Video Technology

Laser 200

FOREWORD

This manual is intended to provide owners of the VZ200/300 series of personal colour computers with additional information to assist in programming, operation and expansion. All reasonable care has been taken to ensure that the information contained herein is accurate and correct; however no responsibility can be accepted, nor liability assumed for either its accuracy or suitability for any particular purpose. Dick Smith Management Pty Ltd reserves the right to make circuit, software and/or mechanical changes to the products described herein, without notice.

Although much of this information contained herein will be of interest to all VZ200/300 owners, it is assumed that the reader is reasonably familiar with the technicalities of digital computer electronics. It is strongly recommended that owners without suitable experience in the field of computer service techniques not attempt to repair or modify their computer's equipment.

CONTENTS

THE BASIC COMPUTERS

The VZ200/300 computers employ a Z80A microprocessor running at approximately 3.6MHz.
A Microsoft Basic interpreter and I/O routines are contained in 16K of mask-programmed ROM.
Included in the computers are user RAM, a PAL colour video display circuit, a VHF RF modulator,
a "QWERTY" keyboard, a cassette interface and a simple sound effects circuit.
Also included is provision for memory and I/O expansion via two rear edge connectors.
Devices which can be plugged into these connectors include a 16K RAM expansion cartridge,
printer interface, floppy diskette interface and game joysticks.

SPECIFICATIONS

	VZ200	VZ300
CPU	Z80A	Z80A

CLOCK SPEED	3.58MHz		3.54MHz
INTERNAL ROM	16K	16K	
INTERNAL RAM	6K	16K	
DISPLAY RAM	2K	2K	

SCREEN FORMAT

MODE 0

16 lines of 32 characters. 128,upper-case text characters in normal or inverse format plus 128 2 pixel X 2 pixel chunky graphics characters in 8 colours.

MODE 1

64 rows of 128 individually addressable pixels in 4 colours.

VIDEO OUTPUT

1V p-p into 75 ohms composite video, negative sync. PAL colour encoded.

RF OUTPUT

lmV into 75 ohms VHF Ch. 1 (57.25Mhz) PAL colour encoded.

KEYBOARD

45 key "QWERTY" style.

POWER SUPPLIES

The computer is powered from a 10-12V 800MA dc source. Normally this will be an approved "plug-pack" although battery powered operation is also possible. In both models the raw dc input is regulated by a 3 terminal regulator IC to 5V dc which powers most of the internal circuitry.

In the VZ200, the colour encoder circuitry requires a +12V rail which is generated from the +5V rail by a regulating inverter circuit.

In the VZ300, additional supply rails of +12V and -5V are required to power the dynamic RAMS. These voltages are generated from the raw dc supply by an inverter circuit comprising Q2, Q3 and associated components.

CPU AND ASSOCIATED CIRCUITRY

The CPU is clocked by a 3.5795MHz crystal oscillator, comprising 3 inverters (Ul3).

The -RESET pulse is generated by a simple RC circuit and buffered by 2 inverters (Ul3).

The -INT input is activated during screen retrace periods by the video circuitry. The interrupt is serviced by a ROM routine which performs some housekeeping and provides a user book. The condition of the -INT input can also be sampled as bit 7 during reads of the keyboard addresses (680OH-6FFFH).

The signals, -NMI, -WAIT, -RFSH, -Ml, -HALT, and -IORQ are not used within the machine but are available at the rear expansion connectors, as are all of the Z80 address, data, control and status signals with the exception of -BUSACK and -BUSREQ.

VZ300

The VZ300 differs in the following ways.

The CPU is clocked at 3.5469MHz. This is obtained by division of the master oscillator by 5 within UIO. The 17.734MHz master oscillator, comprising 3 inverters (u9), is also divided by 4 to provide the 4.43362MHz PAL subcarrier.

The RC derived -RESET pulse is buffered by 2 inverters of U9.

RAM AND ROM

VZ200

6K of program RAM is provided, implemented as three 2K x 8 static RAMS (U2', U3', U4') mounted on a small daughter board. U2' occupies addresses 780OH-7FFFH and is enabled by the address decoder circuit on the main pcb (U2, U3). U3' and U4' occupy addresses 800OH-87FFH (U3') and 880OH-8FFFH (U4') and are enabled by Ul'.

Another 2K x 8 RAM (U7) is used for the video display buffer. The video RAM occupies addresses 7600H,-77FFH and is enabled for CPU access by U2 and U3. For CPU reads, the video RAM data is buffered by U14. The video RAM address lines are decoupled from the CPU address lines by series resistors to avoid conflicts between the CPU and the Video Display Processor (Ul5) at times other than CPU access to the video RAM.

The BASIC interpreter and I/O routines are contained in 16K of ROM addressed in the range OOOOH-3FFFH. In early VZ200s this is implemented as two 8K x 8 devices (U9, U10). U9 occupies addresses, 0000-lFFFH and U10 occupies 200OH-3FFFH. Later machines use a single 16K x 8 ROM. To address the larger ROM over the 0000-3FFFH range, A13 is taken to the ROM (pin 26) and the ROM chip select (pin 20) is generated by 'ORing' the two ROM select signals from U3 with a pair of diodes.

VZ300

16K of program RAM is provided implemented as eight 16K x 1 dynamic RAMS (Ul-B). The RAM occupies the 16K address block 780OH-B7FFH. A custom gate array (UlO) contains all of the necessary circuitry to enable the RAM, multiplex the CPU address and provide the correct -CAS and refresh timing.

A 2K x 8 RAM (Ul6) is used for the video display buffer. The video RAM occupies addresses 700OH-77FFH and is enabled for CPU access by the address decoder within U14. For CPU reads, the video RAM data is buffered by U14. The video RAM address lines are decoupled from the CPU address lines by series resistors to avoid conflicts between the CPU and the Video Display Processor (Ul5-) at times other than CPU access to the video RAM.

The BASIC interpreter and I/O routines are contained in a single 16K x 8 ROM enabled for the 0000-3FFFH address range by address decoding circuitry within U13.

THE KEYBOARD

The 45 keys are arranged in a 6 x 8 matrix. Each of the 8 rows effectively occupies a specific memory address (actually, a series of addresses due to the simplified decoding) in the range 6800-6FFFH. The individual keys are mapped onto the least significant 6 bits of that location, according to the column they occupy.

The 8 least significant bits of the address bus pull down the rows of the matrix through diodes. The keyboard is scanned by software sequentially taking each of these 8 lines to a logic low level. If the upper 8 address lines represent 68H (or, in fact, 69H-6FH) then the condition of the 6 key columns o the particular row will be enabled onto the data bus through U12.

For example, if the '2' key were pressed, it would cause bit 1 at address 68V7H to drop to 0. The data retrieved by reading that address, neglecting the 2 most significant bits which are not driven by the keyboard, would be 3DH (binary 111101).

The keyboard matrix and its (lowest) row addresses are shown below. Note that each key causes a logic 0 to appear at the bit position shown, when its row address is read.

ROW ADDRESS	BIT POSITION 5	4	3	2	1	0
68FEH	R	Q	E		W	T
68FDH	F	A	D	CTRL	5	G
68FBH	V	Z	C	SHFT	X	B
68F7H	4	1	3		2	5
68EFH	M	SPC				N
68DFH	7	0	8		9	6
68BFH	U	P		I RETN	0	Y
687FH	J		K		L	

VZ300

The VZ300 keyboard is logically the same as the VZ200, although it is read through a custom I/O IC (14). Physically the VZ300 keyboard differs in that it uses the more common, moulded keys and has a full space bar.

THE VIDEO INTERFACE

The heart of the video interface is a 6847 video Display processor. This IC contains the upper-case ASCII and chunky graphics character generator, and, logic to produce the dot addressable graphics, the video timing signals, the video RAM

control and address signals, a video luminance (Y) output and 2 matrixed colour outputs (R-Y and B-Y)

5OHz SYNC GENERATION

The 6847 is intended to produce 6OHz vertical synchronization signals and 262 lines per field. In order to produce 5OHz 312 line video signals, 50 extra lines must be added to each field. This is achieved by 3 counters, U18, U20, U21 and associated logic.

When the VDP outputs -FS, the reset inputs to U20 are released, allowing it to count video lines from the VDP. U20 counts the first 25 lines of the bottom border and then inhibits the 3.58MHz video clock via U16 and U19. Instead of clocking the VDP, the clock is fed to U18 which is configured as a divide-by-228 counter. U18 generates horizontal sync pulses (between clock edges 208 and 228) during the period that the VDP is disabled. U21 counts these "dummy" video lines. When 25 additional lines have been completed, the clock is switched back to the VDP. The VDP generates a further 7 lines before resetting -FS. This again inhibits the VDP and allows U18, U21 etc. to insert a further 25 "dummy" lines. The VDP is then allowed to operate as normal for the next 230 lines after which the cycle repeats itself. In summary, starting from the falling edge of -FS, the 312 line cycle is as follows:

-25 lines of bottom border	(from	VDP)
-25 lines of bottom border	(from	U18 etc.)
-1 line of bottom border	(from	VDP)
-6 lines vertical retrace	(from	VDP)
-13 lines of blanking	(from	U18 etc.)
-12 lines of top border	(from	U18 etc.)
-38 lines of top border	(from	VDP)
-192 lines of active display	(from	VDP)

VIDEO DISPLAY MODES

The video interface operates in one of two modes, text/low-res graphics (MODE 0) or hi-res (MODE 1). The display mode is determined from the -A/G input on the VDP (pin 35). This input is controlled by bit 3 of the Cassette/speaker/VDP control latch. If bit 3 is set then MODE 1 is enabled.

MODE 0

In MODE 0 the screen is organised as 16 rows of 32 characters. Each screen location is represented by a unique memory location in the first 512 bytes of the video RAM (i.e. 7000H - 71FFH, or 28672 - 29183 decimal).

The background colour in this mode is determined by the condition of pin 39 of the VDP (CSS). If CSS is set, then the background colour is orange; if it is reset then the background is green. CSS is controlled by bit 4 of the Cassette/speaker/VDP control latch.

A total of 256 different characters can be displayed consisting of 64 upper-case characters, the same 64 characters in inverse format and 128 lo-res graphic characters. Bit 7 of the video character data determines whether the character is text (bit 7=0) or graphic (bit 7=1).

If bit 7 is reset, indicating a text character, then bit 6 determines whether it is displayed in normal (bit 6=0, light on dark) or inverse (bit 6=1, dark on light) format. The remaining 6 bits are the character code.

If bit 7 is set, indicating a graphic character, then bits 4, 5 and 6 indicate the colour of the character and bits 0, 1, 2 and 3 determine its shape. Each of the 4 least significant bits corresponds to a pixel in a 2 x 2 matrix which occupies the same

screen area as a text character.

The 3 bit colour code is: Bits 0-3 of the graphics character code are mapped onto pixels as shown below:

B6	B5	B4	HEX	COLOUR
0	0	0	0	Green
0	0	1	10	Yellow
0	1	0	20	Blue
0	1	1	30	Red
1	0	0	40	Buff
1	0	1	50	Cyan
1	1	0	60	Magenta
1	1	1	70	Orange

MODE 1

In this mode the screen is organized as 64 rows of 128 pixels, giving a total of 8192 pixels. Each pixel can be displayed in one of four corners, one of which is the background colour. This means that for each of the two possible background colours, each pixel can be either 'turned off' (ie the same colour as the background), or displayed in one of three colours.

The video RAM coding scheme used for this display mode uses each byte to encode four adjacent pixels. This means that each pixel is encoded in two bits. To illustrate this, here is the coding for the first four pixels on the screen, in the top left hand corner:

ADDRESS 7000H: B7 B6 B5 B4 B3 B2 Bl B0

The next four pixels along the line are stored in location 7001H, and so on. The 2-bit colour coding used for each pixel is shown below:

(i) Background colour 0 (green):

00=GREEN (background colour)
01=YELLOW
10=BLUE
ll=RED

(ii) Background colour 1 (buff):

00=BUFF (background colour)
01=CYAN
10=ORANGE
ll=MAGENTA

Note that from BASIC, any pixel may be individually turned on or off usinia the SET(x,y) and RESET(x,y) commands, and given various colours using the COLOR(m,n) command.

Video display worksheets for both mode(o) and mode(l) are given at the rear of this manual. These can be very handy for planning the display screens, menus etc when you are writing programs. Feel free to photocopy these worksheets, so you can use the photocopies in this way.

I/O MAPPING

The Z80A microprocessor in the VZ200/300 can address 256 ports in I/O space (ie ports 0 - FF hex). The following I/O address ranges have been allocated for expansion peripherals:

I/O ADDRESS (hex)		DEVICE
00	OF	Printer
10	1F	Floppy disk controller
20	2F	Joystick interface
30	3F	Communications MODEM
70	7F	Memory bank switch

VZ200/300 CASSETTE/SPEAKER/VDC OUTPUT LATCH

An internal latch is used to generate the cassette output, the drive for the internal piezo speaker, and two control signals for the video display controller chip (6847). The latch is write-only and memory-mapped occupying all addresses from 6800H - 6FFFH (26624 - 28671 decimal) inclusive. In the VZ200 this latch is Ul (74LS174), whereas in the VZ300 this latch is part of U14 (the GA004 LSI). A bit-map of the latch is shown below:

WEIGHTING BIT FUNCTION

Hex	Dec				
20	32	5	Speaker B		
10	16	4	VDC Background	0	green 1 orange (text) buff (graphics)
08	8	3	VDC Display Mode	0	mode 0, text / low res. 1 mode 1, graphics / hi-res.
04	4	2	Cassette out (MSB)		
02	2	1	Cassette out (LSB)		
01	1	0	Speaker A		

speaker

The speaker is driven in push-pull fashion by bits 0 and 5. To make the speaker sound a note, the software should toggle bits 0 and 5 alternately at the required rate. ie when bit 0 is a logic Oil, bit 5 should be logic 'O' and vice-versa. Note that when this is done the software should not alter the other bits of the latch.

Cassette output

Bits 1 and 2 are used to generate the cassette recording signal, which is approximately 200 millivolts peak-to-peak.

VDC display mode

The VDC display mode is controlled by bit 3. If bit 3 is a logic 10', the VDC will operate in its text/low-resolution mode. If bit 3 is made logic '1', the VDC operates in its hi-res graphics-only mode.

VDC background colour control

Bit 4 is used to control the VDC background colour. In text/low- res mode (mode 0), a '0' on bit 4 gives a green background colour while a '1' on bit 4 gives an orange background. In hi-res mode (mode 1) a '0' on bit 4 gives a green background, while a '1' gives a buff background.

JOYSTICKS

The two Joystick units are connected to a plug-in module that contains I/O address decoding and switch matrix encoding. IC U2 (74LS138) enables I/O reads between 20 - 2F Hex. Address lines AO - A3 are used separately to generate active LOW signals on the joystick or switch to be read. Switch state is then read at the resultant address from Data bits DO - D4. When a switch is ON it provides an active-low Data bit. As below:

1 = Right-hand joystick, 2 = Left-hand joystick

I/O Address	Hex	Joystick Switch	Data (Hex)
2E (46 dec.)	1	Up	FE
	1	Down	FD
	1	Left	FB
	1	Right	F7
	1	Fire	EF
2D (45 dec.)	1	Arm	EF
2B (43 dec.)	2	Up	FE
	2	Down	FD
	2	Left	FB
	2	Right	F7
	2	Fire	EF
27 (39 dec.)	2	Arm	EF

VZ200/300 DISK CONTROLLER

This is a plug-in port-mapped device capable of supporting two X-7302 disk drives. The Disk Controller occupies the I/O address space from 10 Hex to 1F Hex of the port map. Effectively only 4 I/O locations are used to control and read back data from the Disk Drives.

I/O address	Function
10 Hex	Latch (write-only)

Bit 0 - 3: Stepper-motor control phases (active HIGH)
Bit 4: Drive 1 enable.(active LOW)
Bit 5: Write data (active HIGH)
Bit 6: Write request (active LOW)
Bit 7: Drive 2 enable (active LOW)

11 Hex DATA (read-only)

Bit 0 - 7: Data byte read from disk

12 Hex POLLING (read-only)

Bit 0 - 6: not used
Bit 7: clock bit polling input

13 Hex WRITE PROTECT STATUS (read-only)

Bit 0- 6 : not used
Bit 7: 1 = write-protect
0 = no write-protect

VZ200/300 DISK DRIVE

General Operation

The X-7302 VZ200/300 floppy disk drive consists of read/write, control and drive motor electronics, drive mechanism, read/write head, and track positioning mechanism. These components perform the following functions:

i) Receive and generate control signals
ii) Position of the read/write head to the desired track
iii) Read/write of data
iv) Control of drive motor speed

READ/WRITE and CONTROL ELECTRONICS

The three electronic boards contain:

i) Stepper motor driver
ii) Write amplifier
iii) Read amplifier and control circuits
iv) File protect sensor
v) Drive enable circuit
vi) Drive motor control circuit

DRIVE MECHANISM

The drive motor rotates the spindle at 85 rpm through a beltdrive system. The speed of the motor is controlled by a tachofeedback servo circuit. A hub clamp that moves in conjunction with the door closure mechanism centres and clamps the floppy disk onto the spindle hub.

R/W HEAD POSITIONING MECHANISM

The R/W head is positioned to the desired track by applying the control signals to the stepper motor. The connection between the head carriage and the stepper motor is through a steel belt. The stepper motor rotates 2 steps per track.

R/W HEAD

The R/W head is used to read/write data to and from the floppy disk. The R/W head is mounted on the head carriage which moves on rails and is positioned by the stepper motor. The floppy disk is held on a plane perpendicular to the R/W head.

TRACK 0 STOPPING MECHANISM

After powering on and track location failure, the position of the R/W head is indeterminant. In order to assure proper positioning of the R/W head after powering on, a step-out operation (recalibration) is performed until it is locked at track 00 by the track 00 stopper.

DRIVE SELECTION

The drive is selected by activating the -BENBL line. After being selected, the drive motor and the LED on the front panel bezel will be on.

FILE PROTECTION MECHANISM

The file protect mechanism is constructed with a LED and phototransistor to detect the existence of the write enable notch of the disk jacket. When a disk with the notch covered is installed and the light passing for detection is disturbed, no write or erase current will flow through the R/W head. The recorded information on the disk is protected from an erroneous input of a write command.

FUNCTION of TEST POINTS

i) TP1, TP3 PCBA Control and R/W amplifier.

Test points for observing the read pre-amplifier output signals after passing through the low-pass filter. Hence TP1 and TP3 are used for the check and adjustment of the head seek mechanism. ie track alignment.

For observation of the read waveforms, use two channels of an oscilloscope with one channel set to INVERT mode and then ADD both channels. Use test point TP2 for the oscilloscope ground. This method will display full 'balanced' signal, if these modes are unavailable on oscilloscope then observe waveform using single oscilloscope channel from either TP1 or TP3 and TP2 as ground.

ii) TP2, TP5 are both system ground terminals.

iii) TP6 is a test point for observing read data pulses.

iv) TP4 is not used.

FUNCTION of VARIABLE RESISTORS

i) VR1 PCBA control and R/W amplifier
VR1 is used for adjusting peak shift of the read data.

ii) VR2 PCBA drive motor
VR2 is used for adjusting the rotational speed of the spindle.

TROUBLESHOOTING GUIDE

TOOLS and EQUIPMENT

i) Dual channel oscilloscope with Differential Mode input (ie ADD, INVERT), of 1OMHz or better
ii) Frequency counter
iii) VZ200/300 and Disk Controller
iv) Software: DISK CONTROL program (for controlling the stepping motor to move the R/W head for alignment and TRK 00 recalibration). Refer to suggested listing)
v) DYSAN 48 TPI alignment disk. (#206-10)
vi) Cleaning disk (if available)
vii) Working disk
viii) Another VZ200/300 Disk Drive (used as working disk)
ix) Screwdrivers: PHILIPS screwdriver, 5mm Blade screwdriver, 3mm
X) Hexagon wrench key, 1.5mm
xi) Locking agent (ie nail-polish)

GENERAL PROCEEDURE

i) Remove the top and bottom cases by removing securing screws under unit.

ii) Set up the computer with the working drive as Drive 1 and the Drive under test as Drive 2.

iii) Connection and disconnection of connectors.
iv)
Note-complete orientation and position of connectors before removing-them. Be sure to turn the power OFF before connecting or disconnecting the connectors. When plugging or removing connectors, this should be done without applying excessive force to the cables or post pins.

iv) If the LED on the front bezel is ON but the Drive Motor remains stationary, check that the connectors are securely connected.

CHECK and ADJUSTMENT of DISK ROTATION SPEED.

i) Install Alignment Disk in disk drive to be checked. Select DRIVE by typing DIR command.

ii) Use the Frequency Counter to monitor the output of test point TP1 (Ground on TP2).

iii) The reading of the frequency counter should be 35.417 kHz.
If the frequency is off by more than 1 kHz (approx 3%) then adjust VR2 on the Drive Motor PCB.

iv) After checking that this measurement is satisfactory, fix VR2 with a small drop of locking agent.

CHECK and ADJUST of TRACK ALIGNMENT

i) connect two channels of the oscilloscope to TP1 and TP3 on the Control and R/W amplifier PCB.

ii) oscilloscope setting:2OmS/division, CH.A and CH-B both AC mode .5V/division

iii) Set one Channel to INVERT and.ADD both channels.

iv) Load the DISK DRIVE CONTROL program.
v) Install the Alignment disk in the Drive to be tested.
vi) Using the control program, send the head carriage to TRK 16.

vii) The lobe patterns displayed should be within 70% of each other, see diagram below. If they are, then no adjustment is required. If they are not, then proceed with adjustment.

viii) Loosen the stepper-motor fixing screws and while observing the waveform, turn the stepper motor to correct the lobe pattern.

ix) Check that the adjustment is stable by stepping off TRK 16 in both directions and returning.

x) once corrected and stable, tighten the stepper-motor fixing screws, and seal with a small amount of locking agent.

CHECK of FILE PROTECT SENSOR

i) Load Disk control Program

ii) Insert a work disk without a write-protect tab, halfway into the disk drive.

iii) Use the 'P' command to check the drive status. The message 'DISK IS WRITE PROTECTED' should appear.

iv) Now fully insert and close door, the message 'DISK IS NOT WRITE PROTECTED' should appear.

NOTE: If any of the above adjustments do not rectify the Disk Drive's problem, then return the Drive to a Dick Smith Service Dept for a detailed diagnosis.

PREVENTIVE MAINTAINANCE

If the DISK DRIVE is used in a dusty environment, it is suggested that a periodic cleaning is made of the magnetic-head suface.

i) Setup DISK DRIVE in position 2.

ii) If a CLEANING DISK is available, insert this and using the DISK CONTROL program move the R/W HEAD between track 00 and the innermost track several times.

iii) If CLEANING DISK is unavailable. Remove covers to gain access to R/W HEAD assembly.

iv) Use a cotton swab lightly dampened with pure alcohol. Carefully lift the HEAD LOAD PAD ARM and clean the R/W Head and surrounding area. Wipe the HEAD surface with a clean dry cloth after the alcohol has evaporated. Be sure to inspect the area for dirt or fluff left on the HEAD surface, before letting the HEAD LOAD PAD ARM down.

v) Reassemble and check for normal operation.

LISTING OF DISK CONTROL PROGRAM

```
10      REM DISK CONTROL PROGRAM
20      REM A. LATCH CONTROL -----
30      REM I/O ADR:10H
40      REM BIT 0 - BIT 3:
50      REM STEPPER PHASE CONTROL
60      REM BIT 4:DRIVE 1 ENABLE
70      REM BIT 5:WRITE DATA
80      REM BIT 6:WRITE REQUEST
90      REM BIT 7:DRIVE 2 ENABLE
100     REM     B. DATA STROBE          -----
110     REM     I/O'ADR : 11H
120     REM     BIT 0 - BIT 7:
130     REM     DATA BYTE READ FROM
140     REM     DISK DRIVE
150     REM
160     REM     C. POLLING     -----
170     REM     I/O ADR : 12H
180     REM     BIT 0 - BIT 6:
190     REM     NOT USED
200     REM     BIT 7: CLOCK BIT
210     REM     POLLING INPUT
220     DIM D(4)
230     CLS
240     PRINT:PRINT TAB(6)"DISK CTRL PROGRAM"
250     PRINT
260     PRINT:PRINT TAB(6)"COMMANDS:"
270     PRINT:PRINT TAB(6)"R RECALIBRATION"
280     PRINT TAB(6)"G GOTO TRACK
290     PRINT TAB(6)"I STEP IN
300     PRINT TAB(6)"O STEP OUT
310     PRINT TAB(6)"P CHECK WRITE PROTECT"
320     PRINT TAB(6)"Q QUIT"
330     PRINT:INPUT "COMMAND ";A$
340     IF A$="R" THEN GOSUB 410
350     IF A$="I" THEN GOSUB 560
360     IF A$="O" THEN GOSUB 700
370     IF A$="G" THEN GOSUB 900
380     IF A$="Q" THEN GOSUB 870
390     IF A$="P" THEN GOSUB 1070
400     GOTO 330
410     REM - RECALIBRATE R/W HEAD
420     P=O
430     OUT 16,192
440     FOR J=l To 24
450     FOR I=3 TO 0 STEP -1
460     D(I)=l:GOSUB 1050
470     OUT 16,192+LA
480     D(I)=O:GOSUB 1050
490     OUT 16,192+LA
500     NEXT
510     NEXT
520     D(O)=l:D(1)=O:D(2)=O:D(3)=O
530     OUT 16,193
540     TC=O
```

```
550     GOSUB 840:RETURN
560     REM MOVE THE R/W HEAD TO
570     REM INNER TRACKS
580     IF TC=39.5 THEN RETURN
590     D(P/2)=O
600     GOSUB 1050
610     OUT 16,192+LA
620     P=P+2
630     IF P=8 THEN P=O
640     D(P/2)=l
650     GOSUB 1050
660     OUT 16,192+LA
670     TC=TC+.5
680     GOSUB 840
690     RETURN
700     REM - MOVE THE R/W HEAD
710     REM TO THE OUTER TRACKS
720     IF TC=O THEN RETURN
730     D(P/2)=O
740     GOSUB 1050
750     OUT 16,192+LA
760     P=P-2
770     IF P=-2 THEN P=6
780     D(P/2)=l
790     GOSUB 1050
800     OUT 16,192+LA
810     TC=TC-.5
820     GOSUB 840
830     RETURN
840     REM - SHOW TRACK NUMBER
850     PRINT "TRACK = ";TC
860     RETURN
870     REM - EXIT THE PROGRAM
880     OUT 16,40:END
890     REM - MOVE THE R/W HEAD TO THE DESIRED TRACK
900     INPUT "ENTER TRACK NUMBER =";TN
910     IF (TN>39.5) OR (TN<O) THEN GOTO 900
920     TT=TN-TC
930     IF TT<=O THEN 990
940     TT=TT*2
950     FOR CN=l TO TT
960     GOSUB 560
970     NEXT
980     RETURN
990     IF TT=TC THEN RETURN
1000    TT=TT*(-2)
1010    FOR CN=l TO TT
1020    GOSUB 560
1030    NEXT
1040    RETURN
1050    IA=D(3)*8+D(2)*4+D(1)*2+D(O)
1060    RETURN
1070    WP=INP(19)
1080    IF WP>127 THEN PRINT "DISK IS WRITE-PROTECTED"
1090    IF WP<128 THEN PRINT "DISK IS NOT WRITE-PROTECTED"
1100    RETURN
```

VZ200/300 SCREEN CONTROL CODES

The following codes can be used for screen control from BASIC:

Cursor left	PRINTCHR$(8)	Cursor right	PRINTCHR$(9)
Cursor up	PRINTCHR$(27)	Cursor down	PRINTCHR$(10)
Rubout	PRINTCHR$(127)	Insert	PRINTCHR$(21)
Home	PRINTCHR$(28)	Clear screen	PRINTCHR$(31)

VZ200/300 SYSTEM POINTERS AND VARIABLE STORAGE LOCATIONS

POINTER or VARIABLE	HEX LOC	DECIMAL
Top of memory (ptr)	78Bl/2	30897/8
Start of BASIC program (ptr)	7BA4/5	30884/5
End of BASIC program (ptr) (also start of simple variable table)	78F9/A	30969/70
Start of dim. variables table (ptr)	78FB/C	30971/2
End of BASIC's stack (ptr) (also start of string variable storage area)	78A0/l	30880/1
Execute address for USR program (note: high byte of address must go in 788F)	788E/F	30862/3
Interrupt exit (called upon interrupt)	787D/E/F	30845/6/7
Start of BASIC line input buffer (buffer is 64 bytes long - 2 screen lines)	79E8	31208
Copy of output latch	783B	30779
Cursor position	78A6	30886
output device code (0=video, 1=printer, -1=cassette)	789C	30876

The contents of the BASIC stack pointer stored in 78A0/l are basically equal to the contents of the 'top of memory' pointer stored in 78Bl/2, less a figure equal to the number of bytes reserved for string storage. The default value for string storage space is 50 bytes; this can be modified from within a basic program by using the CLEAR command - ie CLEAR 1000 will increase the string space to store 1000 bytes.

The VZ200/300 printer interface uses I/O port address OE Hex for the ASCII character code data and strobe output, and address OOH for the busy/ready-bar status input (bit 0).

RESERVING SPACE FOR A MACHINE CODE PROGRAM

There are a number of ways to reserve memory space for a machine code program, from within a BASIC program. But before details of these methods are given, we should clarify the way BASIC normally organizes memory space.

A range of addresses at the bottom of user RAM is reserved for system pointers and variables. This section is often termed the communications region'. It includes locations which store pointers to the boundaries of the various regions in upper RAM, like the 'Top of Memory' pointer, the 'Start of BASIC program' pointer and so on. The latter pointer is stored at 78A4/5 Hex (30884/5 Decimal).

Normally the BASIC program itself is stored next, in locations starting at address 7AE9 Hex. At the end of the BASIC program text, the system stores a table containing the program's variables. This is known as the 'variable list table' (VLT). This is divided into two sections: first, the simple variable table containing simple numeric variables and pointers to the simple string variables, and second - the subscripted variable table containing dimensioned variables.

As the BASIC program text changes in length, the VLT is moved up or down in memory so that it always begins from the end of the BASIC program. The pointer to the start of the VLT is stored in location 78F9/A, and the pointer to the start of the subscripted variable table in location 78FB/c.

The remaining major regions extend downward from the top of user RAM. Normally at the very top of RAM is the string storage area, extending down from the top of RAM (pointer stored at 78Bl/2) by either the default figure of 50 bytes, or a different amount established by the CLEAR N command. The BASIC interpreter's stack then extends downward in memory from the bottom of the string area (pointer stored in 78AO/1). The space between the top of the VLT region and the bottom of the stack is not used, and is designated 'free' space So that normally, the RAM organization looks like this:

METHOD 1: This method of reserving space for a machine code program involves shifting the BASIC program area upward in memory from its normal start at 7AE9, creating a space immediately above the communications region. The machine code program can then be loaded into this space, probably by POKEing it from your main BASIC program.

Needless to say, the BASIC program area can only be shifted up before your main program is loaded into it (if it were done afterwards, the start of the program would be lost). But the shifting is quite easy to do, because all that is required is A) change the 'Start of BASIC program' and 'End of Program/Start of VLT' pointers, together with B) creation of a new 'null program' at the start of the new program area.

This can be done quite easily from a small BASIC program which is fed into the computer ahead of your main program. Here is what it looks like if you want to reserve say 128 bytes:

```
10 POKE 31593,0:POKE 31594,0:POKE 31595,0
20 POKE 30884,105:POKE 30885,123
30 POKE 30969,107:POKE 30970,123
```

Here, line 10 pokes a 'null program' of 3 zero bytes into the start of the new program area (which starts at 7B69H or 31593). Line 20 pokes the decimal equivalents of the low and high bytes of this new starting address of the program area into its pointer address, while line 30 pokes in the corresponding values for the EOP/VLT pointer.

Note that this shifting program 'self-destructs' - once you run it, the BASIC interpreter loses all knowledge of its existence in memory. So if you then try to LIST or RUN, nothing will happen, because as far as the interpreter is concerned, it now has nothing in its (new) program storage area.

Once the program has run, however, any BASIC program loaded will start at the new, higher address (here, 128 bytes up), leaving the space immediately above the communications area free for a machine language routine or program.

Needless to say you can vary the above program to adjust the amount of space reserved. You'll need to change both the values poked into the pointer locations in lines 20 and 30, and the poke addresses in line 10.

Don't forget that if you use this method, the 'reserving' program will have to be loaded an ' d run ahead of the main program

every time you want to use it. The reserving operation can't be done from within the main program itself.

This is one disadvantage of this method; another is that it is not easy to load in your main BASIC program and the machine language program directly from tape.

METHOD 2: With this method of reserving space for a machine language program,, you create the required space in between the end of the main BASIC program and the start of the VLT, by shifting the VLT upward in memory.

This is simpler to achieve than method 1, because all that is required is to change the 'End of BASIC program/start of VLT' pointer stored in 78F9/A Hex (decimal 30969/70). In effect, we 'fool' the BASIC interpreter into thinking that the BASIC program is longer than it really is.

How do you work out this hew value for the EOP/VLT pointer? Probably the best way is to PEEK at the value of the pointer when your main program is loaded in normally, and then add to this figure the amount you need for your machine language routine plus a small amount (say 64 bytes) for a safety margin.

Let's say again you want to reserve 128 bytes. First load in your main basic program, then key in this command:

```
PRINT PEEK(30969) + 256*PEEK(30970)
```

The answer you get is the current value of the EOP/VLT pointer, in decimal. In other words it represent the actual end of your BASIC program. So add say 192 to this (128 plus a saftey margin), to get the new EOP/VLT pointer value.

Say the value you get is 32800. Now find the decimal equivalents of the high and low pointer bytes for this figure, by keying in this line:

```
P=32800:PRINT INT(P/256),P-(256*INT(P/256))
```

The first number you get is the pointer high byte (in this case 128), while the second is the pointer low byte (here 32). obviously if you get a different value from 32800, key this into the above line to get the corresponding values.

Now all you have to do is fit these values into a pair of POKE statements at the very start of your main BASIC program:

```
1 POKE 30969,32:POKE 30970,128
```

This line must be right at the start of your program, so that the EOP/VLT pointer is moved before the program introduces or uses any variables. Otherwise the variables would be 'lost'.

This method allows you to load save and run the BASIC program normally, without any prior preparation. once you have loaded the machine language program into the reserved space between the BASIC program and its VLT, you can also save and re-load it along with the BASIC program.Note that the 64 byte 'safety margin' allows for the small increase in program length when you add line 1 above.

Method 3: This method of reserving space for a machine language program involves changing the 'Top of Memory' (TOM) pointer so that it points to an address lower than the actual top of memory. This forces the BASIC interpreter to move its string storage area and stack downward, leaving a space for your machine language program at the top. Like Method 2, this is quite easy to do and it can be done from within your BASIC program.

First, you need to PEEK the current value of the TOM pointer. This is found quite easily by:

```
PRINT PEEK(30897) + 256*PEEK(30898)
```

ie This will give you 36863 for a basic VZ200 (53247 for a VZ200 with 16k expanded memory).

Then you simply subtract from this figure the amount of space you want to reserve for the machine language program, to

give a new TOM address. Then it's simply a matter of poking the low and high byte figures for this address into the TOM pointer, at the start of your program.

For example, say you want to reserve 256 bytes, and you have a basic VZ200 so the normal TOM is 36863. So the new artificial TOM will be 36863-256, giving 36607. To work out the two new pointer bytes in decimal type in :

T=36607:PRINT INT(T/256),T-(256*INT(T/256))

The first number you get is the pointer high byte (here, 142), while the second is the low byte (here 255). If you have a different value of TOM (for the VZ300 for example), you will get corresponding values.

Having found these values all you need do is add the following line to the start of your program:

1 POKE 30897,255:POKE 30898,142

The pointer must be changed before the program uses string variables or the stack, otherwise the system could 'crash'.

Note that this method allows your BASIC program to be loaded, saved and run normally. However it does not allow the machine language program to be loaded directly into the reserved area at the same time. The machine code must be loaded either separately, or POKED into the reserved area by the BASIC program itself after the pointer is changed.

FINDING THE TOP OF YOUR VZ200/300's MEMORY

This is somewhat more simple - type in the line:

PRINT PEEK(30897) + 256*PEEK(30898)

CALLING A MACHINE CODE ROUTINE FROM BASIC

The standard way of calling a machine language program or routine from BASIC is to use the USR(X) command. But before this command can be used, the starting address of the machine language routine must be loaded into the USR program pointer, stored at address 788E/F Hex (decimal 30862/3). This can be done using POKE statements.

As it happens, the BEEP subroutine in the VZ200/300's BASIC ROM can easily be called to do this, using the USR(X) command. The calling address for the routine is 3450 Hex, so the decimal figures for the USR pointer bytes are 80 (low byte, equal to 50 Hex) and 52 (high byte, equal to 34 Hex).

So if you want to produce a 'beep' at various places in your BASIC program, all you need to do is put this line near the start of the program (before the first beep is needed).

20 POKE 30862,80:POKE 30863,52

This sets up the USR pointer. Then, whenever a 'beep' is required in the program, simply use the command:

X=USR(X)

Note that before control is passed to the user routine at the designated address, the value of the argument variable X is stored in locations 31009/31010 (7921/2 Hex). So this can be used to 'pass' a parameter value to the user routine. If the routine doesn't need any parameters (like the 'beep' routine above), simply use a 'dummy' variable name like X, as shown.

The same general technique is used for calling other machine language routines, whether they are located in ROM or RAM. It's simply a matter of poking the start address of the routine into 30862/3, and then using the USR command.

you aren't limited to calling a single machine code routine. You can call a number of routines in turn, simply by poking each routine's start address into 30862/3 before you use the USR command to call it. Just remember to POKE the right

routine address into the pointer each time!

USEFUL ROM SUBROUTINES FOR ASSEMBLY PROGRAMMING

KEYBOARD SCANNING ROUTINE

The keyboard scanning routine resides at 2EF4 hex. This routine scans the keyboard once and returns. If a key is pressed, the A register will contain the code for that key; otherwise this register will contain zero. Registers AF, BC, DE and HL are all modified by the routine, so if the contents of these registers must be preserved they should be pushed onto the stack before the routine is called. The following example shows how the routine would be used to wait for the RETURN key to be pressed:

```
SCAN   CALL   2EF4H        ;scan keyboard once
       OR     A            ;any key pressed ?
       JR     Z,SCAN       ;back if not
       CP     0DH          ;was it RETN key ?
       JR     NZ,SCAN      ;back if not
                           ;otherwise continue
```

CHARACTER OUTPUT SUBROUTINE

A routine which outputs a single character to the video display is located at 033A Hex. The code for the character to be displayed must be in the A register, while the character will be displayed on the screen at the position corresponding to the current value of the cursor pointer. All registers are preserved. Here is how the routine is called to display the word 'HI' followed by a carriage return:

```
LD     A,4lH   ;load reg A with code
CALL   033AH   ;& display
LD     A,4lH   ;same with I
CALL   033AH
LD     A,0DH   ;now load A with CR code
CALL   033AH   ;& update screen
```

MESSAGE OUTPUT SUBROUTINE

A very useful subroutine located at 28A7 hex can display a string of character codes as a message on the screen. The string of character codes must end with a zero byte. The HL register pair must be set to the start of the string before the subroutine is called. All registers are used by the subroutine. Here is how it is used:

```
       LD     HL,MSG      ;load HL with start of string
       CALL   2BA7H       ;and call print subroutine

MSG    DEFM   'READY'     ;main message string
       DEFB   ODH          ;carriage return
       DEFB   0            ;null byte to terminate
```

COMPARE SYMBOL (EXAMINE STRING) - RST 08H

A routine which is called using the RST 08H instruction can be used to compare a character in a string pointed to by the HL register, with the value in the location following the RST 08H instruction itself. If there is a match, control is returned to the instruction 2 bytes after the RST 08H, with the HL register incremented by 1 and the next character of the string in the A register. This allows repeated calls to check for an expected sequence of characters. Note that if a match is not found, the RST 08H routine does not return from where it is called, but jumps instead to the BASIC interpreter's input phase after ' printing the 'SYNTAX ERROR' message. Here is how the routine is used to check that the string pointed to by HL register is 'A=B=C':

RST 08H
DEFB 41H
RST OBH
DEFB 3DH
RST 08H DEFB 42H RST 08H DEFB 3DH RST 08H DEFB 43H ...

;test for 'A'
;hex value of A for comparison ;must have found, so try for ;hex value of '=' ;OK so far, try for 'B'

;now look for second ;finally check for 'C'
;must have been OK, so proceed

LOAD & CHECK NEXT CHARACTER IN STRING -- RST 10H

The RST 10H instruction may be used to call a routine which loads the A register with the next character of a string pointed to by the HL register, and clears the CARRY flag if character is alphanumeric. Blanks and control codes 09H and OBH are skipped automatically. The HL register is incremented before each character is loaded, therefore on the first call the HL register should be set to point to the address BEFORE the location of the first string character to be tested. The string must be terminated by a null byte.

Here is an example of this routine in use. Note that if it is used immediately after the RST 08H instruction as shown, the HL register will automatically be incremented to point to the next character in the string:

```
RST 08H       ;test for
DEFB 3DH
RST 10H       ;fetch & check next char
JR NC,VAR     ;will go to VAR if alpha
....          ;continues if numeral
```

COMPARE DE & HL REGISTER PAIRS - RST 18H

The instruction RST 18H may be used to call a routine which compares the contents of the DE and HL register pairs. The routine uses the A register only, but will only work for unsigned or positive numbers. Upon returning, the result of the comparison will be in the status register:

```
HL    <     DE : carry set
HL    >     DE : no carry
HL    <>    DE : NZ
HL    =     DE : Z
```

Here is an example of its use. Assume the DE pair contains a number and we want to check that it falls within a certain range - say between 100 and 500 (decimal):

```
LD      HL,500      ;load HL with upper limit
RST     18H         ;& call comparison routine
JR      C,ERR       ;carry means num>500
LD      HL,100      ;now set for lower limit
RST     18H         ;& try again
JR      NC,ERR      ;no carry means num < 100
....                ;if still here, must be OK
```

SOUND DRIVER

Located at 345C hex is a routine which can be used to produce sounds via the VZ200/300's internal piezo speaker. Before calling the routine, the HL register pair must be loaded with a number representing the pitch (frequency) of the tone to be produced, while the BC register pair must be loaded with the number of cycles of the tone required (ie the duration in cycles). All registers are used. The frequency coding used is inversely proportional to frequency, ie the smaller the number loaded into the HL register pair, the higher the frequency. As a guide, the low C produced by the VZ200/300's SOUND command in BASIC can be produced using the decimal number 526, the middle C using 529 and the high C using 127. Here is how you would use the routine to get say 75 cycles of the middle C:

```
LD      HL,259  ;set frequency code
LD      BC,75   ;set number of cycles
CALL    345CH   ;& call sound routine
```

'BEEP' ROUTINE

The routine which is used by BASIC to produce the short 'beep' when a key is pressed is located at the address 3450 hex. It disturbs all registers except the HL pair. To make a beep:

```
CALL    3450H       ;make a 'beep'
```

CLEAR SCREEN

A routine located at 0lC9 hex may be used to clear the video screen, home the cursor and select display mode (0). it disturbs all registers. Again it is used by simply calling it.

```
CALL    01C9H               ;clear screen, home cursor etc.
```

PRINTER DRIVER

The printer driver routine is located at 058D hex. To send a character to the printer, load the chracter's ASCII code into the C register and call the driver - After printing, the character code will be returned in both the A and C registers. All other registers are disturbed. For example to print the letter 'A' (ASCII code 97 decimal), you would use:

```
LD C,97         ;set up code in C register
CALL 058DH      ;& call printer driver
```

A line feed character (OAH) is automatically inserted after a carriage return (ODH). If the driver is called with a null byte in the C register, it will simply check the printer status and return with bit 0 of the A register either set or cleared. The routine does check for a BREAK key depression, and if one is detected, it will return with the carry flag set.

CHECK PRINTER STATUS

A routine to check printer status is located at 05C4 hex. When called it loads the printer status (I/O port 00H) into the A register and returns. Bit 0 will be set (1) if the printer is busy, or cleared (0) if it is ready. No other registers are disturbed. An example:

```
TEST   CALL   054CH        ;check is printer ready
       BIT    0,A          ;test bit 0
       JR     NZ,TEST      ;loop if busy -continue if ready
```

SEND CR-LF TO PRINTER

A routine located at 03AE2 hex may be used to send a carriage return and line feed combination to the printer. No registers need be set up before calling, but all registers are disturbed. If the break key is pressed while printing occurs (or while the printer driver is waiting for the printer to signal 'ready'), the routine will return with the carry flag set:

```
CALL   3AE2H       ;go send CR-LF to printer
JP     C,BRK       ;check if BREAK key is pressed
...                ;apparently not
```

VZ200/300 DISK OPERATING SYSTEM (DOS) ANALYSIS

Information is included here to describe the operation and structure of the VZ200/300 DOS. The information will cover the format of the diskette, the recording technique, the DOS entry points and the structure of the DOS. It can be used to allow direct assembly language access to the DOS, and also to allow advanced programmers to enhance their DOS.

DISKETTE FORMATTING

The VZ200/300 DOS initializes the diskette into 40 tracks, with 16 sectors per track. They number from 0 to 39, track 00 being the outermost track and track 39 the innermost, The stepper motor (which moves the R/W-head arm) can position the disk arm over 80 'phases'. To move the arm from one track to the next, two phases of the stepper motor must be cycled. The DOS uses only even phases. Programmers may use this feature to generate protected disks by using odd phases or combinations of the two, provided that no two tracks are closer than two phases from one another. See the section on the disk controller I/O addresses for the control of the stepping motor.

The DOS subdivides the track into 16 sectors. It is the smallest unit of 'updatable' data on the diskette. The DOS reads or writes a sector at a time. This is to avoid using a large chunk of memory for a buffer to read or write an entire track. The DOS uses 'soft sectoring' to divide a track into 16 sectors without the use of the INDEX hole of the disk. Each sector may contain 128 bytes of data, sectors are arranged into a 2-sector interleave sequence to reduce the access time. The sequence of the sector arrangement is: 0, 11, 6, 1, 12, 7, 2, 13, 8, 3, 14, 9, 4, 15, 10, 5. Each sector is subdivided into fields. See the following diagram for the structure of a sector and a track.

RECORDING TECHNIQUE

The VZ200/300 DOS uses the recording technique of FM (frequency modulation) to write data on the diskette. In FM format, each data bit is enclosed within a bit cell. When data is read back from the diskette it takes the form of the following diagram.

As the diagram shows, the data bits (if present) are interleaved. The presence of a data bit between two clock bits represents a binary 1, the absence of a data bit between two clock bits represents a binary 0. The timing of each bit cell is shown below:

In the DOS the length of each cell is 32.2uS with the data bit appearing 13uS behind the clock bit.

Due to the low signal transfer rate, the spindle rotation speed is reduced from 300 RPM (as in other drives) to 85 RPM to keep a high recording capacity.

THE STRUCTURE OF THE DOS

The DOS is a ROM based DOS which is located in 4000H to 5FFFH. When the computer is powered up, the BASIC interpreter will jump to the DOS after initializing the BASIC pointers. The DOS will reserve a DOS vector of 310 bytes at the top of memory available. The DOS vector is pointed to by the index register IY and this vector is used to keep track of all DOS operations. Programmers should avoid modifying the IY register, otherwise the DOS will probably crash.

The DOS vectors contain the following elements:

DOSVTR = IY

NAME	BYTES	OFFSET	
FILNO	1	IY+O	FILE#
FNAM	8	IY+l	FILENAME
TYPE	2	IY+9	FILE TYPE
DK	1	IY+ll	SELECTED DRIVE# PATTERN
RQST	1	IY+12	REQUEST CODE
SOURCE	1	IY+13	SOURCE DRIVE FOR DCOPY
UBFR	2	IY+14	USER BUFFER ADDRESS
DESTIN	1	IY+16	DEST DRIVE FOR DCOPY
SCTR	1	IY+17	USER SPEC. SECTOR NUMBER
TRCK	1	IY+18	USER SPEC. TRACK NUMBER
RETRY	1	IY+19	RETRY COUNT
DTRCK	1	IY+20	CURRENT TRACK NUMBER
NSCT	1	IY+21	NEXT SCTR NUMBER
NTRK	1	IY+22	NEXT TRK NUMBER
FCB1	13	IY+23	FILE CONTROL BLOCK 1 OPEN FLAG, STATUS, FNAM, TRK#, SCTR#, ENTRY IN SCTR
FCB2	13	IY+36	FILE CONTROL BLOCK 2 OPEN FLAG, STATUS, FNAM, TRK#, SCTR#, ENTRY IN SCTR
DBFR	2	IY+49	DATA BUFFER ADDRESS
LTHCPY	1	IY+51	COPY OF LATCH
MAPADR	2	IY+52	TRACK/SECTOR MAP ADDRESS
TRKCNT	1	IY+54	TRK CNT FOR DCOPY

TRKPTR	1	IY+55	TRK PTR FOR DCOPY
PHASE	1	IY+56	STEPPER PHASE

DISK STRUCTURE

The DOS uses TRK 0, sector 0 to sector 14 as the directory. TRK 0 sector 15 is used to hold the track map of the disk with one bit corresponding to a sector used. Each directory entry contains 16 bytes. Therefore 1 sector can hold 8 entries and 1 diskette can have a maximum of 112 entries.

File type	1	byte
Delimitor (3AH)	1	byte
File name	8	byte
Start address	2	byte
End address	2	byte
Start track	1	byte
Start sector	1	byte

DOS ENTRY POINTS

A jump table to the DOS subroutines is positioned at the fixed address from 4008H to 4044H. The jump table contains the following elements:

ADDRESS	CONTENT	DOS SUBROUTINE
4008H	JP PWRON	Disk power ON
40OBH	JP PWOFF	Disk power OFF
400EH	JP ERROR	Error handling routine
4011H	JP RDMAP	Read the track map of the disk
4014H	JP CLEAR	Clear a sector of the disk
4017H	JP SVMAP	Save the track map to the disk
401AH	JP INIT	Initialize the disk
401DH	JP CSI	Command string interpreter
4020H	JP HEX	Convert ASCII to HEX
4023H	JP IDAM	Read identification address mark
4026H	JP CREATE	Create an entry in directory
4029H	JP MAP	Search for empty sector
402CH	JP SEARCH	Search for file in directory
402FH	JP FIND	Search empty space in directory
4032H	JP WRITE	Write a sector to disk
4035H	JP READ	Read a sector from disk
4038H	JP DLY	Delay mS in reg C
403BH	JP STPIN	Step in
403EH	JP STPOUT	Step out
4041H	JP DKLOAD	Load a file from disk
4044H	JP SAVEOB	Save a file to disk

DOS SUBROUTINES

PWRON

Turn ON the power of the drive selected in DOS vector IY+DK. To turn ON drive 1 , 10H should be written to IY+DK.

To turn ON drive 2, 80H should be written to IY+DK before calling PWRON.

Entry parameter: None
Exit parameter: None
Registers affected: A

PWROFF

Turn OFF the power to the disk. Both disks are turned OFF with the write request line set high at the same time.

Entry parameter: None
Exit parameter: None
Registers affected: A

ERROR

This subroutine reads the content of register A and prints the .error message before going back to BASIC.

Entry parameter: Error code in A
Exit parameter: None
Registers affected: The subroutine will re-initialize the BASIC pointers and jump to BASIC.

ERROR CODE	ERROR
0	No error
1	Syntax error
2	File already exists
3	Directory full
4	Disk write protected
5	File not open
6	Disk I/O error
7	Disk full
8	File already open
9	Sector not found
10	Checksum error
11	Unsupported device
12	File type mismatch
13	File not found
14	Disk buffer full
15	Illegal read
16	Illegal write
17	Break

RDMAP

Read the track map from the disk and place it.,into the address pointed to by IY+MAPADR.

Entry parameter: Disable interrupt
Exit parameter: Error code in A
Registers affected: A, BC, DE, IIL

CLEAR

Clear the sector specified in IY+TRCK and IY+SCTR.

Entry parameters:	Disable interrupt Track number in IY+TRCK Sector number in IY+SCTR
Exit parameter:	Error code in A
Registers affected:	A, BC, DE, HL

SVMAP

Save the track map in the address pointed by IY+MAPADR to track 0 sector 15 of the disk.

Entry parameter:	Disable interupt
Exit parameter:	Error code in A
Registers affected:	A, BC, DE, HL

INIT

Initialize a blank disk.

Entry parameter:	None
Exit parameter:	None
Registers affected:	A, BC, DE

CSI

This subroutine reads the user specified filename and puts into IY+FNAM if the syntax is correct.

Entry parameter:	Input message pointed to by HL
Exit parameter:	Error code in A
Registers affected:	A, BC, HL

HEX

This subroutine converts 4 bytes of ASCII pointed to by HL into DE reg pair.

Entry parameter:	HL points to 4 bytes of ASCII
Exit parameters:	Carry=l if error found, DE invalid Carry=O if no error, DE=2 bytes of HEX HL advanced by 4
Registers affected:	A, DE, HL

IDAM

Search for the identification address mark (IDAM) of the disk.

Entry parameters:	Desired track in IY+TRCK Desired sector in IY+SCTR Disable interrupt
Exit parameter:	Error code in A
Registers affected:	A, BC, DE, HL

CREATE

Generate an entry in the directory.

Entry parameters:	File name in IY+ENAM
	File type in IY+TYPE
	Disable interrupt
Exit parameter:	Error code in A
Registers affected:	A, BC, DE, HL

MAP

Search for an empty sector in the track map.

Entry parameter:	Track map in buffer pointed to by IY+MPADR
Exit parameters:	Error code in A
	Next sector available in IY+NSCT
	Next track available in IY + NTRK
Registers affected:	A, BC, DE, HL

SEARCH

Search for matching of filename in IY+FNAM with that in the directory.

Entry parameters:	Disable interrupt.
	File name in IY + FNAM
Exit parameter:	Error code in A
Registers affected:	A, BC, DE, HL

FIND

Search for an empty space in the directory.

Entry parameter:	Disable interrupt
Exit parameter:	Error code in A
Registers affected:	A, BC, DE, HL

WRITE

Write the content of the buffer pointed to by IY+DBFR to the track#, sector# specified by IY+TRCK and IY+SCTR.

Entry parameters:	Track number in IY+TRCK
	Sector number in TRK+SCTR
	Data to be written in buffer pointed to by IY+DBFR (128 bytes)
Exit parameter:	Error code in A.
Registers affected:	A, BC, DE, HL, BC', DE', HL'

READ

Read the content of track#, sector# specified by IY+TRCK and IY+SCTR into the buffer pointed to by IY+DBFR.

Entry parameters:	Track number in IY+TRCK
	Sector number in IY+SCTR
	Disable interrupt
Exit parameter:	Error code in A
	Read data in buffer pointed to by IY+DBFR (128 bytes)
Registers affected:	A, BC, DE, HL

DLY

Delay N mS specified by B.

Entry parameters:	Disable interrupt
	Number of mS to be delayed in B
Exit parameter:	None
Registers affected:	A, BC

STPIN

St - ep the stepper N tracks inwards specified by register B.

Entry parameters:	Disable interrupt
	Number of tracks to be stepped in B.
Exit parameter:	None
Registers affected:	A, BC

STPOUT

Step the stepper N tracks outwards specified by register B.

Entry parameters:	Disable interrupt
	Number of tracks to be stepped in B.
Exit parameter:	None
Registers affected:	A, BC

DKLOAD

Load the file specified in IY+FNAM to the memory.

Entry parameters:	Disable interrupt
	Filename in IY+FNAM
Exit parameters:	Error code in A.
	File in memory
Registers affected:	A, BC, DE, HL

SAVEOB

Save the filenamespecified in IY+FNAM and pointed to by 78A4H to the disk.

Entry parameters:	Disable interrupt
	Filename in IY+FNAM
	File start address in 78A4H
	File end address in 78F9H

	File type in IY+TYPE
Exit parameter:	Error code in A
Registers affected:	A, BC, DE, HL, BC', DE', HL'