50p

YOUR COMPUTER

DECEMBER 1981

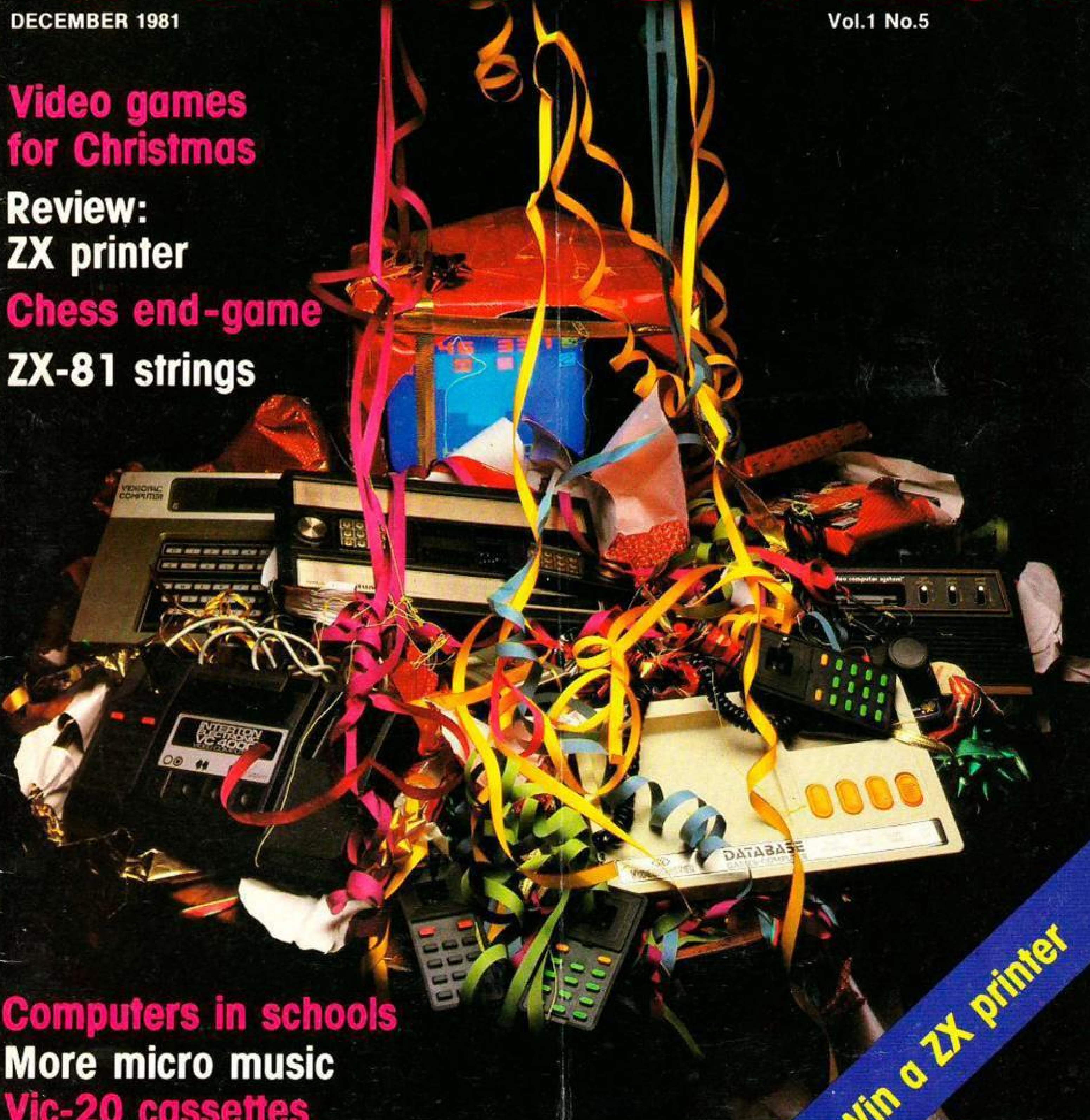
Vol.1 No.5

Video games for Christmas

Review: ZX printer

Chess end-game

ZX-81 strings



Computers in schools

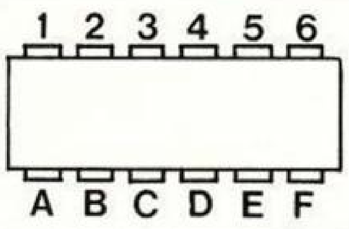
More micro music

Vic-20 cassettes

Win a ZX printer

VIC-20 CASSETTES

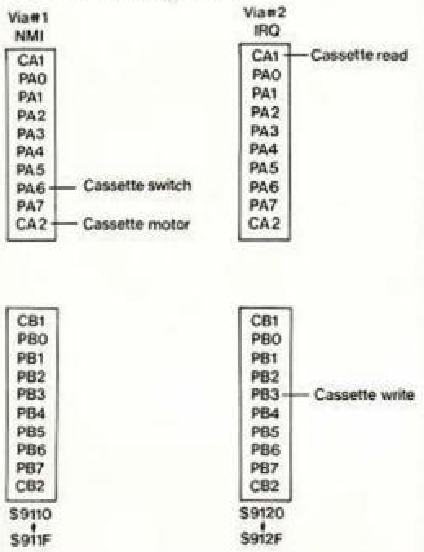
BY NICK HAMPSHIRE



Pin#	FUNCTION
A-1	GND
B-2	+5V
C-3	Cassette motor
D-4	Cassette read
E-5	Cassette write
F-6	Cassette switch

Figure 1. Connector configuration.

Figure 2. VIA assignments.



Hardware

THE VIC HAS a single, external cassette unit which is used for program and data storage. This unit is connected to the Vic by six lines — write, read, motor, sense and two power lines, ground and +5V. The connections are shown in figure 1.

The cassette is controlled by I/O lines from the two VIA (versatile interface adaptor) chips, and you can see the source of each of the cassette-control lines from the VIAs in figure 2.

The cassette-motor power-supply lines are connected to the interface chips via a three-transistor driver which is used to boost the power and voltage — it allows the motor to be driven directly. The output to the motor is an unregulated +9V at a power rating of up to 500mA. The cassette-deck motor can be turned on and off by toggling the CA2 line on 6522 # 1.

POKE 37148, PEEK (37148) AND 241 OR 14 turns the motor on:

POKE 37148, PEEK (37148) OR 12 AND NOT 2 turns it off.

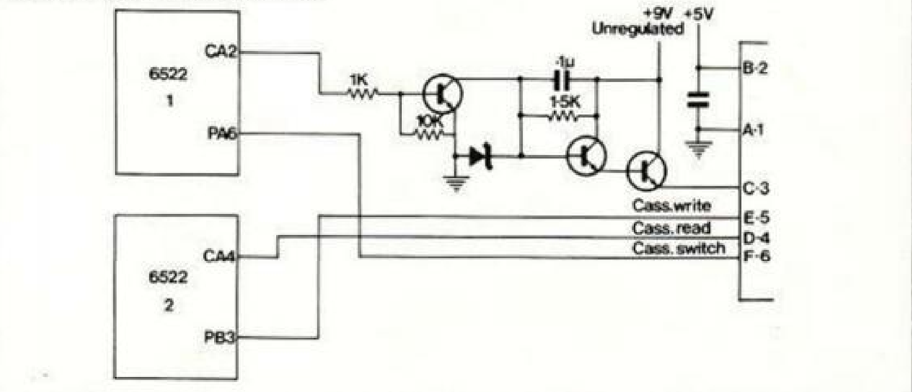
The sense-line input, line PA6 on VIA# 1, is connected to a switch on the cassette deck which senses when either the play, rewind or fast-forward buttons have been pressed. The switch is only required to sense whether or not you have pushed the play button during a read- or write-to-tape routine. This is done by a subroutine at $F8AB.

If either the rewind or fast-forward button is pressed accidentally instead of the play button, the system will be unable to tell the difference and will act as if the play button had been pressed. Because recording will start as soon as the play button has closed the sense switch, you must press the record button first in any record routine.

The cassette read line is connected to the CA1 line of VIA # 2 and the cassette write line to line PB3 of VIA # 2. During a read operation, the operating system uses the setting of the CA1 interrupt flag to detect transitions on the cassette-read line. The read and write lines are controlled entirely by the operating system — the only hardware required is signal-amplification and pulse-shaping circuitry.

These circuits are contained on a small, printed-circuit board within the cassette deck. Their function is to give correct voltage and current to the record head and to amplify the input from the read head. That gives a 5V square-wave output capable of producing an interrupt on the CA1 or CB1 lines.

Figure 3. Cassette interface circuit.



Cassette operation techniques

FOR NORMAL purposes the cassette deck is assigned the device number 1. The I/O number of the device currently in use is stored in location 186. This number, the logical file number, and the secondary address are used when saving or retrieving data files from the cassette deck.

The logical file number can be any number from 1 to 255 and is used to allow multiple files to be kept on the same device. It is of little use with cassette tape and is intended primarily for floppy-disc units. Usually the logical file number is the same as the device number and is stored in location 184.

Since it determines the operational mode of the cassette, the secondary address is important and the current one is stored in location 185. The normal default value is zero. If the secondary address is zero, the tape is Opened for a read operation. If it is set to one, it is opened for a write operation and if two, it is opened for a write, and an end-of-tape header is forced when the file is closed.

The Vic operating system is configured to allow two types of file to be stored on cassette: program files and data files. These names are however rather misleading since a program can be stored as a data file and data can be stored as a program file.

The difference between the two types is not in their application but in the way the contents of the machine's memory is recorded. Instead of program and data files, we must look on them as binary and ASCII files.

A binary file is usually used to store programs, since it is created by the operating system to store the contents of memory between a starting location and an end location. It is called a binary file because it stores on tape the binary value in each memory location within the assigned memory area.

Basic statements are stored in memory using tokens. The use of tokens means that Basic commands are not stored in the same manner as they are listed on the display or were entered from the keyboard. Instead, they are stored in memory in a partly-encoded form. Being partly encoded, a binary file is a quicker and more efficient way of storing programs. Binary files are essential when saving and loading machine-code programs.

The starting address from which a binary file will be saved is stored in locations 172 and 173. These locations are loaded by the save routine, with the memory locations at which the save will begin normally set to 0 and 4, thereby pointing to the start of the Basic text area at 1024.

They can be altered by the save routine to point to any location in memory. The end address of the area of memory to be saved is stored in locations 174 and 175. Normally, when saving a Basic program, these are set to the address of the double-zero byte which

(continued on next page)

(continued from previous page)

terminates the link address. The end address can be altered to any desired location.

To change either of these addresses one cannot use the normal save routine since it automatically initialises these locations. Instead, one must write a small machine-code initialisation routine incorporating the desired operating-system subroutines. By default, a Save command will write a binary file and a Load command will read a binary file.

ASCII files are normally used to store data but they can be used to store programs. Their format is the same as that displayed on the screen or entered from the keyboard. ASCII files are created or read almost exclusively by instructions from within a Basic program. A binary file is created or read mostly by direct instructions, though the Load and Save instructions can be used within a program.

An ASCII file must first be opened with an Open statement which specifies the logical file, device number, secondary address and file name. The operating system interprets these parameters and allows the user to read or write the file to the specified device.

Data is written to an ASCII file on a particular device with a command to Print to the specified logical file number, and data is read by a Read from the logical-file command.

Tape buffer

Whereas a binary file is loaded with the contents of successive memory locations, an ASCII file is loaded with a string of variables. Storing these would require the tape to be turned on and off repeatedly, retaining a few bytes of data at a time. The Vic overcomes this by having a 192-byte tape buffer into which all data to be written to, or read from tape is loaded. Only when this buffer is full is the tape motor turned on.

Data is stored on tape in blocks of 192 bytes and since the motor is turned on and off between blocks, a two-second interval is left between blocks to allow the motor to accelerate and decelerate. The beginning of the 192-character buffer starts at address 828; the pointer to the start of the buffer is located at addresses 178 and 179; the number of characters in a buffer is stored at location 166.

These locations can be used by the programmer to control the amount of space left in a data file. If, having opened a file on cassette, the command Poke 166,191 is executed, then the contents of the tape buffer — even if empty — are loaded on to the tape. If records are kept in multiples of 191 bytes, we can very easily keep null or partially-filled records allowing future data expansion.

Whether the file being stored is binary or ASCII, the recording method used is the same and involves an encoding method peculiar to Commodore and designed to ensure maximum reliability of recording and playback. Each byte of data or program is encoded by the operating system using pulses of three distinct audio frequencies, these are: long pulses with a frequency of 1,488Hz, medium pulses at 1,953Hz and short pulses at 2,840Hz.

All these pulses are square waves with a mark-space ratio of 1:1. One cycle of a medium frequency is 256μs. in the high state and 256μs. in the low state.

The operating system takes about 9ms. to record a byte of data consisting of the eight data bits, a word-marker bit and an odd-parity bit. The data bits are either ones or zeros and are encoded by a sequence of medium and short pulses. A one is one cycle of a medium-length pulse followed by one cycle of a short-length pulse and zero is one cycle of a short-length pulse followed by one cycle of a medium-length pulse. Each bit consists of two square-wave pulse cycles, one short and one medium with a total duration of 864μs. The wave-form timing is shown in the diagram in figure 4.

The odd-parity bit is required for error checking and is encoded like the eight data bits — using a long and short pulse. Its state is determined by the contents of the eight data bits. The word marker separates each byte of data and also signals to the operating system the beginning of each byte. The word marker is encoded as one cycle of a long pulse followed by one cycle of a medium pulse, see figure 4.

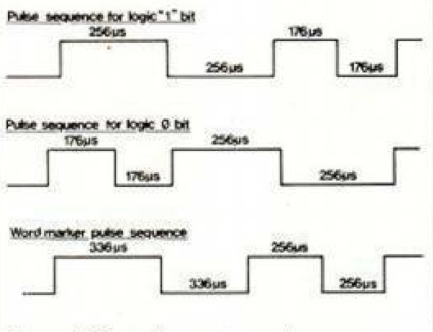

Pulse sequence for logic "1" bit
256μs
256μs
176μs
176μs
Pulse sequence for logic 0 bit
176μs
176μs
256μs
256μs
Word marker pulse sequence
336μs
336μs
256μs
256μs

Figure 4. Operating system pulse sequences.

Since a byte of data is recorded in just 8.96ms., a 192-byte block of data in an ASCII file should be recorded in slightly more than 1.7 seconds. However, timing such a recording shows that it takes 5-7 seconds. There are two causes for this discrepancy in timing. First, to reduce the possibility of audio dropouts, the data is recorded twice. Secondly, a two-second inter-record gap is left between each record of 192 bytes.

The extensive use of error-checking techniques is one reason why the tape system on the Vic is so much better than that available on most other popular computers. There are two levels of error checking. The first divides the data into blocks of eight bytes and then computes a ninth byte, the check-sum digit. The check-sum is obtained by adding the eight data bytes together; it is the least-significant byte of the result.

On reading the tape, if one bit in the eight bytes is dropped and a zero becomes a one or vice versa, the check-sum can be used to detect this error. To do this, the same procedure to calculate the check digit is performed. The result will be different to that stored in byte 9 which is the check digit of that block computed when the tape was recorded.

The second level of error checking involves recording each block of data twice. This allows errors detected by the check digit to be corrected during the second reading of the 192-byte data block. By recording the data twice, a verification can be performed by comparing the contents of the two blocks. This will detect the few errors not detected by the check-sum.

The use of pulse sequences, rather than two frequencies as in a standard FSK (frequency-shift keying) recording, has a great advantage since it allows the operating system to compensate easily for variations in recording speed. Normally, a hardware phase-locked-loop circuit would be used to lock the system on to the correct frequencies transmitted from the tape head. The Vic, however, uses software to perform this process.

Inter-record gaps

A 10-second leader is written on the tape before recording of the data or program commences. This leader has two functions: first, it allows the tape motor to reach the correct speed, and secondly, the sequence of short pulses written on the leader is used to synchronise the read-routine timing to the timing on the tape.

The operating system can thus produce a correction factor which allows a very wide variation in tape speed without affecting reading. The system timing used to perform both reading and writing is very accurate, based as it is on the crystal-controlled system clock and timer 1 and timer 2 of VIA #2. Inter-record gaps are only used in ASCII files and their function is to allow the tape motor time to decelerate after being turned off and accelerate to the correct speed when turned on prior to a block read or write.

Each inter-record gap is approximately two seconds long and is recorded as a sequence of short pulses in the same manner as the 10-second leader. There is also a gap between blocks. When the first block of 192 bytes is recorded, it is followed by a block end-marker which consists of one single, long pulse followed by more than 50 cycles of short pulses. Then the second recording of the 192 block starts.

The first record written on the tape after the 10-second leader in both ASCII and binary files is a 192-character file-header block. The file header contains the name of the file, the starting memory location, and the end location. In an ASCII file these addresses are the beginning and end of the tape buffer; in a binary file they point to the area of memory in which the program is to be stored.

The file name can be up to 128 bytes long, the length of the file name is stored in location 183, and when read is compared with the requested file name in the Load or Open command. If the name is the same, the operating system will read the file; if different, it will search for the next 10-second inter-file gap and another header block.

The file name is stored during a read or write operation in a block memory whose starting address is stored in locations 187 and 188. When the operation is completed these are reset to point to a location in the operating system. The starting location is normally set to the beginning of the user-memory area.

The starting address is pointed to by the contents of locations 172 and 173. The end address is stored in locations 174 and 175. Normally this is the highest byte of memory occupied by the program; it can, however, be altered to point to any address, providing it is greater than the start address. ■