	(2) End of floating point calculator stack.
KPFLG	5C67H	Function keys if even, number pad if odd.
MEN	5C68H	(2) Start of calculator's memory area.
KLFLAG	SC6AH	8 if caps lock is on, else zero.
SDTOP	5C6CH	Line number of top line in an automatic listing.
COPPC	SC6EH	Line number that CONTINTUE goes to.
COSPPC	SC70H	statement number that CONTINUE goes to.
FLAGE	SC71H	Flags used by INPUT command and the editor.
STRIL	5C72H	Used when variables are assigned to.
SEED	5C76H	Random number seed. Set by RANDOMIZE.
FRAMES	5C78H	(3) Frames since machine was switched on (LSB first).
UDG	SC7BH	(2) Address of CHR$ 144.
HUDG	SC7DH	(2) Address of CHR$ 169 (initially undefined).
FRAMES34	5C7FH	(2) 2 more bytes of FRAMES counter.
OLDPOS	5C82H	Used by editor in clearing lower window.
SCRCT	5C8CH	Counter used to give "scroll?" prompt.
KBQB	5C8DH	(8) Keyboard queue. At 5A5AH in vl.0 ROM.
KBQP	5C95H	(2) Keyboard queue end/head pointers. At 5A6AH in vl.O ROM.
Reserved	5C97H	(6)
SCPTR	5C9DH	(2) Address in SCLIST at current screen's entry.
FISCRNP	5C9FH	Page of screen 1.

SCLIST	5CA0H	(16) Screens list. MODE/page of screens 1-16, or FFH if screen is closed. Bits 6 and 5=MOflE, 4-O=page.
LASTPAGE	5CB0H	last page reserved by Basic. set by OPEN, OPEN TO or CLOSE.
RAMTOPP	5CB1H	Page of RAMTOP.
RAMTOP	5CB2H	(2) Offset of RAMTOP.
PRAMTP	5CB4H	Last 16K page physically present in machine.

VARIABLE FORMATS

The Coupe has five main variable types:

Numeric variables
FOR-NEXT variables
Strings
String arrays
Numeric arrays

Numeric variables and FOR-NEXT variables are kept in the area pointed to by NVARS and NVARSP. This area begins with 26 2-byte entries, which hold the offset to the first variable starting with a given letter (from A-Z - hence 26 entries). If the offset is FFFFH, there are no variables starting with this letter. For each numeric variable, the start of the variable has this form:

Type/name length byte, offset LSB, MSB, character1, character2...

The type/length byte stores the name length-1 in bits 4-0 (so names can be up to 32 characters long}. Bit 5 is set if the variable is no longer in use, bit 6 is set for a FOR-NEXT variable and bit 7 is set if the variable is "hidden" (this is used by procedures to implement LOCAL variables). The offset is the distance to move from the MSB of the current offset to the type/length byte of the next variable starting with the same letter, or FFFFH if there are no more such variables. There may be from zero to 31 characters after the offset, forced to lower case if they are letters, and excluding spaces.

After this come five bytes to hold the value of the variable. (See the section on the floating point calculator for information on the makeup of these 5 bytes). If it is an ordinary numeric variable, there is just a value, but a FOR-NEXT variable has an additional 14 bytes; the complete format is:

VALUE (5): LIMIT (5): STEP (5): LOOP ADDRESS (3): STATEMENT (1)

The loop address is the page and offset (8000-BFFFH) of the start of the line to loop back to, and the statement is the number of the statement within that line to loop back to (loops to the first statement are thus the fastest).

Strings and arrays are kept in the area pointed to by SAVARS and SAVARSP. There is a buffer zone between the numeric and string/array variables, so that creation of a new numeric variable does not usually require the strings and arrays to be moved.

Variables in this area are in the order of their creation (by DIM or LET; LET a$=aS will move aS to the end of the area, unless a$ is an array). For each variable, the format is as follows:

Type/length byte, character1....character10, length (3)

The type/length byte holds the name length (1-10) in bits 4-0. Bit 6 is set for string arrays, bit 5 is set for numeric arrays, and both bits are reset for ordinary strings. Bit 7 is set if the variable is "hidden". Ten bytes of space follow to hold the variable name; unused bytes are undefined. The length is given as number of 16K pages, then length MOD 16K (LSB/MSB), for the rest of the array plus the 3 length bytes. For strings, the text of the string follows; for arrays, the next byte is the number of dimensions, followed by the length of dimension 1, dimension 2, etc. (2 bytes for each dimension). The array data then follows, all data with a first subscript of 1 being first, then the data with a first subscript of 2, etc. FFH terminates the list of string and array variables.

- FORMAT OF A BASIC PROGRAM -

The first line of a program is pointed to by PROG and PROGP (offset 8000-BFFFH and page). Each line has the format:

LINE NUMBER (MSB/LSB): LINE LENGTH (LSB/MSB): TEXT: ODH

Note that the line number is unusual in being stored most significant byte first. The line length is the length of the text and 0DH terminator. The line text contains compressed keywords (see keyword code table) and invisible forms of all literal numbers (decimal, binary and hexadecimal). The invisible forms are OEH followed by 5 bytes to store the value. The line may also contain control codes and their parameters.

The final line in the program is followed by FFH (so the maximum line number allowed is FEFFH).

SAM DISIC OPERATING SYSTEM

INTRODUCTION

SAMDOS has been designed specifically for the SAM Coupe computer. It is similar to G+DOS as used with the MGT Plus D spectrum disk interface.

- DISK DRIVE -

The internal SAM disk drive is a Citizen 3.5" slimline drive. Each drive is cased and fitted with the MGT disk controller interface, which utilises a VL-1772-02 floppy disk controller. By default the disks are formatted as double sided, 80 track per side, 10 sectors per track, to the IBM 3740 standard.

- DISK FORMAT -

We use 80 tracks per side, giving 160 tracks per disk. A track is made up of 10 data sectors, each giving 512 bytes of storage. The first 4 tracks of the disk are given up to the SAMDOS directory, leaving 156 tracks available for storage. This leaves available 1560 data sectors of 512 bytes (798720 bytes)

Although each data sector can hold 512 bytes, only 510 bytes of them are available for storage. The last two bytes of the data sector are used by the DOS to locate the next part of the file stored. Byte 511 holds the next track used by the file, while byte 512 holds the next sector.

DISK FILE HEADER

At the beginning of each disk file there is a file header. The file header is 9 bytes long:

	SAMDOS type	Plus D type
0	File type	File type
1-2	Modulo length	Length of file
3-4	Offset Start	start Address
5-6	Unused	
7	Number of pages	
8	starting page number	

Details of the Plus D header can be found in the technical information for the Plus D.

- FILE TYPE -

Each file type in the SAMOOS is allocated a numeric identifier:

5	- ZX Snapshot file	SNP 48k
16	- SAM BASIC program	BAS
17	- Numeric array	D ARRAY
18	- String array	$ ARRAY
19	- Code file	C
20	- Screen file	SCREEN$

- MODULO LENGTH & NUMBER OF PAGES -

In the SAMDOS header the length of the file is calculated by multiplying the number of pages (byte 7) by 16384 and adding the modulo length (word 1-2), LSB/MSE, i.e. the length MOD 16K.

- OFFSET START & STARTING PAGE NUMBER -

Read starting page number (byte 8): AND this with 1FH to get the page number in the range 0 to 31. To find the start multiply the page number by 16384, add the offset, and subtract 4000H (since the ROM occupies 0-3FFFH).

When SAMDOS is paged in it resides at 4000H, and ROM0 is placed at 0-3FFFH.

- SAMDOS DIRECTORY -

The first 4 tracks of the disk are allocated to the disk directory, starting at track 0, sector 1. These 4 tracks give us 40 sectors each split into two 256 bytes entries. Each of these entries will identify one file, thus allowing up to 80 entries in the directory.

The format of each directory entry is as follows:

(The User Information File Area CUIFA) will be described later).

Byte	UIFA	Description
0	0	STATUS/FILE TYPE. This byte is allocated one of the file types listed previously, but is also used as a file status. If the byte is 0 then the file has been erased. If the file is HIDDEN then bit 7 is set. If the file is PROTECTED then bit 6 is set.
1-10	1-10	FILENAME. This filename can be up to 10 characters
11		MSB OF THE NUMBER OF SECTORS USED IN THE FILE.
12		LSB OF THE NUMBER OF SECTORS USED IN THE FILE.
13		TRACK NUMBER FOR START OF FILE
14		SECTOR NUMBER FOR START OF FILE
15-209		SECTOR ADDRESS MAP (195 BYTES) (detailed further on).
210-219		MGT FUTURE AND PAST (10 BYTES) These were used in the PLUS D directory but are not used by the SAMDOS. They are allocated to MGT for future use.
220	15	FLAGS (MGT USE ONLY)
221-231		FILE TYPE INFORMATION
	16-26	If the file type is 17 or 18 then these bytes contain the file type/length and name.
	16	If the file type is 20 then these bytes contain the screen mode.
	16-18	If the file type is 16 then these bytes contain the program length excluding variables.
	19-21	If the file type is 16 then these bytes contain the program length plus numeric variables.
	22-24	If the file type is 16 then these bytes contain the program length plus numeric variables and the gap length before string and array variables.
232-235	27-30	SPARE 4 BYTES (reserved).

236	31	START PAGE NUMBER, in bits 4-0, bits 7-5 are undefined.
237-238	32-33	PAGE OFFSET (8000-BFFFH). This is as per file header, although when the ROM passes a file to be saved it starts it in section C of the addressing map.
239	34	NUMBER OF PAGES IN LENGTH. (as per file header)
240-241	35-36	MODULO 0 TO 16383 LENGTH, i.e. length of file MOD 16384. (as per file header)
242-244	37-39	EXECUTION ADDRESS Execution Address, if CODE file, or line number if an auto-running BASIC program.
245-253	40-47	SPARE 8 BYTES
254-255		FOR FUTURE USE BY MGT ONLY.

- SECTOR ADDRESS MAP -

SAMDOS allocates 195 bytes to the sector address map, giving 1560 bits, which is the exact number of sectors available for storage on the drive.

A sector address map is calculated for each directory entry. when a file is created a directory entry is made for that file. A sector address map is created by setting the specific bit(s) corresponding to the sector(s) allocated to the file. (Bit 0 of the first byte is allocated to track 4 sector 1). For example, if the file uses 5 sectors then five corresponding bits in the sector address map are set and saved as part of the directory entry.

- BIT ADDRESS MAP (BAM) -

The bit address map is not stored on the disk by SAMDOS. It is generated by performing a bitwise OR of each file's sector address map. This then gives SAMDOS a usage map of the disk. When a file is created the first thing SAMDOS does is calculate the BAM, and then by looking at the available sectors (i.e. bits not set), it can work out if there is room for the file. If there is room for the file, then the directory entry is created, including the sector address map specific to the new file, and the file is stored in the sectors which have been specified in the file's sector address map.

SAMDOS INTERFACING

SAMDOS's machine code interface provides pointers to specific parts of the DOS, called hook codes. These pointers perform the DOS functions, e.g., SAVE, LOAD, VERIFY, etc.

To use these hook codes various registers in the Z80B must be set up for the subroutine called to perform the required tasks. Each hook code is described later in this manual together with the necessary register information.

When a hook code is used the ROM will page SAMDOS temporarily into section B of the 64K addressing space (ie at 400011):

```
      RST 08H           ; call ROM RST 8
      DEFB x            ; Where x equals the specific
                        ; Hook code required.
```

If an error occurs it will normally be handled by the ROM, producing an error message in the lower part of the screen. However, if the system variable DOSER (5BC0H) is loaded with the address of your own 'return from DOS' routine then this routine will be entered after SAMDOS has executed any hook codes, or examined any of BASIC's error codes, with the Accumulator holding zero for no error, or an error number. This will be either one of BASIC's error codes or a SAMDOS error code of 128 and above.

- USER INFORMATION FILE AREA (UIFA) -

When using SAMDOS hook codes the calling program must set up an area in memory, pointed to by the IX register, which details the information required for file handling, e.g., SAVE and LOAD. This file is called the UIFA.

Byte	Description
0	STATUS/FILE TYPE.
1-14	FILENAME. 14 characters are allocated to allow for device identification, for example: D1:filenamexx. SAMDOS will strip off the device identifier, so the maximum length of a filename is still ten characters.
15	FLAGS
16-26	If the file type is 17 or 18 then these bytes contain the type/length byte and the name.
16	If the file type is 20 then this byte contains the screen mode.
16-18	If the file type is 16 then these bytes contain the program length excluding variables.
19-21	If the file type is 16 then these bytes contain the program length plus the numeric variables.
22-24	If the file type is 16 then these bytes contain the program length plus the numeric variables and the gap length before the character variables.
27-30	SPARE 4 BYTES (Reserved)
31	16K PACE NUMBER START
32-33	PAGE OFFSET (8000-BFFFH) LSB/MSB
34	NUMBER OF PAGES IN LENGTH

35-36 MODULO 0 TO 16383 LENGTH ie file length MOD 16384.
37 EXECUTE PAGE NUMBER if applicable
38-39 EXECUTE OFFSET (8000-BFFFH) LSB/MSB if applicable
40-47 SPARE 8 BYTES (Comment Field)

When the DOS has used the UIFA and it wants to pass a UIFA back, eq, for confirmation of load by the calling program the DOS creates a Disk Information File Area (DIFA). Both the UIFA and DIFA are 48 bytes long. This confirmation DIFA is written to an area of memory 80 bytes above the user's specified UIFA.

- SAMDOS HOOK CODES -

SAMDOS provides the user with hook (command) codes which enable the machine code programmer to use SAMDOS's facilities without having to return to or call SAM BASIC.

If an error occurs, SAMDOS set the Carry flag and put an error number into the Accumulator. If no error occurs the Accumulator will equal zero.

The Hook Codes currently available are:

INIT	128	dec	Initialise and look for AUTO file
HGTHD	129	dec	Get file header
HLOAD	130	dec	Load file
HVERY	131	dec	Verify file
HSAVE	132	dec	Save file
HVAR	139	dec	Get address of DVAR
HOFLE	147	dec	Open a File
SBYT	148	dec	Save a byte
HWSAD	149	dec	Write a sector to the disk
HSVBK	150	dec	Save a block of data
CFSM	152	dec	Close file sector map
HGFLE	158	dec	Get a file from disk
LBYT	159	dec	Load a byte
HRSAD	160	dec	Read a sector from the disk
HLDBK	161	dec	Load a block of data from disk
REST	164	dec	Restore disk drive to track 0
PCAT	165	dec	Perform a directory listing
HERAZ	166	dec	Erase a file from disk

Provision has been made for further hook codes to be added to SAMDOS.

HOOK CODE EXPLANATIONS

When the hook code explanation refers to 'RPT', it refers to the pointer used internally by SAMDOS.

INIT This routine looks for an AUTO file on the current disk, and initialises the DVARS.

HGTHD Get file header. This routine should be called with IX pointing to the UIFA, which should contain the file type and filename required. When completed the complete file header will be transferred in DIFA form to IX+80 bytes.

HLOAD Load file in UIFA pointed to by IX register. The C register contains the number of 16K pages used by the file, while DE must contain the length modulo 16K. The HL register pair must point to a destination between 8OOOH to BFFFH, while the destination page must be paged in using the HMPR register. These values can be obtained from the header loaded by HGTHD.

HVERY Verify the memory to the file stored on the disk. Again the IX register must be a pointer to the file UIFA. Use as HLOAD, but verifies rather than loads.

HSAVE Save the file whose UIFA is pointed to by the IX register. The UIFA must be a complete UIFA, including file length, etc.

HVAR This routine calls the jump table thus unstacking the number following the DVAR into BC. The routine supplies the address of the DVAR by putting it onto the BASIC floating point calculator stack.

HOFLE Open a file on the disk. ix roust point to the UIFA. The routine will create a sector address map, and save the header to the disk and reset pointer RPT.

SBYT Save the byte in the Accumulator to the RAM pointed to by the pointer RPT. If the sector is full the data will be stored in the next sector pointed to by the sector address map.

HWSAD D contains the track number, and E contains the sector number. The Accumulator holds the drive number (1 or 2). Writes the sector pointed to by the DE register pair. The Accumulator contains the drive number, while the HL register pair is the pointer to the source data which must be resident in the 64K address map.

HSVBK Save a block of data to the disk where the DE register pair points to the start of the data, and the BC register pair holds the byte count.

CFSM Close file sector map. This routine empties the RAM and copies the header area on to the directory, closes the file, then updates the directory.

HGFLE Get a file from the disk. The IX register must point to the UIFA. The return is made with the first sector of the file loaded into RAM and RPT pointing to the first byte.

LBYT Load the byte pointed to by RPT from RAM, place it in the Accumulator, and increment the RPT. When the sector has all been read then the next sector is loaded from the disk and the pointer adjusted.

HRSAD D contains the track number, and E contains the sector number. The Accumulator holds the drive number (1 or 2}. Reads the sector pointed to by the DE register pair. The Accumulator contains the drive number, while the HL register pair is the pointer to the destination.

HLDBK Load a block of data from the disk to the memory pointed to by DE with the block count in BC.

REST Restore disk drive to track 0. The Accumulator holds the drive number , ie, 1 or 2.

PCAT Perform a directory listing to current stream.

HERAZ Erase a file from the disk. Register IX must point to the UIFA of the file to be erased.

- LOCATING SAMDOS -

When SAMDOS is loaded the ROM looks at its available memory and loads SAMDOS into the last free 16K page. The ROM uses the last two 16K pages for SCREEN 1, so SAMDOS usually loads into the third from last 16K page, but the page will be different if extra screens have been opened before SAMDOS is loaded. Address 5BC2H (SVAR 450) holds the page number used by SAMDOS, or zero if SAMDOS has not been loaded.
PEEK SVAR 450*16384+16384 will give the start address of SAMDOS, When a DOS command is issued the DOS is loaded into section B (4000H) of the 64k addressing space, the command is performed, and the DOS is then paged out.

- SAMDOS ERROR CODES -

81	Nonsense in SAMDOS
82	Nonsense in SNOS (SAM Network operating system)
83	statement end error
84	Escape requested
85	TRK nnn SCT nnn error
86	Format TRK nnn lost
87	Check disk in drive
88	No BOOT file
89	Invalid file name
90	Invalid Station
91	Invalid device
92	variable not found
93	verify failed
94	Wrong file type
95	Merge error
96	Code error
97	Pupil set
98	Invalid code
99	Reading a write file
100	writing a read file
101	No AUTO file
102	Network off
103	No such drive
104	Disk is write protected
105	Not enough space
106	Directory full
107	File not found
108	End of file
109	file name used
110	No SAMDOS loaded
111	Stream used
112	Channel used