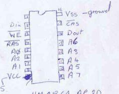
ZX99 USER MANUAL

2 K ROM

data-assette®

44, Shroton Street
London NW1
Telephone:(01) 258 0409



HM4864 AP 20

64K × 1Bit

Hitachi

CONTENTS

Appendix

INTRODUCTION TO THE ZX99

The ZX99 Tape Control Subsystem is a sophisticated extension to the Sinclair ZX81 Microcomputer, providing the following additional capabilities:

- Full software control of up to four tape cassette decks.
- The ability to use tape as a storage medium for data files, rather than just as program storage.
- Automatic tape copy.
- Diagnostic information to assist in achieving the best recording settings and maximum reliability.
- Tape block skip without destroying the contents of memory.
- Output to printers using the industry standard RS232C interface and ASCII character code.
- Automatic program listing via the RS232C printer output.

While primarily intended to equip the ZX81 with file storage, the ZX99 also provides the user with significant advantages during program development. For instance, a directory listing of the programs on a tape can be produced with a simple Basic program given later in this manual. This, coupled with the non-destructive Block Skip operation enables the user to step through to the end of recording on a tape without destroying a program that is already in memory, and then save the program when fresh tape is reached. This is much more reliable than using a tape counter, and indeed generally dispenses with the need for one, or for verbally recorded markers on the tape.

INSTALLING THE ZX99

Upon examining the ZX99 you will see that it has an edge connector that plugs into the aperture at the rear of the ZX81, connecting the two units together. At the rear of the ZX99 is a corresponding aperture to receive the extension RAM memory, if used. (While the ZX99 will function without the extension RAM Pack, the latter is a desirable accessory, as larger block sizes mean more efficient utilisation of tape.)

On the top face of the ZX99 are two 3.5mm jack sockets labelled EAR and MIC which receive the pair of cassette recorder connection leads that was supplied with your ZX81. You no longer connect these leads to the recorder, but instead plug them into the top of the ZX99 so that EAR on the ZX81 connects to EAR on the ZX99, and correspondingly MIC to MIC.

On the right hand side of the ZX99 are four jack sockets which connect to your Output cassette drives. Each drive requires a pair of connections, one 3.5mm and one 2.5mm jack. The 3.5mm connection goes to the recorder's MIC input, while the 2.5mm lead goes to its REMOTE control socket. (It is through the REMOTE input that the ZX99 is able to start and stop the tape, as required.) There is no connection to the EAR socket on the output tape decks.

On the left hand side of the ZX99 are similar pairs of sockets for the Input tape units, only this time the 3.5mm connection is made to the EAR output from the cassette recorder, and it is the MIC socket that is unused. Again the 2.5mm link goes to the recorder's REMOTE socket.

Right at the bottom on the left hand side is an extra 3.5mm jack socket. This is the output from the RS232C interface to the printer, and should not be confused with any of the other sockets. For information on how to connect up to the RS232C interface, see Appendix F.

You do not have to have all four cassette decks in order to use the ZX99, one input and one output are sufficient for many applications. In fact, when developing programs, you can derive many of the extra benefits of the ZX99 even with a single cassette recorder, such as directory listings, block skip, recording level monitoring and program listings.

On the front of your ZX99 are four LED lamps, one for each tape drive. The lamp lights when the corresponding drive is selected, giving a visual indication of which drive is currently activated.

There is no mains power connection to the ZX99, as it draws what it needs from the ZX81's power supply, which has sufficient capacity to handle the ZX81, ZX99 and 16K RAM Pack in combination.

When you have connected up all the cassette units that you intend to use, switch on mains power to all parts of the system. None of the ZX99's LEDs should light up. (If you reset the ZX81 by withdrawing its 3.5mm power supply jack and replacing it quickly, it is possible for one or more of the LEDs to light at random as the ZX99 has not been given time to reset properly. The solution is to wait a few seconds before replacing the ZX81's power entry plug.)

For normal use, engage RECORD on output decks and PLAY on input decks. No tapes should move as they will be inhibited by the REMOTE connection. If any drive does move then check that its 2.5mm REMOTE jack plug is inserted properly. (For program development you will find that it is sometimes more convenient to engage RECORD or PLAY after the drive has been selected by the ZX99. This will be discussed later.) Alternatively, if when a drive is selected and its lamp is lit it does not move, first check that RECORD (Output drives) or PLAY (Input drives) has been properly engaged, that the cassette recorder is receiving mains or battery power, as appropriate, and that the 2.5mm jack plugs are fully home at both ends of the connection.

CONTROLLING THE ZX99

The ZX99 contains its own 2K ROM (Read Only Memory) which acts as an extension to the ROM already resident in your ZX81. (See Chapter 25 of your ZX81 Manual for more information on this.) The ZX99's ROM contains the Tape Operating System, whose functions are accessed via Basic USR function calls. All of the functions can be used in program statements, or in immediate commands (i.e. both in statements with line numbers and in commands without them).

USR is the BASIC facility that enables you to enter an assembler subroutine that is outside the BASIC interpreter itself. In fact USR works as a function, which means that it must conform to the syntax rules for functions, and form all or part of an arithmetic expression. All USRs must quote a single parameter (which is actually the address in memory of the start of the USR subroutine), and they all return a single value as their 'result' on return to the BASIC interpreter. They may be invoked by a statement such as

```
    100 LET STATUS = USR 8192
```

where 8192 is the USRs entry address, and STATUS is a numeric variable that allows BASIC to 'do something' with the value returned by the USR. (Any valid name other than STATUS could of course be used.) In the above example the returned value is simply placed in STATUS (on return from the USR), where the user may do what he likes with it.

The concept returning a value stems from the conventional idea of a function such as SQR, where you give it a number, and it gives you back the square root. While the ZX99 USRs are not functions in this sense, they nevertheless make good use of the return value to pass back to you a 'Completion Code'. This tells you the outcome of the requested operation (good or bad). Use of Completion Codes will be discussed later. For the moment we can just leave it sitting in STATUS (or whatever else you wish to call it).

In order to carry out ZX99 operations you have to provide it with certain information, such as where to find the data you wish to write to tape. ZX99 expects to find the information it needs in variables with particular names. For instance, the length of the block of data that you wish to write must be placed in a variable called 'Z' before invoking the USR that writes tape. For example

```
    210 LET Z = 200
```

to write a 200-byte block on tape. Variables that are used for particular or specialised purposes like this are often referred to as 'reserved' variables, but you should note that you are free to use these variable names in any way you like when you are not actually calling the ZX99 with a USR command. In other words, existing programs using these names are in no way affected. Plugging in the ZX99 does not 'tie up' these variable names in any way. It is only when you are actually executing a ZX99 USR that it scans through your program's variables to find the ones it needs. (Actually, there are only a few of them.) If you have forgotten to define them in your program, then - you've guessed it - the ZX99 lets you know via the Completion Code that it returns.

So communication from your program to the ZX99 is done through variables with particular names. Communication back from the ZX99 to your program is through the Completion Code.

So much for the general principles of driving the ZX99 -- now to get more specific.

ZX99 COMMANDS

The USR commands fall into several groups. These are:

- Select or release a tape drive.
- Read, write or skip a block on the previously selected drive.
- Copy tape. (This USR is a bit special as it is really a whole program in itself.)
- Print a block of data or a program listing via the RS232C interface.

Appendix A summarises all the ZX99 commands, and once you are familiar with them this appendix will act as a useful programming reference.

SELECTING A DRIVE

Drive selection is carried out as a separate operation from initiating read, write or skip. This is done so that you can also select drives direct from the keyboard, when wishing to LOAD or SAVE a program, for example. Try keying in

```
    LET A = USR 8195
```

The Light-Emitting Diode (LED) for Input Drive 1 should light up, showing that you have selected this drive. Now enter

```
    LET A = USR 8204
```

The LED for Input Drive 1 should go out, and that for Output Drive 2 should come on. Notice that selection of a new drive automatically cancels any previous selection. Now try

```
    LET A = USR 8192
```

The LED for Output Drive 2 will go out, leaving all channels deselected.

USR 8192 can be used to deselect the current drive, whatever it is.

You will find the USR commands for selecting all four tape units in Appendix A. The examples above show tape selection as an immediate operation from the keyboard, but the same commands are also used as program statements when selecting one or other of the drives under program control. The only difference is that for a program statement you must of course have a line number first.

```
    110 LET A = USR 8195
```

You will see from Appendix A that USR 8207 selects both Output Drives simultaneously. This can be useful if you want to make two copies of the same data. The recordings will be identical, as they both receive the same signal from the ZX81 output. If you want to record differing information, then you must of course select one drive first and write to it, then select the other and output to that. (Parallel recording direct from the ZX81 is not normally possible as two recorders connected in parallel can interfere with each other, or load the ZX81's output circuit too heavily. The ZX99 incorporates special isolating buffers which permit this trick.)

Notice that there is no command to select both Input Drives at the same time as this is not a very useful thing to do. Two drives talking at once equals confusion! (Actually you could select both Input Drives at once by using POKE, but it could cause problems with your cassette recorders -- you have been warned.)

SAVING AND LOADING

You carry out saving and loading of programs exactly as you did before, using the ZX81's SAVE and LOAD commands. The only thing you have to remember to do first is to select the appropriate tape drive - an output unit for SAVE, or an input one for LOAD, as described in the previous section. If you forget, then SAVE will output all the information but it won't go anywhere. LOAD, on the other hand will sit there for ever, wondering why it isn't getting any input. In either case the BREAK key will put the ZX81 out of its agony, and you can then select the required channel and try again.

When SAVING and LOADING through the ZX99 you may have to adjust your recording and playback levels slightly, but as a starting point use the same settings as you did with the ZX81 on its own.

FILE STORAGE

Now that you know how to select tape drives for input or output, it is time to consider some of the basic principles of file storage on tape. First of all, why use tape for data storage anyway? As long as all the data used by a program can fit into RAM alongside the BASIC code there is no need for any auxiliary storage, but in many computer applications the objective is to perform operations on long lists of data which are far too big to fit into the available RAM memory. For instance, if you have a mailing list that you wish to process, with a hundred bytes used for each entry to store the name, address and other details, 16k bytes would only hold 160 entries, and by the time you had allowed for the work-space required by the ZX81, and of course your program, the number of records you could store would be far less than this.

In the early days of computing, when 1k bytes of main memory cost many thousands of pounds, this problem was even more pressing, but, while RAM is now cheap, its size is still limited by the addressing range of the computer. In the case of the ZX81 this is 64k, out of which must be taken the 8k ROM, the system variables, the display file and other system requirements such as the machine stack, plus any memory space occupied by any hardware add-on's that you may have fitted.

The way to escape this limitation is to hold your data on some secondary storage medium, such as tape. The trick is that you hold your data as a series of blocks on tape which can be read in by your program one at a time. So although the total data on the tape is much bigger than your available RAM, you can work your way right through it by stages. In the mailing list example you might have one tape block per customer. Then, if you wished to print a set of labels for all customers, your program would simply have to read a block, print the label for that customer, read the next block, print that label, and so on, right through the file. ('File' is the name commonly given to a set of related information, such as our mailing list, when it is stored on an auxiliary storage medium. Individual blocks of information within the file are normally referred to as 'records'. In our case we would have one record per customer.)

As supplied, your ZX81 can only use tape to SAVE and LOAD whole programs, along with any attendant variables. When a program is loaded, it overwrites everything previously in RAM, so LOAD cannot be used by a program to read data from a file on tape, as the act of loading will overwrite the program itself. What is needed is a mechanism whereby data can be read from tape into a known area within your program without affecting anything else. An area such as this is generally referred to as a 'buffer'.

INPUT/OUTPUT BUFFERS

With the ZX99, you simply use character strings as your input and output buffers. Thus for output, whether to printer or tape, you put your record together in a character string array, and then call the ZX99 when you are ready.

In order to write from or read into a buffer, the ZX99 must of course know which string array you have in mind. You will often wish to use more than one buffer in your programs, having separate ones for tape input and output, for instance. Another typical case would be when you are printing a report, and wish to keep the page heading intact in one buffer while you use a different buffer to construct the detail lines for each page.

You will recall that in ZX81 BASIC the name for a string variable consists of a single letter followed by a dollar sign, such as A$. In order to allow you full flexibility as to how many buffers you may wish to use, the ZX99 does not tie you down to using specific names for your buffers. Instead, the string variable Z$ is used as a 'signpost' to your buffer. The system works as follows.

Let us suppose that you wish to call your buffer C$, and intend to make it 250 bytes long. You declare this with a dimension statement, thus:

```
    100  DIM C$ (250)
```

(Dimension statements should appear at or close to the start of your program, as they only need to be encountered once to set up the necessary space in RAM.)

At some later point in your program you will have marshalled the data you wish to write into C$, and you now want to call the ZX99 to carry out the output operation. Just before you do this you must indicate that it is C$ from which you wish the data to be taken. This is achieved simply by loading the letter 'C' into the first byte of the 'signpost' variable string Z$. So the sequence will look something like this:

```
    100  DIM C$ (250)             Define buffer.  (Executed once
         ...                                         only.)
         ...
    500  LET Z$ = "C"             Set the 'signpost'
    510  LET STATUS = USR 8210    Write to tape from C$
```

(Don't worry about the USR code for writing tape - this and the others for reading tape and printing will be covered later.)

To see the flexibility of this approach, let us suppose that we have a program that both reads from tape and outputs to the printer, and that because we need to rearrange or expand the information before printing it we decide to use separate input and output buffers. The program would then look something like this:

```
    100  DIM T$ (250)              Declare tape input buffer
    110  DIM P$ (133)              Declare print output buffer
         ...
    500  REM --- READ BLOCK FROM TAPE ---
    510  LET Z$ = "T"              Set 'signpost' to input buffer
    520  LET STATUS = USR 8213     Read from tape into T$
         ...
         Arrange print output data in P$
         ...
    700  REM --- OUTPUT TO PRINTER ---
    710  LET Z$ = "P"              Set signpost to output buffer
    720  LET STATUS = USR 8222     Write to printer from P$
         ...
    990  GOTO 500                  Go to fetch next tape block
```

As you can see, the two Dimension statements are placed so that they are executed just once. From then on Z$, the 'signpost', is used to indicate which buffer is to be used for the ZX99 operation that follows.

In fact, the ZX99 cannot actually do anything until one of its USRs is executed (lines 520 and 720 in the example above). Remember that a USR is in fact an Asembler subroutine that is executed by the ZX81's central processor. Once a ZX99 USR for reading or writing is entered, the first thing it does is to scan through your variables until it finds Z$. Having found Z$, it extracts the first character and thus discovers the identity of the string that you wish to use for your buffer. It then searches through your variables again to find the buffer itself. If you have forgotten to assign Z$ or to dimension your buffer, or have defined either of them incorrectly, further action will be abandoned and a Completion Code will be returned to you to inform you of your error. Your program should therefore always check the Completion Code after any Read, Write or Print USR and take appropriate action if the Code is not what you expect. The example above could be extended thus:

```
         ...
    510  LET Z$ = "T"
    520  LET STATUS = USR 8213              Read tape
    530  IF STATUS=0 THEN GOTO 600          Normal condition
    540  IF STATUS=1 THEN GOTO 570          No more data
    550  PRINT "TAPE READ ERROR ";STATUS    Any other C.C.
    560  STOP

    570  PRINT "END OF TAPE"
    580  STOP

    600  ...
```

This will stop your program if an error occurs, and print out the Completion Code (returned via STATUS) so that you can inspect it. (The meanings of all the Completion Codes will be found in Appendix D.) A cruder approach would be simply to stop the program, leaving you to inspect STATUS by means of a PRINT command entered directly from the keyboard after the program has stopped. Perhaps this would be adequate while you are still developing a program. A more sophisticated solution would be to perform the error checking and handling in a subroutine. If you have several tape or print commands in your program, this will save you a lot of repetitive and wasteful coding.

To return to the use of Z$ for a moment, it is worth noting that if you have a simple program which uses only one buffer, there is nothing wrong with putting the Z$ assignment statement at the front of your program so that it is executed only once. For example:

```
 100 DIM A$ (500)
 110 LET Z$ = "A"
     ...
 500 ...
 520 LET STAT = USR 8222          Write to printer
     ...
 990 GOTO 500
```

In this case it does not even matter if the LET statement appears before or after the DIM statement - either way round will work. Taking the Z$ assignment out of the loop (from 500 to 990) will make the program run a little faster as statement 110 is no longer executed every time around, but if your are going to use more than one buffer, then of course you must keep reassigning Z$ as necessary.

It is worth emphasising one other point about defining buffers. Notice that the buffer has to be declared in a DIM statement as a 'string array with no dimensions'. (This may sound like a contradiction in terms, but it is explained in Chapter 22 of your ZX81 Basic Programming manual in Exercise 2, page 145.) Buffers must be dimensioned this way because input buffers have to be brought into existence and provided with RAM before the ZX99 starts to read data into them. Ordinary string variables expand and contract depending on their current contents, but when the ZX99 is reading from tape there is no time to carry out all the shifting about that this involves. So advantage is taken of the fact that dimensionless string arrays always keep their defined length, whatever their current contents. Since input buffers must be defined in this way, output buffers are treated the same for consistency so that a single buffer can be used both for input and output, if desired. (Z$ itself is simply an ordinary string variable, not an array, as this takes up a little less memory space.)

Although the length of a buffer is thus fixed, it is advantageous to be able to vary the amount of data written out of it. Print lines may need to be different lengths, for example. Also, on input the size of the block actually read from tape may not be the same as the size of the buffer into which you are trying to read it. We therefore must have some means of indicating the number of characters that are actually transferred, so this brings us on to our next topic.

SPECIFYING DATA BLOCK LENGTH

When a buffer is declared:

```
    100  DIM A$ (500)
```

the quoted size represents the maximum number of bytes of data that may be transferred into or out of the buffer. To specify the actual number of bytes that are to be transferred in a particular operation, we use another reserved variable, this time a simple numeric variable, Z (surprise, surprise). When you wish to output a block to tape or printer you must first load Z with the number of bytes that you wish to transmit, starting always with the first byte of the buffer. For example:

```
    100  DIM P$ (133)
         ...
         ...
    650  LET P$ = "THIS IS AN EXAMPLE"
    660  LET Z = 18                     Length (including spaces)
    710  LET Z$ = "P"
    720  LET STATUS = USR 8222          Write to printer
         ...
```

Although the buffer size in the above example is 133 bytes, just 18 characters will be output to the printer.

When reading from tape, the ZX99 places the size of the block it has read into Z. Since the ZX99 USRs cannot add variables to the list, you must include some reference to Z before performing a tape read, in order to ensure that it is available for the ZX99 to write into it. This can be done simply by a LET statement that assigns any value to Z before your first tape read statement:

```
    100  DIM T$ (500)
    110  LET Z = 0                 'Create' Z in variable list
         ...
    510  LET Z$ = "T"
    520  LET STAT = USR 8213                     Read tape
    530  IF STAT<>0 THEN STOP                    Error trap
    540  REM - Z NOW CONTAINS BLOCK LENGTH
```

If the size of the incoming block is bigger than the buffer you have provided, then Z will be set equal to the full size of the buffer (as appears in the DIM statement), and a Completion Code will be returned that indicates that there was more data than would fit into the buffer. (The excess data will be lost.)

Since Z is used in all input/output transactions, you should copy its contents to some other variable if you need to preserve the data block length for later use. Remember that the next input/output operation will overwrite the current contents of Z.

TAPE INPUT AND OUTPUT

The previous Chapter introduced the ideas of tape files and input/output buffers. Let us return again for the moment to some general principles.

When processing tape it is not feasible to write back into the middle of a previously recorded tape, as with the ZX81 recording format the space required by data varies with its content. In fact, on most mainframe computer systems you cannot write back into the middle of a previously recorded tape. (There are some types of tape drive that permit it, but they use special extra tracks on the tape with pre-recorded addressing information which takes up space, and such systems are in the minority.)

So the first principle to observe is that during the processing of a particular tape it will be used either as an input tape, or as an output one, and will not change roles halfway through. The division of the drives that the ZX99 can control into Inputs and Outputs is not therefore a limitation in practice.

This ties in with another very important principle of tape storage, and that is the need to always have 'back-up'. Tapes will wear after much use, or may become damaged by accident. Even if your tape is good, your program might not be, or you might run a job and find out afterwards that you had used the wrong data. This may sound excessively pessimistic, but even computers cannot correct for human error in this case! What you need is an insurance policy. Read on.

Files of data need to be altered from time to time. Perhaps you wish to add new names and addresses to your mailing list, and delete entries for people who are no longer to be included. In order to assist when searching for entries, you would probably keep your records in alphabetical order, so new names would need to be slotted into their correct positions in the file, rather than just be tacked on the end. Even if writing back into previously recorded tape were feasible, you can see that it would still be impossible to 'open up' gaps to accept new entries. So the approach to updating a file is not to write on to the old tape, but to create a new tape, copying unchanged material from the old one, but incorporating the required changes as you go.

To summarise, you have a tape that contains your mailing list - the current master copy of the file. In order to amend this from time to time you need to write an updating program. This knows enough about the data on your file to read in records and write them out again, but also allows you to key in additions and deletions through the keyboard. It could proceed one record at a time, or preferably work through the file automatically until it finds the appropriate place for the next alteration that you wish to make. This would be done by comparing your [illegible] name with the name in each record as it reads them in. Records are simply transcribed to the output tape until the next input record has a name that is 'after' the one you wish to insert, or matches the one you wish to delete.

After an updating session you will have two tapes - a new one that becomes your new master tape, and the original one, generally referred to as the 'old master' (nothing to-do with paintings, though). Don't discard the old master just yet! You cannot be sure that your new master is good. There might have been a fault in the recording, or you may find that you have inadvertently deleted your most important customer from the list! So you hang on to your old master, and discover that the principle of always creating a new tape whenever you modify the file has solved the back-up problem. In fact mainframe computer installations often keep three generations of any master file, known as the 'grandfather', 'father' and 'son' levels of the file.

If you find that the latest level of a file is bad, or becomes damaged, say, you then have to go back to the previous generation and reapply the last set of changes. If this involves a lot of work, or if you use the file very much between updates, for printing off labels, for example, then it is probably a good idea to make a duplicate copy of the latest level. The ZX99 can help you here, with its automatic tape copy. (This is covered in the next Chapter.)

Now to look in detail at the commands for reading and writing tape.

WRITING TAPE

The command to write tape is simply of the form:

```
        LET CCODE = USR 8210
```

where 8210 is the entry address of the USR that performs ZX99 tape output. We have already seen in the previous Chapter that there are various actions that must be carried out before this command can be given. The complete sequence of events is:

1) Define an output buffer by means of a DIM statement, e.g.:

```
        100  DIM W$ (300)
```

2) Load the data to be written into the buffer, arranged as required,

3) Set Z equal to the number of characters that you wish to write, e.g.:

```
        280  LET Z = 240
```

4) 'Point' to the buffer via Z$, e.g.:

```
        300  LET Z$ = "W"
```

5) Select the appropriate tape drive, e.g.:

```
        310  LET CC = USR 8201        (Selects output drive 1)
```

6) Issue the 'Write' command itself, e.g.:

```
320 LET CC = USR 8210
```

7) Return the ZX81 to SLOW mode, if required, e.g.:

```
330 SLOW
```

8) Analyse the Completion code and take appropriate action, e.g.:

```
340 IF CC<>0 THEN STOP              Trap errors
```

You will notice two extra steps in the above sequence that were omitted in Chapter 3 for the sake of simplicity. In step (5) we select the output tape drive that we wish to use. This should be left to the last possible moment in your program (i.e. right before the 'Write' USR itself), because the tape drive will start to move as soon as this command is executed.

The ZX99 automatically releases all tape drives at the end of the Write operation, so there is no need for you to include a separate command in your program to effect this.

Tape input/output uses the ZX81's own recording circuitry, which commandeers the video output to take the tape output signal (which is why you get the various striped patterns on your T.V. screen when reading or writing tape). This has to take place in FAST mode, so USR 8210 forces the system into this condition, and leaves it that way when returning to the ZX81, so if you wish your program to revert to SLOW mode you must include the appropriate command, as in step (7). If you wish your program to run at maximum speed, do nothing and leave it in FAST mode.

READING TAPE

The command to read tape is of the form:

```
LET CCODE = USR 8213
```

The complete sequence of operations for a Read is:

1) Define an output buffer by means of a DIM statement, e.g.:

```
110 DIM R$ (400)
```

2) 'Create' Z as a variable by any LET statement, e.g.:

```
120 LET Z = 0
```

3) 'Point' to the buffer via Z$, e.g.:

```
400 LET Z$ = "R"
```

4) Select the appropriate tape drive, e.g.:

```
        410  LET CC = USR 8195          (Select input drive 1)
```

5) Issue the 'Read' command itself, e.g.:

```
        420  LET CC = USR 8213
```

6) Return the ZX81 to SLOW mode, if required, e.g.:

```
        430  SLOW
```

7) Analyse the Completion Code and take appropriate action, e.g.:

```
        440  IF CC=1 THEN GOTO 1000        Test for end of file
        450  IF CC<>0 THEN STOP            Trap errors
```

8) Preserve the count of input bytes if Z is likely to be reused before you have finished with the count, e.g.:

```
        460  LET INLEN = Z
```

9) And finally - do whatever it is that you are going to do with the data that you have read in.

Again, you will notice some extra steps that were omitted in Chapter 3 for simplicity, namely (4) to select the required drive and (6) to return to SLOW mode (optional). The comments concerning these operations that were made in the previous section ('Writing Tape') also apply here.

As with Tape Write, the ZX99 automatically releases all drives at the end of a Read operation.

While the sequence of events for writing and reading is very similar, it is worth noting what the differences are. Obviously when writing you must prepare your data first, but when reading you process the data after the Read operation. On output, you must set Z to the block length before initiating the Write. On input, although you do not have to load Z with a meaningful value, you must nevertheless make sure that it exists among your variables by a dummy LET statement, as per step (2) above. After the Read, the ZX99 places the length of the input block in Z.

When checking the Completion Code after a Read, you should remember to look for the end-of-file condition (C.C.=1). On writing tape this does not apply as there is no way of testing whether you have hit the end of an output tape (or even whether you have remembered to place a tape in the output drive for that matter).

BLOCK SKIP

This is a very useful instruction, especially when entered direct from the keyboard. In essence, it performs exactly the same function as a Tape Read, but it does not store the data in RAM. It does however check the data as it scans through it, and returns a Completion Code that will indicate if any Read errors were encountered. It can thus be used for checking the validity of tape data without destroying the contents of RAM, and is therefore very useful during program development, as it allows you to determine whether you have SAVEd a program successfully without destroying the original that you have so painstakingly built up in RAM.

Since the Block Skip does not store any information in RAM it does not require a buffer, and is therefore much simpler to use that tape Read or Write. The sequence is simply:

1) Select the appropriate (input) tape drive, e.g.:

```
LET CC = USR 8195
```

2) Issue the Block Skip command, e.g.:

```
LET CC = USR 8216
```

The examples above show the commands as entered direct from the keyboard (i.e. without line numbers) but they could just as easily be statements that are part of a program.

As with Read and Write, the ZX99 automatically releases all tape drives at the end of a Block Skip operation.

If you wish to use Block Skip as an immediate command direct from the keyboard, then press STOP on your cassette recorder before entering the drive selection command (step (1) above). If you do not do this the tape will start to move straight away before you have had time to enter the Block Skip command. So first put your tape drive into the stopped condition. Then enter your drive select command, followed by the Block Skip command. Once you see the dashed pattern on your T.V. screen that indicates that the ZX99 is searching for data, press PLAY on your cassette recorder. In this way you will always scan the block right from the beginning, which is important if you are checking it for possible errors.

Another use for Block Skip is to step through a library tape on which you have saved several programs. This avoids the need to use a tape counter, which is often a not too accurate device.

SETTING UP THE CASSETTE RECORDERS

When you wish to run a program that uses tape input/output, first ensure that your tapes are fully rewound. The quickest way to rewind when a drive is deselected is to pull the 2.5mm jack plug out of the REMOTE socket at the recorder, and then press REWIND. Note that pulling the jack plug out at the ZX99 end of the cable will not have the desired effect of allowing the recorder's drive motor to operate. If you do pull out the jack plug for this or any other purpose, be sure to hold the body of the plug. Never pull on the cable, as sooner or later this will lead to a break in the wire and much frustration. (If you suspect that one of your cables has broken, the quickest way to check is to swap it for another cable and see if this clears the hang-up.)

Alternatively, instead of pulling out the plug, you can select the drive direct from the keyboard and than carry out the REWIND operation.

Once your tapes are all rewound, deselect all drives, using USR 8192 via the keyboard if necessary.

Now engage PLAY on the Input drives, and RECORD on the Output drives. Nothing should move, as the drives are not yet selected by the ZX99. If anything does move, then the REMOTE lead is not properly inserted into the offending cassette recorder.

Everything is now ready, and when you RUN your program the drives will respond correctly when selected.

AUTOMATIC TAPE COPY

This command is in reality a complete program in itself. Since the ZX99 uses the same tape data format as the ZX81 itself, this facility can be used to copy either program tapes or data files. Moreover, it can be used to make two copy tapes at a time if two output drives are available.

To use the automatic tape copy, first deselect all drives, using USR 8192 if necessary. Then mount the tape to be copied on Input Drive 1, fully rewound or positioned at the point from which you wish to start copying. (If you wish to copy the third program on a tape, for instance, you can use Block Skip to step over the first two and leave you correctly positioned to copy the third.)

Now mount a blank tape on Output Drive 1, and a second one on Output Drive 2 if you wish to make two copies simultaneously.

Engage PLAY on the Input Drive, and RECORD on both output drives. (If you are only making one copy, ignore references to the second drive.) Nothing should move yet. Now key in:

```
    LET A = USR 8219
```

The ZX99 will first write a blank leader to both output drives for five seconds, to ensure that recording starts on good tape rather than on the transparent header tape. Then it will switch to the Input Drive and start searching for the first block. When it finds this it will read it into RAM.

When the end of the first input block is reached, the ZX99 will switch back to the output drives and write blank tape for five seconds to provide the necessary gap between records. The data then follows, written out from RAM (not copied directly while the input is being read as this is impossible with the ZX81 and, indeed, with all conventional computer systems).

The ZX99 then returns to the Input Drive to acquire the next block of data and the process is repeated until the data is exhausted, the BREAK key is pressed, or a tape error is encountered while reading the input.

End of data is assumed when the Input Drive is selected for twenty seconds without sensing any data.

In order to copy the maximum possible block size, the Automatic Tape Copy takes over the whole of RAM, obliterating the system variables, program space, display file and everything else. Consequently, when copying is complete, it has to bring the ZX81 back up with a cold start, as though you had disconnected its power lead and then reconnected it again. For this reason the Tape Copy cannot return you a Completion Code to indicate the nature of any errors, if such were encountered, but the only possible causes for the Copy to terminate are:

1) End of data, as defined above.

2) Tape Read error.

3) Block too big for available RAM.

4) Use of BREAK key to cancel the operation.

In view of reason (3) for termination, always ensure that you are not trying to copy a data block or program that was created on a ZX81 with more RAM memory than you have available.

TAPE PROCESSING GUIDELINES

This Chapter provides hints and advice on how to get the best results from your cassette recorders, and on useful techniques that you can employ in your programs.

TAPE RECORDING LEVEL ADJUSTMENT

Tape drives on mainframe computers achieve consistent recording levels by saturating the magnetic field recorded on the tape. With audio tape recorders, as used with the ZX81, saturation means distortion of the sound, so they are not designed to saturate the tape. Because of this, the ZX81 uses short bursts of a fixed tone to represent each binary digit stored on tape. With this technique there is no established absolute reference level for the recording, and users sometimes experience difficulty in achieving consistent results.

With many recorders it is difficult to find a single setting of the VOLUME and TONE controls that is satisfactory for both record and playback operations. In fact this can well be because the best levels for the two functions are different. With a single tape recorder, constant readjustment when swapping between record and replay is inconvenient, to say the least. With the ZX99 this problem disappears, as you keep one unit permanently assigned to recording, and use a separate one for playback. Both drives can therefore be adjusted to the best settings for their respective roles and not altered thereafter.

In order to help with tape recorder adjustment, the ZX99 can be used to create a 'commissioning tape'. With the ZX81 you can only record whole programs, which are inevitably a mixture of information, but with the ZX99 it is possible to write blocks of data that are consistent patterns of all zero bits or all one bits, making it much easier to examine the effect of your settings. The following program will write alternate blocks of zeroes and ones on a tape mounted on Output Drive 1:

```
 100  REM ** TEST TAPE GENERATOR **
 110  LET  A = USR 8201
 120  PAUSE 250
 130  LET  A = USR 8192                 Write blank leader on tape
 150  CLEAR
 200  LET Z$ = "X"                      Define buffer
 210  LET Z = 500                         --     --
 220  DIM X$(Z)                           --     --
 250  FAST
 300  FOR N = 1 TO 4                    Write four pairs of blocks
 310  FOR A = 1 TO Z
 320  LET X$(A) = CHR$ 0                Fill buffer with all zeroes
 330  NEXT A
 350  LET A = USR 8201
```

```
360  LET A = USR 8210                  Write 'zeroes' block
510  FOR A = 1 TO Z
520  LET X$(A) = CHR$ 255              Fill buffer with all ones
530  NEXT A
550  LET A = USR 8201
560  LET A = USR 8210                  Write 'ones' block
600  NEXT N
650  SLOW
660  CLS
700  LET A = USR 8201
710  PAUSE 1500                        Write blank trailer on tape
720  LET A = USR 8192
810  STOP
```

(If you only have the basic 1K RAM memory you will have to reduce Z, the buffer size.)

Having entered the program, you now need to set up your output cassette recorder for a trial run. The volume should be as high as possible, short of producing distortion, so as to record the strongest possible signal on the tape. If your system has a recording level meter then use this, but if not, try setting the Volume at or close to maximum, and set the Tone control about halfway through its range. (This will help to minimise any noise above the frequency used to represent the data.)

Make sure that no drives are selected (use USR 8192 to clear them), and engage RECORD on the cassette drive. There should be no tape movement yet. Now RUN the program. After writing a length of blank leader, the tape will halt and the screen will remain grey for a while. Do not worry, the ZX81 is filling the buffer with all zero bits. When this has been done, a block of zeroes will be written on to tape, after which there will be another delay while the buffer is filled with one bits. A block of ones will then be written, and the whole process repeated until four pairs of blocks have been written on to tape. (You can alter the number of blocks by changing line 300, or just write one block of each type if you prefer by deleting lines 300 and 1490.)

When the job is complete, rewind the tape, then transfer it to Input Drive 1. Now enter the following program, which can be held in memory together with the previous program providing you have enough RAM.

```
2000 REM ** PLAYBACK TEST **
2010 CLEAR
2020 LET Z$ = "X"
2030 LET Z = 510
2040 DIM X$(Z)
2100 LET A = USR 8195
2110 LET A = USR 8216                  Block skip
2120 SLOW
2130 IF A=1 THEN STOP
2140 PRINT "COMPLETION CODE = ";A
2150 PAUSE 250
2160 GOTO 2100
```

Engage PLAY on Input Drive 1, and set both the Volume and Tone controls to their mid-points. Enter RUN 2000 to start the playback program, and observe the pretty patterns on the television screen as it runs.

First of all you will see a pattern of dashes as the ZX99 scans through the blank leader tape looking for the first block. This should look relatively clean, without a lot of random black streaks or flecks. If a lot of 'mush' is visible then you probably need to turn the tone control towards minimum to filter it out.

When the first data block is reached you should see a series of horizontal stripes, alternately black and flecked grey, of about equal thickness. The black bands are actually the tone-bursts for zero bits, and the grey stripes are gaps to separate them. (If the pattern tends to jump a lot you may be able to improve its stability by adjusting the tuning on your television set.) If the black bands are broken by white flecks, or if the whole screen is a mass of grey flecks then the incoming signal is too weak, so turn up the Volume on the Input drive. If the black bands appear much broader than the grey ones, either permanently or intermittently, then noise in the gaps is being picked up as extensions of the tone burst, so try turning down the Volume and/or Tone controls until a good steady result is obtained.

At the end of the block a Completion Code will be displayed on the screen for a few seconds. More about this later.

An inter-record gap will follow which will look like the pattern seen for the initial tape lead-in. You may see a short burst of noise during this, which will mark the point where the tape recorder stopped then re-started when making the tape. On some recorders it may be a sizeable fraction of a second, but should be ignored by the ZX99.

The gap will last about five seconds and will be followed by the block of all one-bits. This differs from the zero-bits block in that the black bands are about twice as broad as the grey gaps. It is the size of the band that distinguishes between a zero and a one bit on the tape. Again, breaks in the black band indicate too weak a signal while too much black means too strong a signal, or too much high frequency noise.

The whole sequence will repeat as further blocks are read, giving you the opportunity to try further adjustments to the cassette recorder's controls. The aim is to get the inter-record gaps as clean as possible without weakening the data block signals too much, and solid clearly defined black bands of the appropriate widths when reading data. If you cannot get the inter-record gaps as clean as you would like, then try re-recording the tape with either the Tone or Volume setting on the Output drive turned down a little. Preferably, use a different tape so that you can compare the results at various input drive settings.

CHECKING THE CONNECTIONS

What if on playback you do not seem to be getting any signal at all, or a very weak or intermittent one, even with Tone and Volume turned full up on your Input drive? Disconnect the input recorder from the ZX99 (pull out the jack plugs at the recorder when doing this), then play the tape again listening to the audible output that you will now be able to hear. The initial lead-in part of the tape may be silent, or you may hear a high-pitched tone, depending on the settings on your Output drive when you recorded the tape. After about ten seconds you will reach the first data block which should sound as a fairly loud medium-pitched tone -- round about note G in the middle of a piano keyboard, if you have access to one.

If you can hear this tone strongly then the problem must lie with your input cabling, so check all connections and also try swapping the cable for another one. Pay particular attention to the cable from the ZX81 that goes into the top of the ZX99; the one that was supplied with your ZX81. The tips of the jack plugs on this cable are a little narrower than on many other 3.5mm jack plugs and sometimes make a poor connection. Try pulling the connector out of the socket by a small amount.

If on the other hand you are unable to hear the tone when playing the tape as described above, then the problem occurred while recording the tape. Check Volume and Tone settings, and that 'Record' was engaged when making the tape, not just 'Play'. Then check all connections, again paying particular attention to the connections between the ZX99 and ZX81. The recording signal put out by the ZX81 is at an extremely low level, as it simulates the output from a microphone which is very small. Good connections are therefore of the utmost importance.

If you find that your tape appears to give a good signal to start with, but then comes to a weak patch, indicated by loss of consistent black bands on the screen, or premature stopping of the tape before the true end of the block, you may have an intermittent connection, but should also consider whether your tape is getting old and worn. If it is, discard it. You should also check the state of your tape unit recording heads from time to time, and if there is any build up of oxide on them use a proprietary cleaner. If this is a frequent problem, use a better brand of tape or have the tape transport mechanism checked.

COMPATIBILITY WITH OTHER TAPES

Because the ZX99 incorporates special buffer circuits to isolate the two output drives from one another, it tends to write a stronger signal than the unaided ZX81 at the same Volume and Tone control settings. You may therefore find that tapes recorded previously with the ZX81 alone require slightly different playback settings from those required for ZX99 tapes.

TAPE FAULT COMPLETION CODES

While reading the commissioning tape (above), the playback program will display a Completion Code after each read operation. This tells you whether the ZX99 detected any faults during the tape read operation. The last Completion Code (C.C.) should be a 1, indicating that the ZX99 could not find any more data within 20 seconds, which is taken to signal the end of recorded data. If all preceding C.C.s are zero then you have no problems -- all data was read cleanly. If not -- read on.

Completion Codes 16 thru 22 indicate faults detected in reading the data back from tape. If you have any C.C.s other than these, details are given in Appendix D. C.C. 15 is caused by use of the BREAK key while reading, and C.C. 14 is due to reading a record that is longer than the buffer into which you are trying to read the record. The other C.C.s are caused by various programming errors such as missing definitions for reserved variables, so check that you have keyed in the Playback Test program correctly.

To analyse the tape fault codes you need to understand a little about how the ZX81 and ZX99 interpret the information that is read from tape. You will recall that zero bits are displayed on the screen as black bands of a certain width, while one bits are represented by broader black bands. The ZX81 measures the width of each band as it reads it and decides whether it is a zero or a one. If the tape unit controls are set correctly, the band widths will be consistent and within quite narrow limits. When LOADing a program, the ZX81 simply sees whether each bit is nearer to the expected width for a one or a zero regardless of how close it is to the ideal. (The ZX81 does reject any very narrow bands as being just noise.)

The ZX99 uses the same principle when performing Block Read (USR 8213) or Block Skip (USR 8216), but in addition makes a judgement as to how close each bit gets to the target width for a one or a zero. It establishes two 'windows' within which bits will be accepted as good zeroes or good ones. Bits that fall between the two windows, or outside them on either side, are rejected as being too suspect to represent good data. We thus get three categories of 'bad bit'. First, bits that are too weak to be considered as good zeroes (but too strong to be ignored as noise). These are referred to as 'Fault A' bits. Second are bits that fall somewhere in between good zeroes and good ones, referred to as 'Fault B' bits. . Finally there are bits where the width of the band is so broad that it is too big for a good one bit (Fault C).

The C.C.s 16 thru 22 indicate various combinations of Faults A, B and C, as shown by the following table:

Completion Code	Fault A	Fault B	Fault C
16	YES	NO	NO
17	NO	YES	NO
18	YES	YES	NO
19	NO	NO	YES
20	YES	NO	YES
21	NO	YES	YES
22	YES	YES	YES

You can interpret these C.C.s to help you adjust the Volume and Tone controls. Any code indicating Fault C (C.C.s 19, 20, 21 or 22) generally means that the playback Volume and/or Tone control is set too high. Likewise, Fault B (C.C.s 17, 18, 21 or 22) when reading a zeroes block is again an indication of too strong a signal. On the other hand, if Fault B is reported when reading an all-ones block then in this case it is a warning that the signal is too weak. Similarly, Fault A on a zeroes block (C.C. 16) probably indicates a weak signal. However it must be noted that Fault A can also be caused by excessive noise, and you should always consider the Completion Code in conjunction with the condition of the patterns being displayed on the television screen.

ADDITIONAL SECURITY

Magnetic media are not infallible; even the most sophisticated disk systems used on mainframes can suffer from the odd flaw in the recording surface. So your use of tapes must take this into account. The mose elementary precaution is to MAKE BACK-UP COPIES of all tapes whose loss would cause you significant grief. But there are also other techniques that you might wish to consider in your applications.

One way of getting over the occasional bad patch on a tape is to record all data twice. So if the first block cannot be read correctly, the duplicate can be used instead. Obviously this halves the effective capacity of the tape and doubles the writing and reading times, but if you have a job where the computing time is much greater than the tape read/write times this may be a good solution, as it will save you having to re-run the whole job from scratch using your back-up tape. (You did make one, didn't you?) If you use this technique, record the duplicate as a separate block on tape, as if you make two copies of some data in a single block you cannot tell whereabouts in the block any indicated fault occurred. Also include some number or other indication in each block to show whether it is the original or duplicate block, so that you can make sure that you pick up the sequence correctly after a fault.

You can use the block length returned in Z to check that you have read all the data that you expected. If your data varies in length from block to block you can include a number at the front of each block to indicate its length. Don't forget that when you read the block back the length returned in Z will include the characters taken up by the length count you have inserted. (Since your buffer must be a string array, use STR$ and VAL for converting numeric values to and from strings for inclusion in your data blocks.) When checking block lengths, a count that is well short of the expected value probably means that a weak patch on tape has given an apparent end-of-block indication before the true end. So check the size of the next block read to see whether it is just the residue of the previous one.

Another technique used on many commercial computer systems is the use of a 'check-sum'. Here, the values of every character (as given by the CODE function on the ZX81) are added together to give a sum that is stored at the back end of the block. When the block is read back the sum can be re-calculated and checked against the value recovered from the block. On mainframe computers the check-sum (or its equivalent) is normally calculated by purpose-built hardware as the block is read or written, and so does not result in any apparent computational overhead. With the ZX81 you would have to do the necessary arithmetic in software, so the benefit must be weighed against the time the calculations will take.

LEADERS, TRAILERS AND INTER-RECORD GAPS

The ZX99 always records a gap of five seconds between data blocks, to correspond to the gaps between programs SAVEd by the ZX81. At the start of a tape, this may not be quite enough to ensure that you are off the transparent leader tape, so it is a good idea to run some extra lead-in, another five seconds, say, before writing your first block. This can be done simply by using PAUSE, e.g.:

```
    100  CLS                      Clear screen
    110  LET A = USR 8201         Select required output drive
    120  PAUSE 250                Let it run for five seconds
    130  LET A = USR 8192         Deselect the drive
```

(If your mains frequency is 60 cycles you will require PAUSE 300).

Note the use of CLS, to clear the screen, as the tape output uses the same hardware inside the ZX81 as the video display. (That is why you 'see' the tape output on the television screen.) If you don't have a clean screen you will get bits of rubbish in your inter-record gap. You can omit the CLS if you know that the screen is going to be blank when you issue the PAUSE, for instance immediately after you start to RUN a program.

At the end of a tape, the ZX99 will stop after the last block. If it is a new tape this would be all right, but once a tape has been used before, it is probable that there will be old previously recorded data on the tape at that point. So you should finish off a tape by writing a long blank trailer of at least 25 seconds, to ensure that when you read it back it will cause a time-out to signal the end of the tape. This can be done by using code similar to that shown above for creating the blank leader, but with a larger PAUSE. For mains frequency of 50 cycles this will require a PAUSE count of 1250 (or 1500 at 60 cycles).

GROUPING RECORDS IN BLOCKS

As mentioned above, the gap between records lasts five seconds. It is therefore uneconomical to write very small records as you could wind up with more gap than data on your tape. Your application may only need a small amount of data in each record, so what do you do? The answer is to group several small records together and write them out to tape as a single block. When this is done, the little records are referred to as 'logical records' in the trade, while the big block that groups them together is known as a 'physical record'.

Grouping records together like this means that your program will have to be a little more complicated, as you will have to process the same format of record when it is located in several different places in memory. One way of achieving this is to move each logical record into a separate 'working buffer' that is just big enough to hold a single logical record, process it there, and then move it back to its place in the physical record before writing out the updated block. If you are inserting new logical records, you will need separate input and output buffers for the tape blocks (physical records), for as soon as you insert or delete a logical record their positions all get out of step when comparing the old and new physical records. However, it is worth the additional complication if you wish to store the maximum amount of data and process it at the highest speed.

THE RS232C PRINTER INTERFACE

A full RS232C interface provides for two-way communication, plus extra circuitry to control a 'modem' (for connection to the telephone network). Since a printer only needs to receive signals, and does not need a modem, the ZX99 does not implement a full RS232C interface, but provides the basic 'Transmitted Data' circuit. (See Appendix F for details of the physical connections.) This output enables your ZX81 to drive most printers that use this interface and accept ASCII character codes (see Appendix E).

The ZX99 does not use the BASIC commands LPRINT and LLIST as these are already assigned to the ZX81 mini-printer, but it provides you with equivalent functions in the form of USR subroutines. You are not constrained by the 32 character width of the television screen, but can print lines that are as long as your printer can handle.

It is important to note that the ASCII character set is not identical to the ZX81 character set. For instance, the ZX81 graphics symbols are not valid ASCII symbols, and cannot be printed as such by a printer using ASCII. On the other hand, ASCII contains symbols that are not available on the ZX81, the whole of the lower case alphabet ('small' letters), for example, plus a number of extra symbols such as % and @, and thirty-two control codes which are used to control telecommunications links, and are also used by the more sophisticated printers to control their more complex functions, such as proportional spacing.

In order to allow you to use the full capability of such printers, the ZX99 implements the full 128 character ASCII set, including all the controls. This is done by taking ZX81 symbols for which there is no ASCII equivalent and assigning them to ASCII characters which do not appear in the ZX81 set. Appendix E gives a table showing all the equivalents, and can be used to 'translate' your ASCII requirements into ZX81 codes. For example, if you wish to print a # symbol on your printer (ASCII code 23 Hexadecimal) you use the £ sign on your ZX81.

Printer output via the ZX99 is very similar to tape output. First you organise your data in a character string array, then use Z$ as a 'signpost', exactly as for tape output, and invoke the Block Print USR (8222). There is however one extra step that must be carried out first, as described in the next section.

SELECTING PRINTER OUTPUT OPTIONS

While the ASCII character set defines the pattern to be used to represent each character, there are still certain other factors which can be varied when transmitting the information. The most obvious of these is the speed at which the data is sent, as printers run at different speeds, depending on their cost and sophistication. There are also several other options that need to be specified.

Selection of all printer options is handled through a single reserved numeric variable 'Y', in the same way that block lengths are passed via 'Z'. In actual fact, 'Y' is not treated as a number by the ZX99, but as a collection of bits which represent the various options to be selected. However, as not every user will be fluent with binary number conversions, it is easiest to treat 'Y' as a value that is simply built up by adding various 'magic numbers' together, according to which options you choose.

For a start, let us consider transmission speed. This is normally quoted as a 'baud' rate, which in this context means the number of bits that are to be transmitted each second. Pointers normally accept data at one of a choice of standard baud rates, and sometimes more than one. Your printer's instruction manual should tell you what it can handle, and you must convert the required rate into an equivalent 'magic number' according to the following table:

Baud Rate	Magic Number
110	0
150	32
300	64
600	96
1200	128
2400	160
4800	192
9600	224

For example, if your printer works at 300 baud, you pick the magic number 64. Then what do you do with it? You add it to the other magic numbers that will result from the further choices that follow.

The next factor to be determined is how many 'stop bits' your printer requires. Teletypes require two, while the majority of newer printers only require one, but again your printer's instruction manual should tell you. (You can always play safe by choosing two stop bits, but this slows down transmission slightly.) The choices you have are:

Stop Bits	Magic Number
One	0
One-and-a-half	8
Two	16

Thirdly, you must decide what form of 'parity' will be used when transmitting the data. The 'parity bit' is an extra bit added automatically to each character, which can be used by the receiving unit to check for errors in transmission. The ZX99 provides you with four options:

Parity	Magic Number
Permanent zero-bit	0
Permanent one-bit	1
Odd parity	2
Even Parity	3

Many printers ignore parity altogether, in which case any choice will work. In fact only the last two choices are valid if parity checking is to take place, but the first two are often employed when parity checking is not required. Once again, consult your printer's instruction manual.

There is one more option that can be selected through 'Y'. In order to generate lower case letters, inverse video is employed. In applications where the output is all or mostly upper case (capital letters), the most convenient situation is for normal video to generate upper case characters, as this avoids constant switching into Graphics mode. In other applications such as word processing, most of the text will be in lower case, with only the occasional capital letter. Here it would be better for normal video to generate lower case, with inverse video providing the capitals. So the ZX99 gives you the choice:

Normal Video	Inverse Video	Magic Number
Upper Case	Lower Case	0
Lower Case	Upper Case	4

The characters will still all appear as capitals on your television screen, of course, in normal or inverse video, but when the data reaches your printer the results will be as you have selected, assuming that your printer can generate lower case. (Some old ones can't!)

Now that you have made all your choices, simply add the magic numbers together and assign them to Y. For instance, you could code:

```
100  LET Y = 64+3+16
```

This would give you output at 300 baud with even parity and two stop bits. Normal video would print as capitals. You could of course save a bit of RAM space by adding the numbers together in your head first, but if you seem to be having problems with your choice of options, it is not a bad idea to enter the values as shown and let the ZX81 add them up, to make sure that it is not your mental arithmetic that is at fault.

The LET statement for Y only needs to be encountered once, so it is generally best to put it at or near the front of your program. It does not need to be executed before every print USR. (The only time you might possibly need to do this is if you have a program that wants to switch the use of reverse video depending on what you are doing, but this would hardly be normal usage.)

USING BUFFERS FOR PRINTER OUTPUT

The television display produced by the ZX81 has a line length of only 32 characters, whereas most printers allow much longer lines than this. The ZX99 allows you to take full advantage of your printer's carriage width due to the use of string arrays as print buffers, in the same way as they are used for tape input and output. Suppose you include the following statements in a program:

```
        100  DIM P$ (200)
             ...
        520  LET P$="THIS STRING IS....
        ..........LONGER THAN 32 CHARAC
        TERS"
```

Although the string will run over several screen lines when you LIST your program, this is purely for display - there are no NEWLINE characters embedded in the string when it is loaded into P$ by the LET statement. So if you now continue with:

```
        530  LET Z$ = "P"          'Signpost' to the buffer
        540  LET Z = 54            Number of characters
        550  LET A = USR 8222      'Print Block' command
```

this will cause the string to appear as a single line on your printer thus:

```
    THIS STRING IS..............LONGER THAN 32 CHARACTERS
```

Since you can make your buffers any size you like (within the limits of available RAM), and load them with whatever strings you wish, you can print lines of any length.

When printing reports that contain columns of figures, much of the page will consist of spaces. Coding character strings with lots of spaces in them uses up RAM memory, and it may be more efficient to use a loop to fill the whole buffer with spaces first, then use substring notation to insert data in the positions in which you want it to appear. (See Chapter 21 of your ZX81 Basic Programming Manual for information on substrings.) For example:

```
        100  DIM P$ (133)
             ...
        200  FOR I = 1 TO 133        Blank whole buffer
        210  LET P$(I) = " "
        220  NEXT I
             ...
        350  LET K = ...
             ...
        450  LET P$(20 TO 25) = STR$ K
             ...
```

PRINTER NEWLINE

In the ASCII character set, there is no single NEWLINE character. Instead, a combination of two characters is used. First, Carriage Return (0D Hex.) which returns the print head to the left hand margin, then Line Feed (0A Hex.) which moves the paper on one line. These characters are among the 32 control codes mentioned earlier. Carriage Return is generated by inverse-video $, and Line Feed by the Graphics symbol on the F key (see Appendix E). This would be a bit of a handful to enter every time you want a new line, so a neater method has been implemented. (The codes given above would work if you wanted to try them, though.)

The ZX81 symbol <> (shifted T) is not a valid ASCII character. (In ASCII you would print this using two separate characters, < and >.) So it has been assigned the task of acting as a NEWLINE for the printer wherever it appears in a printer buffer. If a buffer array is loaded with the following string:

```
        "LINE ONE<>LINE TWO"
```

it will apprea on your printer like this:

```
        LINE ONE
        LINE TWO
```

And the string:

```
        "<><><>"
```

would cause the printer to feed up three blank lines. This ability to include more than one <> symbol in a buffer can be very useful, as it means that a single print buffer can hold several lines of print at once. For instance, if you are printing a report that requires a set of headings on every page, the whole heading can be held in one buffer, even if it consists of several lines of print. Or the name and address to be printed on a mailing label could all be entered into a single buffer and the whole label produced with a single print command. In fact, there is nothing to stop you holding a mailing list on tape in exactly this form, with the <> (shifted T) codes embedded in your data as line separators, as the tape read and write commands give these codes no special significance - it is only the Print Block command that interprets them as newlines for the printer.

Because text processing applications may need to use inverse video for alphabetic upper or lower shift, the graphics symbol on the T key also performs the same printer newline function as <>. You can therefore use shifted T for newline whether you are in graphics or normal mode. (Similarly, for convenience, the comma and full stop characters print correctly whether entered in normal or inverse-video mode.)

Note that to generate a printer newline you must use the single <> (shifted T) code. Using a < followed by a > will not have the required effect, even though it may look exactly the same on the television screen.

One final point - there is no automatic newline at the end of a block print operation. If you want a new line, you must include a <> as the last character to be transmitted from the buffer. Alternatively, you could send it afterwards as the first character of the next buffer, or even as a separate operation on its own. So, not only can one buffer hold several lines of print, but conversely one line of print can be sent in several parts, should you so wish, as you will not get a new line until you insert a <> code.

BLOCK PRINT

The USR number for Block Print is 8222. We can now summarise the sequence of operations for generating printed output:

1) Define an output buffer by means of a DIM statement, e.g.:

```
    110 DIM P$ (133)
```

2) Set the print options that you require, through the reserved variable 'Y', e.g.:

```
    120 LET Y = 64+3
```

(Operations (1) and (2) above generally need to be carried out once only at the beginning of a program.)

3) Load the data to be printed into the buffer, including <> symbols wherever printer newlines are required,

4) Set Z equal to the number of characters that you wish to write, e.g.:

```
    300 LET Z = 81
```

5) 'Point' to the buffer via Z$, e.g.:

```
    310 LET Z$ = "P"
```

6) Issue the 'Print Block' command itself, e.g.:

```
    320 LET CC = USR 8222
```

7) Return the ZX81 to SLOW mode, if required, e.g.:

```
    330 SLOW
```

8) Analyse the completion code and take appropriate action, e.g.:

```
    340 IF CC<>0 THEN STOP          Trap errors
```

When checking the completion code, the only types of errors that you can get after a Print Block are all due to programming mistakes. (I.e. ther is no equivalent to the tape read error and end-of-file conditions that you can get when reading tape.) A simple STOP trap (as shown above) is therefore an adequate check on the completion code, as once your program is debugged you should always get a completion code of zero. It is good practice to leave the trap instruction in your program permanently, though, just in case there are undiscovered bugs, or in case you ever restart your program in the middle, having inadvertently deleted or cleared any necessary reserved variables.

LISTING PROGRAMS VIA THE RS232C INTERFACE

Printing program listings via the RS232C output is very straightforward. First, you must indicate the options you require through the reserved variable 'Y', as defined in the previous chapter. The only difference is that you now enter it as a direct command from the keyboard, not as a program statement (i.e. leave off the line number), e.g.:

```
LET Y = 64
```

This is then simply followed by the USR for Program Listing (8225), again as a command direct from the keyboard:

```
LET A = USR 8225
```

and the program currently in RAM will be listed out on the printer. When the listing is complete the ZX81 will be in FAST mode, so return it to SLOW mode if you prefer by entering:

```
SLOW
```

as a direct command.

If you compare the printout with the program listing as it appears on the television screen, one or two minor differences will be apparent. Long lines that ran over more than one line on the screen will now print straight across the page. Any ZX81 Graphics characters will of course be replaced by their equivalents in the ASCII set, as given in Appendix E. If this makes your listing confusing, one alternative is to use the CHR$ function to enter graphics codes rather than the symbols themselves.

The other difference is due to a ZX99 feature intended to improve the legibility of programs. After every unconditional GOTO, RETURN or STOP statement a blank line is inserted, highlighting that program flow from one statement to the next stops at that point, and making the structure of the program more apparent. (The 'structured programming' approach, preached in professional programming circles, promotes techniques that enhance the clarity and 'readability' of programs, and any improvement in legibility is beneficial.)

ZX99 USR COMMAND SUMMARY

USR Address	Function	Parameters Required
8192	Release all tape drives. (Deselects currently selected drive, if any.)	None
8195	Select Input Drive 1.	None
8198	Select Input Drive 2.	None
8201	Select Output Drive 1.	None
8204	Select Output Drive 2.	None
8207	Select both Output Drives.	None
8210	Write to selected Output Drive.	Z$, Z, Buffer
8213	Read from selected Input Drive.	Z$, Z, Buffer
8216	Skip next block on selected Input Drive.	None
8219	Copy whole tape. Tape to be copied must be mounted on Input Drive 1, and blank tape on Output Drive 1 (and optionally on Output Drive 2).	None
8222	Output block to RS232C Interface.	Z$, Z, Y, Buffer
8225	List program via RS232C Interface.	Y

ZX99 PARAMETER VARIABLES

Parameter Variable Type and Usage

Z$ Type: Scalar string variable (not an array).

The first character of Z$ must be a letter in the range A-Y indicating the name of the Buffer array to be used in the next Tape or Print operation.

Z Type: Scalar numeric variable (not an array).

For tape or print output, you must load Z with the length of the data to be written, before invoking the USR. (The length to be written may be less than the full length of the buffer.)

For tape input, the length of the data block read by the ZX99 is placed in Z on return from the USR. If the block read from tape was bigger than the available buffer size, then Z equals the size of the buffer.

Buffer Type: String array of zero dimensions. Must be defined in a DIM statement, e.g.:

```
110 DIM X$ (2000)
```

The name-letter of the Buffer must subsequently be loaded into Z$ (see above).

For tape or print output, data must be loaded into the Buffer before invoking the USR.

For tape input, the Buffer receives the data read from tape. If this is less than the length of the Buffer, the contents of the remaining space will be unchanged. If the data block is too large for the buffer, the excess will be lost.

Y Type: Scalar numeric variable (not an array).

Defines RS232C output options, by summing magic numbers with one selection from each of the following groups:

1)

Baud Rate	Magic Number
110	0
150	32
300	64
600	96
1200	128
2400	160
4800	192
9600	224

2)

Stop Bits	Magic Number
One	0
One-and-a-half	8
Two	16

3)

Parity	Magic Number
All Zero	0
All Ones	1
Odd Parity	2
Even Parity	3

4)

Normal Video	Inverse Video	Magic Number
Upper Case	Lower Case	0
Lower Case	Upper Case	4

ADDITIONAL BASIC REPORT CODES

These Report Codes are returned to the user in exactly the same way as the ZX81's own BASIC Report Codes.

Report Code	Indication
D	BREAK key pressed during a ZX99 function. (But see also the Note below.)
G	Insufficient memory space available for ZX99's temporary work area.
H	'Z' Length-Specification variable is undefined or incorrectly defined. (Examine the USR's Completion Code for more detailed analysis.)
I	'Z$' Buffer Pointer is undefined or incorrectly defined, or the indicated buffer is undefined or incorrectly defined. (Examine the USR's Completion Code for more detailed analysis.)
J	'Y' RS232C Option Control variable is undefined or incorrectly defined. (Examine the USR's Completion Code for more detailed analysis.)

Note: When initiating certain ZX99 operations directly from the keyboard, such as Block Skip or Program List, you may find that operation stops almost immediately, with Report Code 'D' being returned. This is caused by keeping your finger on the NEWLINE key for too long at the end of entering the command from the keyboard. The solution is just to press the NEWLINE key quickly and release it immediately.

The same problem can also occur within a program if an INPUT or INKEY statement is followed closely by tape input/output or printer output.

ZX99 COMPLETION CODES

Completion Code — Indication

0 No errors. Normal completion.

1 The selected Input Drive was read for 20 seconds, but no data was received. This is the normal condition for end-of-data on the tape, but it may also be caused by any of the following:

- The cassette recorder power is switched off.
- The cassette recorder does not have a tape loaded.
- The cassette recorder does not have 'PLAY' (or the equivalent) engaged.
- The cable between the tape recorder and the ZX99 is not plugged into the EAR socket on the recorder.
- The cable between the tape recorder and the ZX99 is not properly inserted, is not making good contact, or is damaged.
- The cable between the tape recorder and the ZX99 does not enter one of the EAR sockets on the LEFT-hand side of the ZX99.
- The cable between the ZX99 and the ZX81 does not connect the EAR socket on the ZX81 to the EAR socket on the TOP face of the ZX99, or is not properly inserted.
- The required tape channel has not been selected by the software, or directly through the keyboard. (This is indicated by the appropriate LED not being lit.)

2 When the ZX99 scanned through your program's variables to find the reserved variables (Z, Z$, etc.), it found corrupted data. This will not happen under normal circumstances, but could be caused by POKE, if you have inadvertently used a wrong address and overwritten control information in the Variables region of memory. (See Chapter 27 in your ZX81 Basic Programming Manual.)

3 The Z$ reserved variable is undefined. This should be a string in which the first character is a letter indicating the name of your current input/output buffer. Z$ must be defined before any tape input or output, or any printer output. (Any characters after the first are ignored by the ZX99.)

Completion Code Indication

4 The Z$ reserved variable is defined, but as a string array. For use with the ZX99 it must be defined as a simple string (i.e. not in a DIM statement).

5 The Z$ reserved variable is defined, but has zero length. I.e. it is empty. (See C.C.3 above for more details.)

6 The first character of Z$ is not a character in the range A-Y. (See C.C.3 above for more details.)

7 There is no string array defined with the name-letter indicated by the contents of Z$. I.e. your input/output buffer is undefined, or else the first letter of Z$ does not identify it correctly.

8 The string identified as your buffer (via Z$) is only a simple string, and should be defined as a string array. I.e. it should be declared in a DIM (Dimension) statement before any Read, Write or Print USR calls are made to the ZX99.

9 The string identified as your buffer (via Z$) is a multi-dimensioned array and should be a string array with no dimensions. (See Chapter 22 of your ZX81 Basic Programming manual, Exercise 2 on page 145.)

10 The Z reserved variable is undefined. This is a numeric variable which is used to pass to the ZX99 the length of the data to be written or printed, or to receive the length of the data actually read in from tape. Note that even when reading data, the variable Z must be entered into the variable list before the ZX99 is asked to use it. To do this, simply assign any value to Z before calling USR 8213. For example:

```
1000 LET Z=0
```

11 The Z reserved variable is defined, but is either out of range or is not an integer value. Z must be a positive integer value in the range 0 to 65535.

12 The value of Z (which should indicate the length of the data block that you wish to output) is larger than the size of your input/output buffer (as identified via Z$). If the value of Z appears to be correct, check that Z$ is pointing to the correct buffer.

13 You have tried to write an output record that is below the minimum limit (40 bytes). In order to distinguish between genuine blocks of data on tape, and odd bursts of noise which can occur in the inter-record gaps, all output tape blocks must be a minimum of 40 characters long. On input, all blocks of less than 40 characters are treated as noise and ignored.

Completion Code Indication

14 The record read in from tape was longer than the full size of your input buffer (as declared in its DIM statement). Data was read into your buffer until it was full, and the rest of the block was then skipped over without storing it in RAM.

15 A tape read or write or print operation was interrupted by use of the BREAK key.

16-22 These codes all indicate that errors were detected while reading data from tape. They indicate various combinations of the three possible faults as follows:

Completion Code	Fault A	Fault B	Fault C
16	YES	NO	NO
17	NO	YES	NO
18	YES	YES	NO
19	NO	NO	YES
20	YES	NO	YES
21	NO	YES	YES
22	YES	YES	YES

Fault A: Data bits were detected with durations below the minimum acceptable level for a good zero bit.

Fault B: Data bits were detected with durations in between the valid windows for zero bits and one bits.

Fault C: Data bits were detected with durations above the maximum acceptable level for a one-bit.

30 The Y reserved variable is undefined. This variable is used to specify baud rates and other options for the RS232C output interface. (See Appendix B for a summary of the specifications for Y.) Y must be defined through a LET statement before using either the Print Block command (USR 8222) or the Program List command (USR 8225). When performing a program listing, Y should be defined first with an immediate LET command (i.e. one without a line number in front of it).

31 The Y reserved variable has been defined, but is not a positive integer in the range 0-255. (See C.C.30 above for further details, and also Appendix B.)

ASCII CHARACTER SET EQUIVALENTS

ASCII		ZX81		ASCII		ZX81	
HEX	CHAR.	CHAR.	CODE	HEX	CHAR.	CHAR.	CODE
00	NULL	**	216	20	SPACE	SP	0 or 128
01	SOH	g-Q	129	21	!	i-?	143
02	STX	g-W	130	22	"	""	192
03	ETX	g-6	131	23	#	£	12
04	EOT	g-R	132	24	$	$	13
05	ENQ	g-8	133	25	%	i-:	142
06	ACK	g-Y	134	26	&	i-;	153
07	BELL	g-3	135	27	'	g-A	8
08	BSPACE	g-H	136	28	(	(	16
09	H.TAB	g-G	137	29	)	)	17
0A	L.F.	g-F	138	2A	*	*	23
0B	V.TAB	i-"	139	2B	+	+	21
0C	F.F.	i-PND	140	2C	,	,	26
0D	C.R.	i-$	141	2D	-	-	22
0E	S.O.	<=	219	2E	.	.	27
0F	S.I.	>=	220	2F	/	/	24
10	DLE	i-0	156	30	0	0	28
11	DC1	i-1	157	31	1	1	29
12	DC2	i-2	158	32	2	2	30
13	DC3	i-3	159	33	3	3	31
14	DC4	i-4	160	34	4	4	32
15	NAK	i-5	161	35	5	5	33
16	SYN	i-6	162	36	6	6	34
17	ETB	i-7	163	37	7	7	35
18	CAN	i-8	164	38	8	8	36
19	EM	i-9	165	39	9	9	37
1A	SUB	g-1	1	3A	:	:	14
1B	ESC	g-2	2	3B	;	;	25
1C	FS	g-7	3	3C	<	<	18
1D	GS	g-4	4	3D	=	=	20
1E	RS	g-5	5	3E	>	>	19
1F	US	TO	223	3F	?	?	15

NOTE: i- Denotes inverse of indicated character. (I.e. enter GRAPHICS mode, then type the specified character.)

g- Denotes the graphics symbol associated with the indicated character. (I.e. enter GRAPHICS mode, then hold SHIFT down while typing the character.)

(ASCII CHARACTER CODES - CONTINUED)

ASCII		ZX81		ASCII		ZX81	
HEX	CHAR.	CHAR.	CODE	HEX	CHAR.	CHAR.	CODE
40	@	g-D	9	60	`	g-S	10
41	A	A	38	61	a	i-A	166
42	B	B	39	62	b	i-B	167
43	C	C	40	63	c	i-C	168
44	D	D	41	64	d	i-D	169
45	E	E	42	65	e	i-E	170
46	F	F	43	66	f	i-F	171
47	G	G	44	67	g	i-G	172
48	H	H	45	68	h	i-H	173
49	I	I	46	69	i	i-I	174
4A	J	J	47	6A	j	i-J	175
4B	K	K	48	6B	k	i-K	176
4C	L	L	49	6C	l	i-L	177
4D	M	M	50	6D	m	i-M	178
4E	N	N	51	6E	n	i-N	179
4F	O	O	52	6F	o	i-O	180
50	P	P	53	70	p	i-P	181
51	Q	Q	54	71	q	i-Q	182
52	R	R	55	72	r	i-R	183
53	S	S	56	73	s	i-S	184
54	T	T	57	74	t	i-T	185
55	U	U	58	75	u	i-U	186
56	V	V	59	76	v	i-V	187
57	W	W	60	77	w	i-W	188
58	X	X	61	78	x	i-X	189
59	Y	Y	62	79	y	i-Y	190
5A	Z	Z	63	7A	z	i-Z	191
5B	[	i-(	144	7B	{	i-<	147
5C	\	i-/	152	7C	|	i-*	151
5D	]	i-)	145	7D	}	i->	146
5E	^	i-+	149	7E	~	i-=	148
5F	_	i--	150	7F	RUBOUT	g-E	7

COMBINED CARRIAGE RETURN AND LINE FEED

Either <> or the graphics symbol on the same key (i.e. g-T) will generate a Carriage Return (ASCII OD Hex) followed by a Line Feed (ASCII OA Hex). This simplifies insertion of CR/LFs when entering text in upper or lower case. (NEWLINE cannot be used because of its special significance for the ZX81 as the keyboard entry terminator.)

RS232C INTERFACE CONNECTIONS

A full RS232C interface uses a twenty-five pin connector, generally of a standard type known as a 'D' sub-miniature multipole connector. Unfortunately, it is not possible to supply a single cable that will satisfy all requirements, as equipment manufacturers interpret the RS232C standard in slightly different ways. Even with the minimal connections required by the ZX99 there is scope for variation.

The D-type connector that you require may be male (plug) or female (socket) depending on the opposing connector on your printer. Pin numbers are normally moulded into the plastic of the connector body, and you should take great care to identify pins correctly. Note that the pin numbering on a male plug is the mirror image of the numbers on a female socket, so that the numbers on the two halves correspond with each other.

The ZX99 requires only two connections, achieved at the ZX99 end with a standard 3.5mm jack plug, as used for the tape input and output lines. The outer sleeve of the jack plug is the 'Signal Ground' connection, and this should be connected to pin seven (7) on the 25-way D-type connector.

The tip of the jack plug carries the 'Transmitted Data' signal and this is normally connected to the 'Received Data' input at the printer, which is pin three (3) on the D-type connector. Some manufacturers however interpret the standard such that the input should be received on the 'Transmitted Data' circuit, which is pin two (2) of the D-type connector. It all depends on your viewpoint whether the data is being transmitted to you or by you. Your printer's instruction manual should tell you which is the correct connection.

Although these are the only connections necessary to carry the data, your printer may require certain pins to be linked together inside the D-type plug before it will function. This is because some printers are designed to be able to work with a modem, while others have the ability to receive data faster than they can actually print, providing they can use one of the RS232C circuits to suspend the flow of data when their internal memory is full. Such operation is not possible with the ZX99, and you must select the baud rate such that your printer can always cope, even if the flow of data is continuous.

Printers that make use of the interface control circuits as described above, often need to have these inputs tied to a particular state in order to 'unlock' their data input. This is normally achieved by linking pins inside the D-type connector. Connecting pin 4 (Request to Send) to pin 5 (Clear to Send) is a common requirement, or alternatively pin 6 (Data Set Ready) and pin 20 (Data Terminal Ready) may need linking, or some other combination may be called for. Your printer's instruction manual should help you, or failing that, the supplier, but do not make these extra connections unless you are sure that they are needed.

ZX99 IMPLEMENTATION CONSIDERATIONS

Address Space Usage

The ZX99 contains a 2K ROM which holds all the code for its USR subroutines. This is located immediately following the ZX81's own ROM in the memory address space (i.e. from 8192 to 10239), but since the address is not fully decoded the ZX99 will respond to any address in the range 8192 to 16383, precluding use of these addresses by other add-on devices. The output latches that control the tape drives and the RS232C interface are also memory mapped into the same region of addresses.

ZX80 Compatibility

The ZX99 makes use of the 'ROM C.S.' connection on the ZX81 extension connector, pin number 23B. (See Chapter 26 of your ZX81 BASIC Programming Manual.) This output is not available on the ZX80 microcomputer, and the ZX99 cannot therefore be used with a standard ZX80.