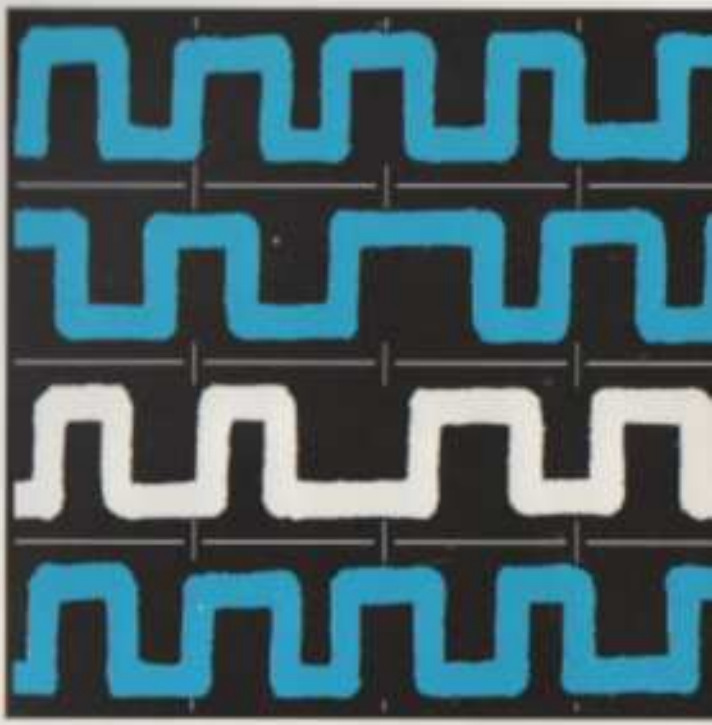
BASICODE 2

THE CHIP SHOP



BBC Radio 4

THE CHIP SHOP

BASICODE 2

BBC Radio 4

BSS

Broadcasting Support Services

ISBN 0-906965-14-4

This edition first published by Broadcasting Support Services January 1984

THE CHIP SHOP
BBC Radio 4
British Broadcasting Corporation
Portland Place
London W1A 1AA

The BASICODE-2 kit is available for £3.95 from:

Broadcasting Support Services
P.O. Box 7
London W3 6XJ

Please make cheques or postal orders payable to Broadcasting Support Services.

Published for The Chip Shop, Radio 4, by Broadcasting Support Services – an independent educational charity providing follow-up services for viewers and listeners.

CONTENTS

Introduction

BASICODE-2

BASICODE has been developed by the radio programme *Hobbyscoop* which is broadcast weekly by Nederlanse Omroep Stichting (NOS), the Dutch domestic service.

This book is based very heavily on the handbook published by *Hobbyscoop* and we would like to thank all our Dutch friends and colleagues for their help in producing this English version.

In particular, special thanks to:

Hans Janssen,	Producer, NOS, *Hobbyscoop*
Jonathan Marks,	Producer, Radio Netherlands, (Media Network)
Klaas Robers,	Inventor BASICODE
Hochem Herrmann,	Developer BASICODE-2

And contributors:

J.J.H.M. Duyf, J. Haubrich, R. Jansen, M. Reinders S. Faber, Ir. P.G.M. Maathuis, David Dawson, H.J. Koevets

TREVOR TAYLOR, Producer, The Chip Shop, BBC Radio 4

Hans Janssen

Jonathan Marks

Klaas Robers

Jochem Herrmann

Chapter One

INTRODUCTION

This book and the accompanying cassette contain the details of NOS-BASICODE. It has been designed as a standard for software exchange between different brands of computers. These include some of the most popular brands being used in many different parts of the world

APPLE II & IIe	**TRS-80**
BBC Models A & B	**VIC 20**
Commodore 64	**Videogenie**
Sharp MZ80A	**ZX81**

Side 1 of the cassette contains the various *translation* programs, each specific to a particular brand of computer. In simple terms you first *teach* your computer the BASICODE standard by loading the translation program, then you can read and write, using the standard. Instructions on how to do this are contained in the following chapters.

Of course, it's possible that your brand of home computer is not listed. If so, in Chapter Four, the BASICODE protocol is described, which will help you compile a translation program for your computer. Naturally, you will need some programming experience, but if you succeed, let us know so we can then include it in future editions of this handbook. Translation programs should be sent to us on cassette with a printed listing if possible. The address is The Chip Shop, BBC, London W1A 1AA.

Programs following the BASICODE protocol are also welcome. These may be used on the air so that others can benefit. Credit will be given to the author.

It is important to stress that BASICODE has been produced on an entirely **non profit-making** basis.

It has its origins in the weekly radio programme *Hobbyscoop* which is broadcast by the Dutch domestic service, Nederlandse Omroep Stichting (NOS).

Hobbyscoop transmitted its first computer program in 1978, at which point many listeners wrote to ask if there was something wrong with their radio receiver! The following year computer data was transmitted each week in cooperation with *TELEAC* which is roughly equivalent to the Open University.

At that time four brands of computer were popular: Apple, Exidy Sorcerer, PET and Tandy TRS-80. Each computer therefore had its own week in the month when machine-readable data could be transmitted. But this led to problems for the broadcasters. Two of the computers used a very slow baud rate which meant that *intelligent* programs took up far too much airtime on a national radio network. Up to eight minutes of objectionable noises was too much even for the enthusiasts! Added to this, the cassette systems proved to be rather unreliable on two of the four computers. Only a very small group of the Dutch computing public was being reached at any one time.

Klaas Robers, who is a well-known inventive radio amateur in The Netherlands, then came up with an interesting proposal. This involved the creation of a type of *Computer Esperanto* which could be *read* and *written* by various home computers. A group of computing hobbyists got together and BASICODE was the result of many months of research. Later, when the limitations of the first BASICODE were recognised, *Klaas Robers*, together with *Jochem Herrmann*, developed the improved BASICODE-2. This has now been in regular on-air use since January 1st 1983.

It was soon discovered that the Dutch medium wave signal was reaching outside the Dutch borders. Letters came from computer enthusiasts in England, Germany, Belgium, France and Denmark asking for more information. Further international interest was created when the Dutch external service, which broadcasts worldwide on short wave, took an interest in BASICODE. The Thursday English language programme *Media Network* ran a feature on the code. They also tried a series of popular experiments with a slowed down version of BASICODE running at 300 Baud. The data went out on short wave and listeners within 1,200 miles of the transmitters reported success.

The weekly *Hobbyscoop* programme is in Dutch, naturally, but it can be received in some parts of the UK. It can be heard on Sundays at 1810 GMT in winter, 1710 GMT in the summer months, on medium wave 747 kHz, or 401 metres. Because listeners in Denmark, Belgium, Southern Britain and West Germany can load BASICODE programs from medium wave without problems the computer segment of the programme (around 1840 or 1740 GMT), now includes short explanations in English. If you are in range of the medium wave broadcasts, tune in and see if you can receive them.

Apart from the people named above who have been involved with the success of BASICODE, many individuals in The Netherlands have helped us in the production of translation programs and, in some cases, hardware designs. We thank them.

Chapter Two

HOW TO USE BASICODE-2

BASICODE-2 was developed because different micro-computers not only speak different dialects of BASIC, but also record or *save* the programs onto cassette in different ways. All micro-computers that use cassettes record their programs as a series of bleeps but different machines use different frequencies and various methods of coding. BASICODE-2 is an audio encoding standard that can be used to store and retrieve software and can be used with a large number of micro-computers. Once a micro-computer has been *taught* BASICODE-2 it can then run programs written on different makes of micro-computer. The only problem that remains is the differences in BASIC dialect but BASICODE-2 again gives us the solution. Our *Computer Esperanto* interprets the program instructions so that they will work on all the micro-computers listed in Chapter One. This does mean that if you want to write programs for BASICODE-2 you should write them according to some simple rules. You will find these in the chapter written specifically for your micro.

How to load BASICODE-2

On the first side of the tape you will find a number of different versions of BASICODE-2. They are each preceded by an introduction, so find the one for your micro-computer and load it in the normal way. There will be some special points to bear in mind for your particular computer so refer to the relevant chapter for full details.

Once you have loaded BASICODE-2 you are ready to try out some BASICODE-2 programs. On the second side of the tape you will find some demonstration programs. To load one of these programs you should follow the instructions on the screen and also, for some micros, refer to the instructions in the relevant chapter. When the program starts loading you will see the program being listed on the screen. You will be able to check the program for errors and make corrections before running it.

Now that BASICODE-2 is running on your micro here's how you can use the *Chip Shop* programs.

Chip Shop Takeaway

The *Takeaway* service allows you to get software from your radio. The BBC

is broadcasting software late at night on Radio 4. Check in the Radio Times for details of transmission times. The software being transmitted is written in BASICODE-2, so, armed with the translation tape, you will be able to run the programs on your micro.

To make sure that the programs run properly you need to make a good recording. Here are some tips:

1. If possible record off VHF, because it's a higher quality signal and less susceptible to interference. If that's not possible then use long wave or medium wave. For frequencies see the Radio Times.

2. Use a direct connection between your tape recorder and the radio. If you use a microphone someone's bound to burst into the room at the crucial moment!

3. Set the recording level before the software is transmitted. Don't over-record as this may distort the signal and the program might not work.

4. Use a short tape if possible: C15 or C30. These are less likely to stretch and it can be frustrating looking for programs on very long tapes.

Once you have successfully recorded the program on the tape you can load it as you did with the demonstration programs.

One final point. When the Dutch started broadcasting software on medium wave, they received reports of near perfect recordings from Germany, Belgium, France, UK and Denmark. We hope that the *Takeaway* service will be as widely received as possible, inside and outside the UK.

Writing BASICODE-2 programs

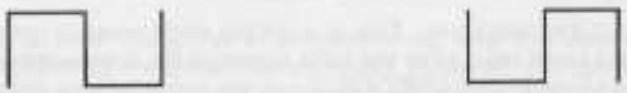
Having run some programs in BASICODE-2 you might like to write some programs. These will then run on any micro using the translation program. Because of the different BASIC dialects, you will need to write the programs according to a set of rules. BASICODE-2 ensures that certain operations (like clearing the screen or moving the cursor) will always work, no matter what make of computer you use. Details of how this is achieved can be found in Chapter Three. The programming rules for your particular micro can be found in the relevant chapter. Some micros have smaller memories than others, so you may find that very large programs don't run on all the machines.

Chapter Three

BASICODE – THE SPECIFICATIONS

Tone Modulation

Two tones are used to record data onto cassette tape, the frequencies are 1200 and 2400 Hz. A "0" is defined as one full cycle of 1200 Hz.



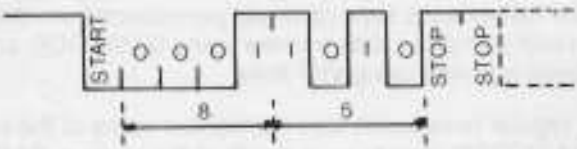
A "1" is defined as two full cycles of 2400 Hz.

Sequence code

The speed of BASICODE is 1200 Baud. A byte comprises the following sequence:

1 startbit (logic 0)
8 databits (the least significant first)
2 stopbits (logic 1)

For example:



BASIC information

The BASIC program is coded either in the form in which it has been typed in, or as it is displayed on the screen when the command LIST is given. In most cases, however, this is **not** the form in which it is retained in the computer's memory.

All letters and ciphers are presented in American Standard Code for Information Interchange (ASCII), and each BASIC line is closed with *CR* (Hex 8D). Each ASCII byte in the program receives an eighth bit=1.

The tone sequence

The cassette tape recording of a BASICODE program will consist of:

Leader: 5 seconds of stopbit(=2400 Hz)
ASCII *Start Text* (Hex 82)
BASIC Information in ASCII
Checksum
Trailer: 5 seconds of stopbit(=2400 Hz)

Checksum

A note about the checksum. This is a simple code used to check that the program has been read from the tape successfully. If an error is indicated by the checksum then BASICODE-2 will tell you and you should list the program and correct the error. Then you can SAVE the program in the usual way.

Technical note: the checksum is a number stored as an 8 bit computer word. It is calculated by performing an *exclusive – or* on all the previous bytes in the block. Unlike all the other transmitted ASCII characters, the checksum's eighth bit is not set to "1" but may be "1" or "0".

The improved BASICODE-2

The idea of the first BASICODE was to be able to transmit machine-readable programs over the air. For the most part this worked extremely well. A program from microcomputer X could be read by computer Y, simply by using the BASICODE translation code tape in between. The regular transmission of BASICODE programs since 1979 by the Dutch domestic service (NOS) showed that BASICODE was extremely reliable, often better than the computer's own cassette interface standard. Radio amateurs in the Netherlands soon obtained permission from the Dutch PTT to experiment with computer data transfer using BASICODE, and excellent results have been reported using VHF links.

However, the regular broadcasts also highlighted some of the limitations of the original BASICODE, mostly as a result of the various BASIC dialects. The rules of the original protocol did not work in all cases.

So, based on experience, improvements were made to the standard to create BASICODE-2. The working of various BASIC instructions was examined in the various dialects. In doing so, many differences were discovered. However, there was common ground amongst the computers

taking part in the BASICODE working group.

All computers worked in Microsoft-BASIC, or had the possibility of doing so. A subset of standard BASIC instructions has now been drawn up, which has the same meaning in all the various BASIC dialects. These are dealt with in more detail in the next chapter. There are also a number of very important functions which are brought into action using different commands on different computers. These include the clearing of the screen and the commands to a printer.

The solution in this case is in the form of subroutines on agreed fixed BASIC line numbers. The subroutines are of course different for each brand of computer, but they all work in much the same way. The subroutines are called when a particular function is required.

This means a new discipline in programming and the adjustment of existing programs to meet the new standard. However, the effort appears to be worth the trouble, since BASICODE-2 programs now work on many brands of computer without having to be adjusted by the user before they will run.

The subroutines are contained in the BASICODE-2 translation programs on Side 1 as part of the loading sequence and therefore do not have to be part of BASICODE-2 programs you exchange with other users. The cassette with this handbook contains the translation programs you will need. In most cases it is possible to choose whether or not the subroutines are read or recorded with the main BASICODE program.

BASICODE-2 is now the fruit of some years of experiments and is a practical medium of data transfer. It is a simple solution in cases where programs have to be distributed to a number of computer brands.

NOS BASICODE–2 PROTOCOL

We have already explained the *Esperanto* concept of NOS-BASICODE, and the idea of writing one program that can be of use to a number of different computers. What follows now is a detailed description of the standard, designed for programmers who wish to use BASICODE–2.

1. BASICODE–2: The ground rules

There are four basic rules in the BASICODE–2 protocol:

a. We only use BASIC statements that are understood by all brands of computers. A list of these can be found in section 5 of this chapter.

b. Since rule a. leads to a lot of practical problems, such as the inability to clear the screen in a standard way, line numbers below 1000 are reserved for subroutines. These perform the extra functions needed in most programs that cannot be achieved using the standard BASIC statements. The precise function of these subroutines is described in section 3. GOSUB 100, for example, is used to clear the screen, GOSUB 110 puts the cursor at a specific point on the screen, and so on. These routines are of course specific to each computer brand and therefore they are not part of the main program in BASICODE–2 standard. They are therefore included in the translation programs for each brand of computer. When combined with the BASICODE–2 data, the whole program is then complete.

c. We have to standardise the video display screen as consisting of 24 lines, each of 40 characters. If you are programming in BASICODE–2 then it is important that you bear the screen size in mind! Unfortunately the rule is not quite as simple as this. There are computers in the group with only 16 lines on the screen (e.g. TRS–80) and some that can only accommodate 22 characters per line (e.g. VIC–20). So unless it is really necessary, do not use more than 16 lines on the screen, and do not make lines longer than need be. Better still, use a subroutine which adjusts the lines to the size of the screen. This is possible using BASICODE–2.

d. A line should not be longer than 60 characters including the spaces and the line number.

2. The program construction

The following line number scheme is used to build up a BASICODE–2 program:

0 - 999: Standard routines (see section 3). These routines are different for each computer and are therefore contained in the translation program.

1000: first line of the BASICODE-2 program. It must be in the following form: 1000 A=(value): GOTO 20: REM program name. (value) is the maximum number of characters that can be used by all strings together. Line 20 is used to reserve memory space for the strings in those computers which need it.

1010-32767: the main program. There are no restrictions on this section, except that line numbers above 32767 are forbidden.

Although it is not compulsory, it is a good idea to build up your BASICODE-2 program systematically. Other users can then quickly understand your method of working, and it is then easier to adapt if necessary. We suggest that you use the following scheme:

1000 - 19999: the main program.

20000 - 24999: subroutines which you need for your program, but which contain statements which are not allowed in BASICODE-2.

25000 - 29999: lines with DATA statements.

30000 - 32767: lines with REM statements. You can use this space for any background details about the program, references or your name and address.

It is worth noting at this point the use of the subroutine lines 20000-24999. Try to avoid the use of statements that are **not** allowed in BASICODE-2. In some cases, of course, this is unavoidable, for example, the storage of variables on tape or disk. This is when you use the lines 20000-24999. It is extremely important that you indicate exactly what these routines are intended to do, or other brand users will not be able to follow your logic. It is also good practice to use line numbers in steps of 10, thus leaving plenty of room to insert lines at a later stage.

3. Operation of the BASICODE-2 standard routines

These are written specifically for each computer brand and form part of the translation program. They look different in the various BASIC dialects but in fact they do exactly the same in each case.

GOSUB 100:

This subroutine is used to clear the screen and set the cursor at position 0,0 (i.e. the top left-hand corner of the screen.)

GOSUB 110:

This places the cursor at a specific point on the screen. The exact point can

be chosen by specifying the variables HO and VE. HO is the position on a line (0 is the furthest position to the left), and VE is the number of the line. The top line on the screen is considered to be number 0. Remember that in BASICODE-2, the maximum screen size is 24 lines, each of 40 characters. For this reason HO must not be larger than 39, and VE cannot be greater than 23. The variables HO & VE are not changed in value on calling the subroutine.

GOSUB 120:

This gives the position of the cursor on the screen and sets the variables HO and VE. HO=0 is the first position on any line, and VE=0 implies the top line on the screen. By using this subroutine, and GOSUB 110, you can move the cursor about the screen.

GOSUB 200:

This checks if a key has been depressed, and if so, puts the character in the variable IN$. If no key was pressed then IN$ is an empty string. In principle it is possible to put any character on the keyboard into IN$, including control characters. But **be careful**. Not all control characters have the same function on different brands of computers. So it is best to avoid control characters. The RETURN (or ENTER, NEWLINE, etc) key, though, does have ASCII code 13 on all computers.

GOSUB 210:

This subroutine waits until a key has been depressed and then sets the character in the variable IN$. It differs from GOSUB 200 in that this subroutine waits until a key has been pressed, whilst GOSUB 200 only checks whether a key has been depressed.

GOSUB 250:

This is used to activate the *beep* on most computers with this facility. Note that the pitch and length of the tone is **not** specified in this routine, and therefore it is not suitable for making music.

GOSUB 260:

This gives a random number in the variable RV. The number is always smaller than 1 and larger than, or equal to, 0. It is a useful routine in statistical programs or in games.

GOSUB 270:

This tidies up the variable space and reports how much memory space is still available. The variables are **not** erased!! The number of free bytes over is placed in the variable FR.

GOSUB 300:

This creates a string SR$ using the value of the variable SR. This string has no spaces at either the beginning or the end of the number, in contrast to the BASIC statement STR$( ). In some BASIC dialects it does contain spaces, so the statement STR$( ) is thus forbidden in BASICODE-2.

GOSUB 310:

This routine creates a string SR$ which is determined by the variables CT, CN, and SR. SR$ is equivalent in value to the variable SR and is always in fixed-point notation. The total length of SR$ is thus CT characters, of which CN characters come after the decimal point. If the number does not fit into the given form, then SR$ will consist of CT asterisks. If necessary, SR will be rounded off. The variables CT, CN, and SR are not changed by calling this routine. Here are a few examples:

CT=7: CN=3: SR=2/3: GOSUB 310 gives SR$ as "0.667"

CT=8: CN=5: SR=-1.1E-3: GOSUB 310 gives SR$ as "-0.00110"

CT=3: CN=0: SR=23.6: GOSUB 310 gives SR$ as "24"

CT=3: CN=1: SR=100: GOSUB 310 gives SR$ as "***"

GOSUB 350:

Prints SR$ on the printer, but does not close that particular line. Thus you can print more than once per line using this particular subroutine. But remember that not all users have a printer and so try to include a choice in your program between using the printer or the screen display.

GOSUB 360:

Closes the line on the printer and begins with a new line.

4. Variables

There are a few restrictions governing the use of variables in BASICODE-2 programming. This is necessary to make the exchange between different brands as simple as possible.

a. Numeric variables are real and single precision. Do not count on precision greater than 6 places of decimal.

b. Names for variables should be a maximum of **two** characters long. The first character **must** be a letter, the second character (if used) can be either a letter or a number. The names of the variables should be written using BLOCK CAPITALS. Small letters are not allowed. In the case of string variables, the name is always followed by a $. All other characters, such as !, #, or % are forbidden.

c. Logic variables are those which are "true" or "false". You are **not** allowed to use the numerical value of the logic variables. This is because in some cases the "true" is interpreted as +1, in other computers as –1. The result can only be used in an IF...THEN construction (i.e. A=3* (B=1) is not allowed).

d. Before a variable is used, it must be given a value. Do not assume that a variable is automatically given a zero value at the start of the program.

e. The maximum string variable length is 255 characters.

f. Names of variables may **not** begin with the letter O. These are reserved for use within the BASICODE–2 standard routines.

g. The following variables are excluded: AS; AT; FN; GR; IF; PI; ST; TI; TI$; TO.

h. For communication with the BASICODE–2 subroutines use is made of the following variables: HO; VE; FR; SR; CN; CT; RV; IN$; SR$.

5. BASIC statements and operators

ABS	INPUT	RESTORE
AND	INT	RETURN
ASC	LEFT$	RIGHT$
ATN	LEN	RUN
CHR$	LET	SGN
COS	LOG	SIN
DATA	MID$	SQR
DIM	NEXT	STEP
END	NOT	STOP
EXP	ON	TAB
FOR	OR	TAN
GOSUB	PRINT	THEN
GOTO	READ	TO
IF	REM	VAL

+	^	<>
–	=	<=
*	<	>=
/	>	

Above we have listed the BASIC commands and operators, most of which can be used without any problems in BASICODE–2 programs. We have no intention at this point to examine in full each of the commands and operators. Many books have already been written on the subject. However we have decided to give a short summary since there are a few restrictions on certain BASIC commands in BASICODE–2.

BASIC commands and their meaning

ABS — Gives the absolute value of the stated variable. For example:

```
A=10:B=ABS(A)      B is now=10
A=-20:B=ABS(A)     B is now=20
A=-1:B=ABS(A-5)    B is now=6
```

AND — Logic AND, can only be used for logic variables. The result is a logical result. Use brackets to show clearly the order in which the work is to be done. Examples:

```
IF (A=5) AND (B=0) THEN ....
Q=(A-5) AND (B=0):IF Q THEN..
```

ASC — This gives the ASCII value of the first character of the given string. Examples:

```
A$="A":B=ASC(A$)        B is now=65
A$="BEER":B=ASC(A$)     B is now=66
```

ATN — Gives the arctangent in radians of the given variable. For example:

```
PRINT ATN(1)
0.785398
PRINT ATN(-1)
-0.785398
```

CHR$ — This gives a character with the same ASCII value as the given variable. The variable can be anywhere between 32 up to and including 127. Be very careful with values less than 32, since control characters on different brands of computers vary. Only the RETURN key always has the same value, ASCII code 13. In addition not all computers recognise lower case letters. So be careful with ASCII codes larger than 96. For example:

```
A$=CHR$(66)     A$ now contains the letter B
```

COS — This gives the cosine of the given angle in radians. For example:

```
PRINT COS(1)
.540302
```

DATA — After this term will follow numbers and/or strings until the end of the line, which can be read using READ. On a DATA line there should be no other statements, such as REM.

The elements should be separated by a comma. String variables must be placed between quotation marks. For example:

DATA 100,200, "HELLO", "BASICODE",4.6,89

DIM — This statement is used to dimension arrays. An array can only be dimensioned **once** in a program, and before it is used. The maximum number of dimensions is two, whilst the maximum number of elements is limited by the size of the memory. One DIM statement can be used for more than one array. But there are two points to note:

1. In some computers arrays up to 10 elements do not have to be dimensioned. In BASICODE-2 programs, an array must **always** be dimensioned.

2. The element with number 0 is also allowed, so A(0) and AD$(0,0) do exist. For example:

DIM A$(12), HD(100, 100), MP(1000)

END — Used to indicate the end of a program. Do not just let a program stop. **Always** finish with an END statement.

EXP — Raises the number e (=2.71828..) to a specified power. For example:

PRINT EXP(2)
7.38906

FOR..TO..STEP..NEXT.. Program loop construction. The loop will be used at least once. STEP can be left out, in which case the step is automatically 1. NEXT must only be followed by a single variable. For example:

FOR X=10 TO 100
Program in loop
NEXT X

FOR C=A TO B STEP -3
Program in loop
NEXT C

FOR I=1 TO 10:FOR J=1 TO 5
Program in loop
NEXT J:NEXT I

Note: don't jump out of a FOR-NEXT loop before it has ended. A way to end the loop is by raising the variable to exceed the end point.

GOSUB

Used to call a subroutine and indicated by the line number following the statement. For example:

GOSUB 100.

Note, though, that A=100:GOSUB A is **not** allowed.

GOTO

Indicates a jump to a given line number. For example:

GOTO 1500

Note, though, that A=1500:GOTO A is **not** allowed.

IF..THEN

Conditional split, for between IF and THEN will be logical variables or logical comparisons. If the logic is "true" then the process continues through to statements after THEN. If not, then the computer simply moves to the next line. A line number may be given after THEN, from which the program should continue. Note that ELSE is forbidden in BASICODE-2. For example:

IF A=3 THEN B=0: C=5
IF A>3 THEN 1500
C=(A>3):IF C THEN GOSUB 100

Note:
Use IF..THEN 2000
and not: IF..GOTO 2000.

Use IF..THEN GOSUB 2000
and not: IF..GOSUB 2000.

INPUT

Asks the user to input either a number or a string variable. A string may **not** consist of either commas or colons. If commas or colons are needed, then it is better to make use of subroutine 210. A prompt string is not allowed, and neither is more than one variable after an INPUT. Most BASICs print a question mark on the screen to show that input is required. In some computers, when the RETURN key is pressed, the line from the last cursor point to the end of the line is erased. Examples:

PRINT "WHAT'S YOUR NAME":INPUT N$
PRINT "KEY VALUE IN": INPUT A: INPUT B

But note!!
INPUT "YOUR NAME"; A$ is forbidden.

INT

Gives the largest whole number (integer) less than or equal to the given variable. For example:

A=2.1: B=INT(A) B is now=2
B=INT(-1.5) B is now=-2

LEFT$ Used to select a number of characters from a given string starting with the character farthest to the left. You may select a minimum of 1 character up to the maximum number of characters in the string. For example:

```
A$=LEFT$("BASICODE",5)
```

A$ now consists of "BASIC".

But note that C$=LEFT$("BASICODE",0) is not allowed.

LEN Reports the length of a given string. For example:

```
A$="BASICODE": A=LEN(A$)    A is thus 8
A$="":A=LEN(A$)             A is thus 0
```

LET The variable name left of the equal sign is assigned the value of the string or expression to the right of the equal sign. It is not really necessary, since LET A=5 is the same as A=5.

LOG Calculates the natural log of the given variable or expression. For example:

```
PRINT LOG(1)
0
PRINT LOG(10)
2.302585
```

MID$ Takes a number of characters from a string. MID$(A$,X,Y) gives Y characters from A$, beginning with the Xth character. Note that the first character is number 1, so X=0 or Y=0 is forbidden. For example:

```
A$="THIS IS BASICODE"
B$=MID$(A$, 9, 8)
```

B$ now consists of "BASICODE"

NEXT Closing statement for a program loop (see FOR). A NEXT statement must always be followed by a variable. Examples are given under FOR.

NOT Logic negation, only usable on logic variables (see also AND). For example:

```
A=5: B= NOT (A=6)    B is "true"
A=(5=5): B= NOT A    B is "false"
```

ON..GOSUB...
ON..GOTO.. Makes a jump to either a subroutine or a program line. After ON follows an expression or variable. After GOSUB

or GOTO follows a series of line numbers. The expression or variable should be a whole number and determines which line number is to be chosen. You can think of the line numbers as having numbers: if the variable is 1 then the first line number is chosen, if the variable is 2 then the second line number is chosen, etc. But the variable cannot be greater than the number of line numbers given. For example:

```
ON K GOTO 1100, 3400, 1500    K has to be 1, 2, or 3
ON (K-5) GOSUB 600, 700, 300  K has to be 6, 7, or 8
```

OR

Logic OR can only be used with logic variables (see AND). For example:

```
IF A=5 OR B<3 THEN...

C=(A-5) OR (B<3):IF C THEN...
```

PRINT

Prints a variable or a string on the screen beginning at the present cursor position. More than one variable in a PRINT statement must be separated by semi-colons. If it is not desired that the computer should automatically continue with the next line then the end of the instruction must be finished off with a semi-colon. Some computers print one or more spaces before and/or after the number(s) being printed. If you do not want this, then use subroutine 300 or 310. For example:

```
A=5: A$="HELLO": PRINT A;A$
5HELLO
PRINT "HELLO"; PRINT "THERE"
HELLO THERE
CN=3:CT=5:SR=5:GOSUB 310:PRINT"FIVE =";SR$
FIVE=5.000
```

READ

Reads the elements after the DATA statements and gives them to variable(s) which follow the READ statement. More than one variable after a READ statement should be separated by commas. After the RUN command, the computer will read the DATA starting with the lowest line number. All data on that line will be read before continuing with any other DATA lines. **But note:** A numeric variable must read only numbers, a string variable can only read strings. For example:

```
DATA 1, "COMPUTER",3
READ A,A$:READ B or
READ A:READ A$:READ B or
READ A,A$,B
```

REM — This is used for adding REMarks in the program to help other users understand what you are doing. Anything after a REM statement until the end of the line is ignored in BASIC by the computer. But **do not** use a colon after the REM statement as this gives problems with some computers.

RESTORE — This statement resets READ-ing from the first DATA statement in the program. But note that you may not give a line number after a RESTORE statement.

RETURN — Is used to indicate the end of a subroutine. The computer then jumps back to the line after the respective GOSUB statement that started the subroutine sequence. A subroutine should **always** be closed with a RETURN statement.

RIGHT$ — Gives a number of characters of a given string, ending with the last character. The minimum number that can be asked is 1, the maximum being the length of the string. For example:

A$="BASICODE": B$=RIGHT$(A$,4) B$ now consists of "CODE".

But note that A$="PROTOCOL": A=0:B$=RIGHT$(A$,A) is not allowed because A=0.

RUN — Starts the program afresh, while all variables are erased. Note that a line number after RUN is not allowed. For example:

IF (A$="J") OR (A$="j") THEN RUN

Note that RUN 100 is forbidden.

SIN — Gives the sine of a variable which should be given in radians.
See COS for further details.

SGN — Is used to give the sign of a variable, i.e. -1 if the variable is negative, 0 if the variable is zero, and +1 if the variable is positive. For example:

A=5:B=SGN(A) B is now 1
A=-.001:B=SGN(A) B is now -1

SQR — Calculates the square root of a variable or expression, which cannot be negative. For example:

A=SQR(2*50) A is now 10

STEP — Sets the step size in a loop. See FOR.

STOP — Stops the program, but retains the possibility of going further, keeping the same variables.

TAB — Is used in PRINT statements to move the cursor to a specific point on the screen. You can only use TAB to move the cursor further on a line, and depending on the computer, either spaces will be printed or whatever is on the line will be kept. Note that TAB(0) is not allowed. Although most computers start counting from 0, there are those that start at 1. So for this reason it is better to use subroutine 110. For example:

PRINT"A"; TAB(5); "B"; TAB(10); "C" gives:
A B C on some computers and:
A B C on others.

TAN — Calculates the tangent of a given angle in radians. See COS.

THEN — See IF.

TO — See FOR.

VAL — Gives the numeric VALue of a string. But if the string is not purely numeric, the result is not the same on all computers. For example:

A$="1.4E6": A=VAL(A$) A is now=1.4E6
A$="12D": A=VAL(A$) A is now undetermined, A=12 or A=0 being possible.

Summary

Now follows a short summary of the operators allowed in BASICODE-2.

\+ — In the case of numbers of variables this operator adds two numbers or variables together. In the case of strings, two are coupled to each other. For example:

B=1:A=B+9 A is now=10
A$="BAS": B$="ICO": C$="DE.":
D$=A$+B$+C$
D$ therefore is "BASICODE.".

-

Subtracts two numbers or variables from each other. For example:

```
A=10-3-4                A is now=3
```

*

Multiplies two numbers or variables together. For example:

```
A=5:B=3*2*A             B is now=30
```

/

Divides two numbers or variables. For example:

```
A=5:B=100/A/2           B is now=10
```

△

Raises a number or variable to a specified power. For example:

```
A=2:B=16:C=A △ B        C is now=65536
```

=

Logic operator: indicates equality between the two expressions on either side of it. See also AND.
Or: the variable name to the left of the equal sign is assigned the value of the string or expression to the right of the equal sign. Examples:

```
A=(5=6)                 A is "false"
A=4*6                   A is now=24
A$="HELLO"              A$ now contains "HELLO"
```

<

Compares two numbers or expressions and checks whether the one to the left of the operator is less in value than the one to the right. The result is therefore a logic variable. If strings are being compared, then the string to the left of the operator is checked to see if it comes earlier in alpha-numeric order than the string to the right. You can use this operator for alphabetical sorting. For example:

```
A=5: B=(A<7)            B is "true"
A$="HO": B$="HA": A=(A$<B$)  A is thus "false"
```

>

Identical to < except that the test is now "greater than" or "later" in alpha-numeric order.

<>

Checks to see whether two variables or expressions are "un-equal". The result is a logical value. For example:

```
A=(6<>7)                A is "true"
A$="HO": B$="H": A=(A$<>B$) A is therefore "true"
IF A<>5 THEN....
```

<= Less than or equal to. For the function of this operator see <, but substitute "less than or equal to" in place of "less than".

=< Greater than or equal to. For the function of this operator see >, but substitute "greater than or equal to" in place of "greater than".

Note: When using the last three operators the order of the two characters **is** important. For example: A=>5 is **wrong**.

We trust the BASICODE-2 protocol is explained clearly enough, and we look forward to your program contributions. For details of how to contribute, see Chapter One.

Jochem Herrmann

Chapter Five

APPLE II & IIe

BASICODE is an encoding standard created to allow the interchanging of programs written in BASIC, between different brands of computers. The program listing is recorded on tape in ASCII using a special standard called BASICODE. This program makes it possible to read and write standard BASICODE through the cassette recorder interface.

No changes are required to the Apple nor to the tape-recorder connections. You can start right away! The program functions with or without DOS and is independent of the size of the memory. However, at least 16K RAM and Applesoft either in ROM or in the language card are required.

The BASICODE-2 system

To transfer programs from one computer to another, without too many problems, the software must meet some special requirements. Briefly, it amounts to the fact that a number of BASIC statements may not be used, and some others only with certain limitations. Instead you will have a number of subroutines replacing these statements.

Each computer has its own routines. A program consists now of two parts: firstly, a machine-dependent part, different for each computer, and secondly, the main program, which is identical for all computers. The main program always starts at line 1000, and uses the subroutines located below line 1000. The Apple standard subroutines have been included in this program. What these subroutines do, and how to use them, is described in Chapter Three.

Only the main program needs to be transferred, since the subroutines are contained in the BASICODE-2 translation program which you already have. Prior to loading a BASICODE-2 program you must have loaded the subroutines.

How to use this program

By typing **&** and then RETURN you will get the main menu displayed on the screen. Prior to reading in a BASICODE-2 program you can enter the subroutines by typing **1**. You can check the subroutines by typing **LIST**. In this

manner you can also load the subroutines if you want to write a program using BASICODE-2 standards.

Option 2 loads a program from tape, and option 3 saves a program on tape. A detailed description of the load and save options now follows.

The loading option

After typing **2** you will see 'START THE RECORDER AND TOUCH ANY KEY' on the screen. Start the tape with the BASICODE program and as soon as you hear the header tone, touch any key (the header is the tone at the start). With the proper tape level the screen shows 'HEADER FOUND'. The computer loads from the tape, indicated by the flashing characters at the extreme top right of the screen. After detection of the trailer tone, and if all is well, you will see the program listing start to scroll on the screen.

The newly loaded program is being added to what is already stored in the memory. If errors occur during loading, the loading process will not stop at the trailer. Press control C; the screen will now display 'LOADING ENDED' followed by a selection menu. Typing **1** results in a listing of the program read. This listing can be stopped by touching any key.

Pressing the SPACE key will restart the listing, any other key halts it. If there are not too many errors, the program may still be entered by typing **2**. Later on you can correct it yourself. When there are too many errors, it is better to type **3**, return to BASIC and try another load with a different volume setting of the cassette recorder.

If the program is too large for your Apple, the load stops with 'MEMORY FULL' and 'LOADING ENDED'. You can then start to follow the interrupted load sequence just described.

The write option

By typing **3** in the main menu you will be shown the next menu page. Again there are three possibilities: option 1 checks for maximum line length of 60 characters. This includes a space after the line number and a space before the keywords 'TO', 'STEP', 'THEN', 'AND', 'OR', 'GOTO' and 'GOSUB'. These spaces are automatically added by the save option.

The program is also checked for control characters. The numbers of the lines containing errors are displayed on the screen; the lines can be corrected before saving on tape. Option 2 saves the program on tape starting at line 1000, the BASICODE-2 subroutines are not included. Option 3 saves the entire program, which in fact contradicts the BASICODE-2 protocol! However it is a useful contradiction.

Before writing, a table is generated followed by the message 'START THE RECORDER AND TOUCH ANY KEY'.

After touching the key, the recording is started; when you hear the beep the save process is complete. If the memory size of your Apple is too small to store the program and the table, the program will be erased to make space.

In this case, you will see 'BASIC PROGRAM OVERWRITTEN'. If the table size is such that it cannot fit in your memory, you will see 'OUT OF MEMORY ERROR'. The program has been erased, but you can divide it in sections which, after save and load, may be merged. The memory size of most Apples is sufficient, so it will not be very often that you need to do this.

Chapter Six

BBC MICRO (MODELS A & B)

You will find two translation programs for the BBC Micro on the cassette. The first is the Load program and the second allows you to Save programs that you have written in BASICODE-2. Both the programs are written in assembly language for optimum efficiency and the code occupies the memory addresses immediately below the screen memory. When running BASICODE-2, HIMEM should not be changed and the micro should always be in MODE 7.

It is not necessary to load both the Load program and the Save program into memory at the same time. Normally you will be using the Load program to translate programs that you have recorded from Radio 4. Once the programs are translated, you can save them as BASIC programs in the normal way and load them again without using the BASICODE-2 Load program.

The Load program

Once the Load program is in memory, type **RUN**. The screen will then display a reminder of what to type in order to load a BASICODE-2 program.

Remember that before you load in a new BASICODE-2 program, type **NEW**. To load a program, set the tape at the beginning of the leader tone and type in **CALL &7A00**. As the program is being loaded it will be listed on the screen. If there is a problem during the loading you will get the message 'CHECKSUM ERROR'. **LIST** the program to find the error and correct it. If the program loads without problems, type **RUN** to set the program going.

When you want to load a new BASICODE-2 program, press the ESCAPE key, type **NEW** and then proceed as before.

Apart from the main translator, the Load program also contains the BASICODE-2 subroutines. The loading process puts the subroutines below screen memory, following the machine code. When a BASICODE-2 program is being loaded, the subroutines are copied to PAGE and then the BASICODE-2 program is copied from the tape to follow the subroutines.

Once you have successfully recorded the BASICODE-2 programs off your radio and loaded them into your micro, you should **SAVE** them as ordinary BASIC programs.

The Save program

This translation program is provided in case you want to write your own programs and save them in the BASICODE-2 standard. If you have followed the BASICODE-2 programming guidelines given in Chapter Four then you will be able to transfer the programs to a variety of other microcomputers. So, to save your programs in the BASICODE-2 standard, **LOAD** the Save program into the micro and **RUN** it. This will compile the code and place it below screen memory. When the prompt appears on the screen, type **NEW** and **LOAD** your BASIC program. When the program has loaded, type **CALL &7A03** if you want to save the program and the subroutines or **CALL &7A00** if you just want to save the program on its own. Now set up the cassette recorder to **record** the BASICODE-2 program and press RETURN. You will see five seconds of dots on the screen as the leader tone is being recorded, and there is a further five seconds of dots after the program has been transferred, when the trailer tone is recorded.

If there are any lines longer than 60 characters the Save program will mark them with a ':' in the 61st character. After the program has been saved, the number of lines marked is displayed on the screen.

Note: The BASICODE program is generated by converting the BBC BASIC keywords to ASCII code using the **LIST** command. The OSWORD vector is altered to pass the interpreter

```
L.1000,
```

if only the BASIC program is to be saved or

```
L.
```

if the BASIC program and subroutines are to be saved. In changing the OSWRCH vector the program inspects characters as they are sent to the screen by the LIST command. The relevant characters are then also sent to the cassette port.

Chapter Seven

COMMODORE COMPUTERS

The instructions in this chapter apply to the following Commodore computers:

CBM-3008,3016 and 3032 (with new ROM's)
CBM-4016 and 4032
CBM-8032 and 8096
Commodore 64
PET- 2001 (with old ROM's)
VIC-20

Introduction

The *reading* and *writing* of programs in the BASICODE-2 standard is possible via the use of the translation programs supplied with this handbook. Each of the computers mentioned above has a separate translation program specifically designed for that model. The translation program checks that it is being loaded into the correct model, and if not, it will automatically stop and give a relevant instruction. The working of the translation program is slightly different with each model. However, as far as the user is concerned, the instructions which follow apply to all Commodore models. If there are comments specific to certain models, these appear on the screen.

There is no need to make any changes to your computer to use BASICODE-2, since the programs work with the standard hardware supplied by the manufacturer.

Description of the accompanying cassette translation program.

Proceed as follows: first **LOAD** the specific BASICODE-2 translation program for your model of computer. When this is complete, type in the command **RUN** whereupon a menu will appear on the screen. You now have a choice of two functions:

1. Preparation of the Reading program to read a BASICODE-2 program from tape.
2. Preparation of the Writing program to put a program in BASICODE-2 onto tape.

Once you have chosen a number, the selected program is prepared in the top 512 bytes of the available computer memory. While this is happening, you will see a counter displayed on the screen which follows the progress of this preparation process. When the counter reaches 0, the job is complete and the relevant translation program is ready. On the screen you will see which SYS instructions you can call up. We suggest you write these down on a piece of paper, since you can use these instructions at any time until you switch the computer off.

The chosen translation program is in fact taken from the *strange* texts in the DATA lines and converted into machine language. This is then placed in the top 512 bytes of the memory, and the available space for BASIC interpretation is therefore reduced by the same amount.

After noting the SYS instructions, if you then press the SPACE key, the menu appears again. If you wish to load the other translation program at this point, you may do so. However, if you are using either a VIC-20, or a PET with limited memory capacity, then it is advisable to load the second translation program only when it is needed.

If you have already listed the program as read from the cassette, you may have been surprised by the first 50 lines. These are the fixed subroutines described in Chapter Four. These are used in all BASICODE-2 programs, so be careful that you do not erase them by accident with the use of NEW. You will probably need these routines at a later stage.

Reading BASICODE-2 programs

With the first SYS instruction displayed on the screen after setting up the Reading program, you can start up the BASICODE-2 Reading program. The memory is then cleared from line 1000 onwards, where the BASICODE-2 program will be stored. If you are using either a VIC-20 or a PET, then the statement 'READY' will appear on the screen, after which you should type in the second SYS instruction. With the Commodore 64 and CBM 3000 and higher the program automatically goes further without a second instruction. The screen now displays an instruction to press the PLAY button on the cassette tape-recorder. As soon as this is done, the statement 'SEARCHING FOR BASICODE' or simply 'SEARCHING' appears. The computer is now looking for the header tone on the tape. Anything which does not resemble a header tone (such as speech before it) is ignored.

When the header is located, the computer will indicate 'FOUND' and will try to load the BASICODE program. If you are using a Commodore 64 computer, then at this point the screen will go blank. This is normal. On the other computers you will see flashing characters at the top of the screen. This is text being read from the tape, and although it goes very fast, you

may be able to follow some of it. As soon as everything has been read, the recorder will stop and the computer gives the statement 'READY'. In the case of the Commodore 64 computer, the screen will now return to normal. You can now **LIST** the program if you wish, **RUN** it, or otherwise adjust it.

If something has gone wrong during the reading process, then after the recorder has stopped, the statement 'LOAD ERROR' will appear. On the PET this is simply displayed as 'ERROR'. Despite this though, everything has been read from tape, so by listing you can check whether the program can be corrected.

Errors such as 'PRUNT' instead of 'PRINT' should show up clearly. If for any reason you wish to interrupt the loading process, press the STOP key which will stop your tape recorder. On some computers you may need to press the STOP and RESTORE key. The section which has been read from tape, up to the interruption, has been stored. You can then type further lines in the normal way.

If desired, the loading procedure can be started using the second SYS instruction which appears on the screen. In this case, everything which is already in the memory remains untouched: the computer simply puts what it reads from the tape **after** what is already in the memory. So be careful that line numbers are in the right sequence, if you are doing this deliberately, otherwise you will have problems running the program!

The Reading program checks to see if there is sufficient memory during the reading process. If this is not the case, the loading process stops and the statement 'OUT OF MEMORY ERROR' appears. The program on the tape is then too large for your computer memory.

The Reading program makes use of the (first) cassette buffer during reading of a BASICODE tape.

Because Commodore computers work with upper and lower case letters in a non-ASCII way, the Reading translation program offers two methods of processing lower case letters from the tape. One possibility is that lower case letters on the tape are converted to 'Commodore' capital letters. The other alternative is that lower case letters will be converted to 'Commodore' lower case letters, and capitals into 'Commodore' capitals. The latter is performed only when letters are in strings, i.e. between inverted commas.

To inform the Reading translation program how to proceed, you must set your computer either in text mode or in graphics mode. The computer will then adjust the incoming program to the mode you have selected. If you set the computer to recognise both upper and lower case letters, incoming programs will be presented with both types as appropriate. However, this

may not always be the most desirable form. If it is not, then you can try again using the graphics mode.

Writing in BASICODE

Like the Reading program just described, the Writing translation program is designed to resemble the ordinary SAVE as much as possible. You can start the Writing program using the SYS instruction which appears on the screen after you have loaded it. Before you can save in BASICODE of course, you will need to load the program that is to be saved in the BASICODE format. When this is done, you can then select the suitable SYS instruction and the computer will ask you to press the **record** buttons on your cassette recorder. Once you have done this, the computer will then start to save the program on tape using the BASICODE standard. In the case of the Commodore 64 computer, the screen will go blank during this saving process. In the case of other computers, you will see flashing characters in the top left hand corner of the screen, indicating what is being saved on tape. After the last character, 5 seconds of trailer tone is recorded. The recorder then stops and the statement 'READY' appears. On the Commodore 64, the screen then becomes active again.

Points to bear in mind!

In order for this system to work, you must follow these guidelines for BASICODE:

1. A program line may not be longer than 60 characters in length. If you try to make it longer, then the Writing translation program will stop the saving process when you try to save your BASICODE program. It will give the statement 'LONG LINE ERROR IN...' followed by the line number in question. **LIST** this line number, make it shorter, or split it in two using two line numbers. Then you can use the SYS instruction to re-save the BASICODE program.

2. Those characters which you see in reverse-field on the screen, such as clear-screen and all graphic characters are **forbidden** in BASICODE. If they appear in your program, then as you try to save it in BASICODE, the process will stop. The statement 'ILL.CHR.ERROR IN ...' will appear, together with the line number of the offending character. ILL.CHR. simply stands for ILLEGAL CHARACTER.

If you decide to save in BASICODE using the graphics mode, then all letters in your program will be saved as upper case. If you use the text mode, then upper and lower case letters that appear in strings will be saved without any change. PET owners can thus save a program using BASICODE knowing that it will appear in the correct form on other Commodore computers as well.

There is a possibility that the statement 'ILL.CHR.ERROR IN ...' may appear when there does not seem to be anything wrong. It may be that you have made a space by pressing the SPACE bar while in the shift mode. Simply re-type the spaces, making sure you use the SPACE bar in its normal setting.

3. Remember that the trailer tone lasts five seconds, so make an allowance for this on the tape. If the tape is too short, the computer may try to put this onto the plastic leader at the end of your tape.

You should see now that the first SYS instruction means that only line numbers 1000 and above are saved on tape, following the BASICODE-2 protocol. The second SYS instruction gives you the option to save everything.

Chapter Eight

SHARP MZ80A

The BASICODE translation program for the Sharp MZ80A is clearly identified on Side 1 of the cassette tape. It is called BASICODE-2 5510.

Do not use lines 1 to 1000; these are reserved for a subroutine handling command, LOAD/2. This command ensures subroutines necessary for BASICODE-2 are added to the program being loaded from cassette.

You should remember this when saving a program in BASICODE-2. The command for this operation is **SAVE/B1000-**.

Using the command **SAVE/B1000-**, lines 1000 and above will be saved onto cassette in the BASICODE format. But before the 'RECORD PLAY' instruction appears on the screen, the computer will check the program you wish to save. If there are lines that are 60 characters, or longer, in length, then that line will be printed on the screen. The translation into the BASICODE format will stop. You should now make the line shorter (by splitting over two line numbers), and then give the command **SAVE/B1000-**.

You do have the option to save the entire program on tape, if desired. If you use **SAVE/B** you will achieve this. All the usual commands you are used to remain unaffected by the BASICODE system. So it is also possible to use the LOAD and SAVE functions if you wish.

We wish you success with the BASICODE-2 system and look forward to your contributions to the broadcasts. Details of how to participate are given in Chapter One.

Chapter Nine

SINCLAIR ZX81

Introduction

The Translation program on the cassette is the latest addition to the BASICODE system although the ZX81 has been available for some years. There have been two major problems to overcome:

1. Memory capacity: Early models did not have enough space for the translation program. This problem has now been largely overcome with the availability of more memory.

2. The Sinclair BASIC: The type of BASIC used in the ZX81 is not immediately comparable with the form used in BASICODE 2. The Sinclair ZX81 is not able to cope with more than one statement per line whereas we allowed this in BASICODE to avoid long programs.

How to use the Translation program

Load the program on Side 1 of the cassette for the ZX81. Use **LOAD ""** to do this. The program consists of two routines, the Reading routine designed to make the ZX81 read the BASICODE program off the radio, and the Translation program to convert what has been read into a form that can be dealt with by the ZX81.

After loading the program, the routine should present itself and the ZX81 will give an error message '7/9090'. **This is normal.** Your ZX81 is now operating the FAST mode which is essential for the Reading routine.

You are now ready to load a program recorded off the radio. Remember to keep the recording levels as high as possible, without distortion. Having got a recording, wind it back to the start and cue up the cassette so that you can hear the start tone. This is the steady tone that starts all BASICODE programs.

Now type in **RAND USR 16618**
Start your tape recorder in the playback mode and press (NEWLINE). The BASICODE program should now start loading. This process will stop if:

a) You press BREAK
b) The memory is full
c) An ETX CHR is loaded.

Now type in **LIST 999** and on that line you should see a REM statement. What comes after the REM has been loaded by the computer.

If the BASICODE program hasn't loaded correctly, it may be due to the volume control on your tape recorder being set too low. This will result in meaningless information after the REM on some or all lines. If the whole program is nonsense, start again from the beginning and try a different volume setting. If only the last part has gone wrong, there is no reason why you shouldn't start loading half-way through the BASICODE program rather than winding it back to the start. It may take some experimenting at first, but you should find a point where the process goes smoothly.

Now that we're in the position of having loaded the BASICODE program, it has to be translated into a form that the ZX81 recognises as an ordinary ZX81 program. You start this process by typing in **RAND USR 17165** (NEW LINE)

The Translation program now starts looking at the BASICODE material, line by line. A statement is replaced by a token and when it comes across a second statement on one line, it generates a new line for this second statement. LET and GOTO statements are inserted where necessary, and when this is complete, the REM statement is erased. You can therefore expect the following result.

Original After Loading From Radio		**New Version**	
1210	REM A=400 : B=2	1210	LET A=400
		1211	LET B=2
1220	REM LET A$="ABC" : B$="FGH"	1220	LET A$="ABC"
		1221	LET B$="FGH"
1225	REM IF A > B THEN A=B: PRINT "A WAS <> B" : 1300	1225	IF NOT (A> B) THEN GOTO 1230
		1226	LET A=B
		1227	PRINT "A WAS <>B"
		1228	GOTO 1300
1230	REM PRINT A : PRINT B	1230	PRINT A
		1231	PRINT B
1300	REM PRINT "END VB", A$	1300	PRINT "END VB", A$

You can test this system out by using **RAND USR 16618** (NEWLINE).

You don't need a cassette, but you will need to press BREAK. You can now

type in a few test lines and have them translated by entering **RAND USR 17165**. If you press BREAK during the translation process, the computer will stop translating and you will once again see the screen. You can go back to translating by using **RAND USR 17165** again and the computer will pick up where it left off.

Adjustments

Once you have loaded and translated a BASICODE program you can now try to RUN it. Some programs broadcast in *The Chip Shop* will run immediately. But there will also be a few that need some changes to be made. These changes will become clear during a test run. Type in **RUN** (NEWLINE) and note any error messages that come up. Check the line the computer refers to. Remember that unlike other computers in the BASICODE system, the ZX81 does not like string variables and FOR-NEXT variables which consist of more than one character, e.g. 'IN$' and 'FOR IN=0 TO 10'. You will have to change these to variables of a single character instead.

Suppose you choose Z$ to replace IN$ in the program. That is fine, as long as Z$ doesn't appear elsewhere in the BASICODE program with, of course, another meaning. So it's a good idea, once you have chosen your single character variable to check that it doesn't exist in the BASICODE program. To do this type in:

RUN 400,-"1" (for search),-**"Z$"** (for word)

This simple routine now scans through the entire program and gives the message 'FOUND' if it does find, in this case, Z$. If you **don't** get this message, then it is safe to replace IN$ in the original by Z$. You do this by typing in:

RUN 400,-"2" (for replace), **-"IN$"** (for word), **-" Z$"** (replaced by).

Note: The space before the Z$ is deliberate. Both the variables must be the same length, so when you put the single character in, you must leave that space.

DATA-RESTORE-READ process:
In most cases you can replace DATA by DIM D$ (....) LET D=1. You must work out the dimensions of D$. The number of DATA elements is the first number, and the **length of the longest element** is the second. In addition, D$ must be filled with a number of LET statements. RESTORE thus becomes LET D=1, and READ A$ becomes LET A$=D$(D) LET D=D+1.

In most BASICODE programs, programmers tend to put DATA into table form, and use statements elsewhere in the program to manipulate it.

Before

```
2000  FOR N=1 TO ...
2010  LET READ M$(N)
2020  NEXT N
        .
        .
        .
3000  LET DATA "MO", "TU", "WE"
```

After

```
2000  GOSUB  3000
3000  REM M$(1)="MO":
      M$(2)="TU":M$(3)="WE"
      RETURN
```

We don't need lines 2000-2020 in the above example. Instead, the data is rewritten in another form in line 3000. If we now start up the Translation routine using **RAND USR 17165**, a number of neat lines will be made from line 3000, each with its own LET statement.

Note: Remember that some other computers allow the table index to be 0, i.e. LET A$(0)="VB".

All we have to check now are LEFT$, MID$, and RIGHT$.

```
LET A$="ABCDEFG"
LET B$=LEFT$(A$,3)      LET B$=MID$(A$,3,1)      LET B$=RIGHT$(A$,4)
    B$ is now="ABC"         B$ is now="C"            B$ is now="DEFG"
```

But we could also write B$=LEFT$("ABCDEFG",3), and so on, and get the same result. So for the ZX81 this would become:

```
LET B$=A$(TO 3)    LET B$=A$(3 TO 4)    LET B$=A$(LEN A$-4) TO)
```

Line number 998 is a very special line. REM COPY COPY. This is the terminator line and everything after the terminator line with a higher line number (i.e. larger than 998) is erased. You can change this line number if you wish.

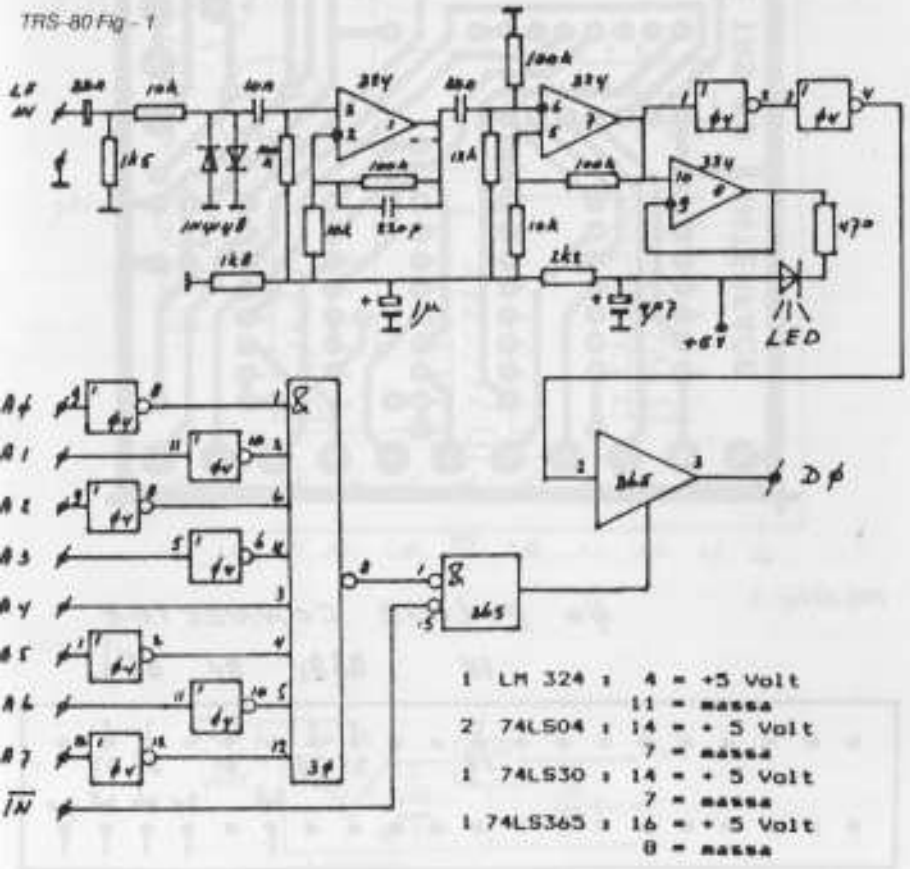
Note too that the Reading routine will scrap any lines above 9990, and line numbers will move in steps of 10.

Chapter Ten

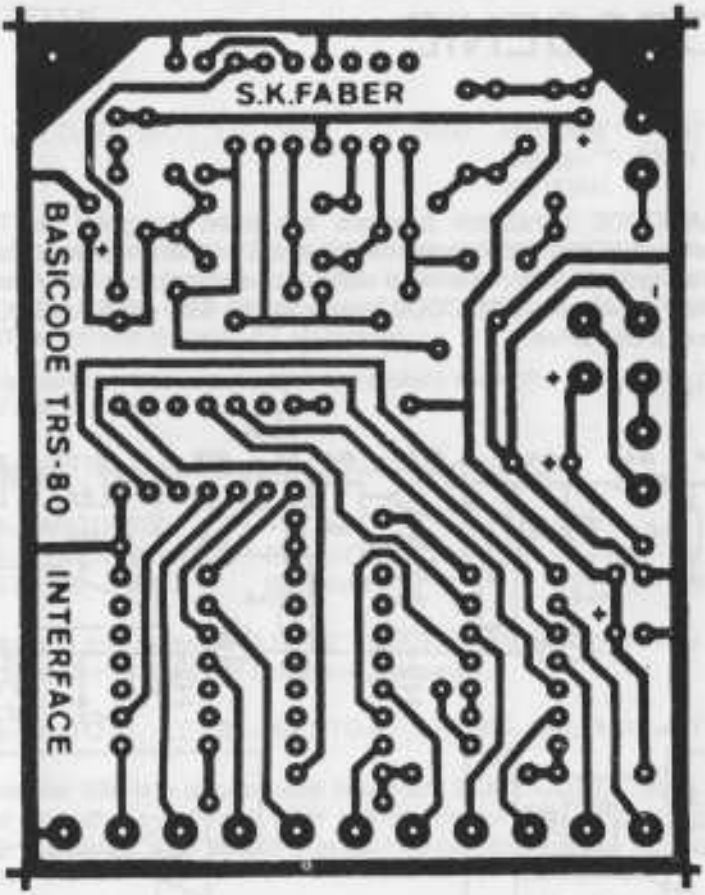
TANDY TRS–80 MODELS I & III VIDEOGENIE

The BASICODE translation program has been designed for TRS–80 Models I and III and the Videogenie. However, you will need to construct a small interface. This is not all that difficult and should not deter you from going further with the BASICODE idea. You can also use BASICODE with or without a disk drive.

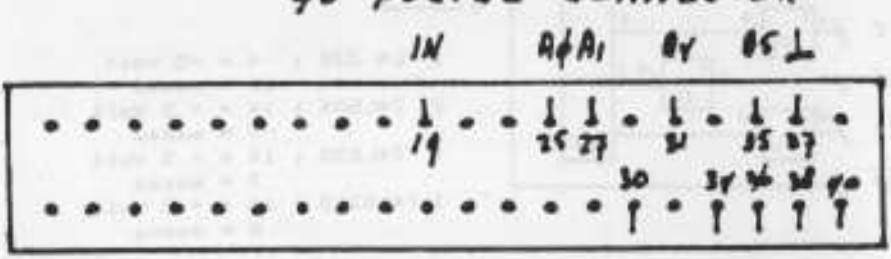
TRS-80 Fig - 1



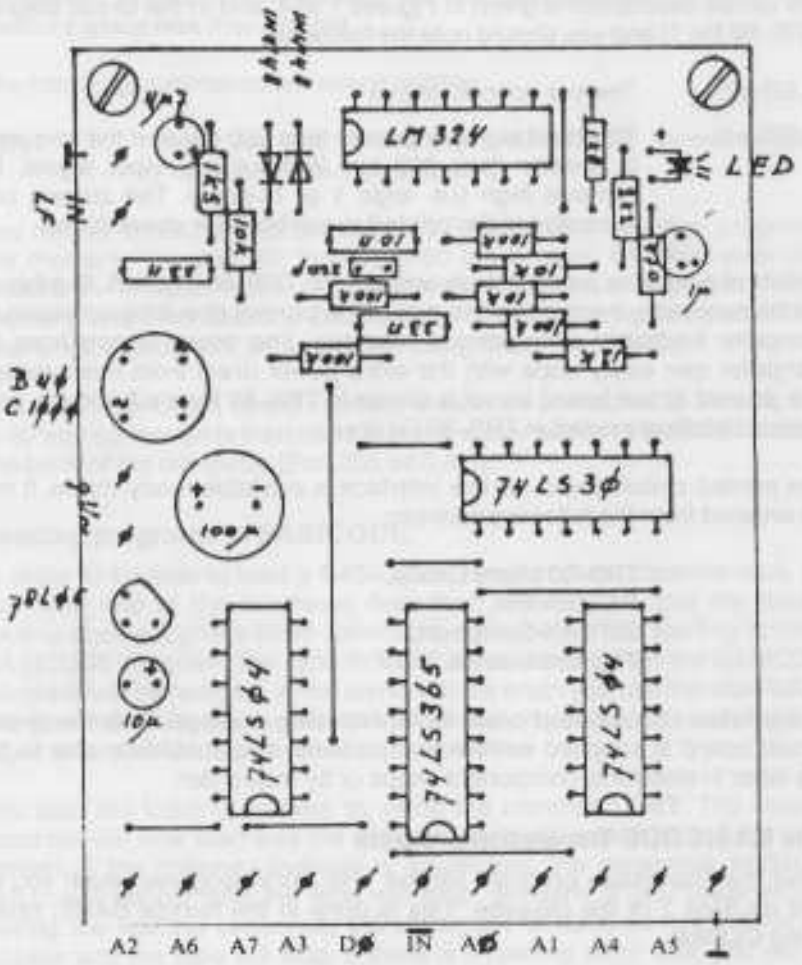
TRS-80 Fig - 2



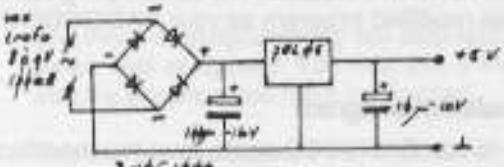
TRS-80 Fig - 3



TRS-80 Fig - 4



TRS-80 Fig - 5



The extension circuit board

The circuit description is given in Figures 1 to 4, and in the circuit diagram (TRS–80 Fig 1) and you should note the following:

– LED off: There is no input signal

– LED on: The input signal is greater than 200 mV and the frequency is greater than 300 Hz. Without any input signal, DO remains high (i.e. logic 1 or hi-state). The current consumption of the printed circuit board is about 30 mA.

Details of a suitable power supply are given in TRS–80 Figure 5. But this will not be necessary if you decide to mount the printed circuit board inside the computer keyboard or expansion interface. The power supply from the computer can easily cope with the extra power drain from this interface. The printed circuit board layout is shown in TRS–80 Figure 2 and the component details are noted in TRS–80 Figure 4.

The printed circuit board for this interface is available ready-made. It may be ordered from the following address:

TRS–80 Users Group,
P.O. Box 551,
2070 AN Santpoort,
The Netherlands.

The printed circuit board costs £2.00 including postage. **Only** the printed circuit board is supplied **not** the components. You should be able to find the latter in electronic component shops or by mail order.

The BASICODE Translation program

Load the Translation program for the TRS–80/Videogenie which you will find on Side 1 of the cassette. This is done in the normal BASIC mode, using **CLOAD**.

After you have typed in **RUN**, a menu will appear asking you which computer model you are using. If you follow the instructions on the screen, you should end up with the Translation program ready for use. Once you have done this, **STORE** the adjusted Translation program either on disk or cassette. Use this modified program as your BASICODE–2 translator from now on.

Using the Translation program

When you wish to use BASICODE again, load the modified program back into the memory using **CLOAD** or **LOAD**, as appropriate. After you have

RUN it, you will see the statement 'READY' appear on the screen. The Translation program is now ready for use, and you have 1019 bytes of memory space **less** than normal.

The following commands are now available:

- GET: You use this for loading a BASICODE format program from tape into the computer.

The normal BASIC is unaffected by the BASICODE translation program in the memory. You can still load TRS-80 tapes using **CLOAD**, save using **CSAVE** in TRS-80 format, etc. **But note:** If you have a program in the memory, and then load the BASICODE Translation program, you will lose your original program. So save it first using **CSAVE** or **SAVE.**

The extension circuit board interface should be connected to Port 16, bit 0. A 40 way connector is then used to make a connection to the *bus* located at the back of the computer (Port 255 bit 0 & 1).

Loading programs in BASICODE

In order to be able to load a BASICODE program from cassette tape, you will need one of the interfaces described above. Note that the loading routine does not give a NEW command, so if you are just starting to load a BASICODE program, first type in **NEW.** If you do not, then the BASICODE program will be merged, in the same way as when you use the disk-BASIC merge commands. Remember this, especially if you want to put programs together and eventually re-number the result.

You start the loading process by using the command **GET**. The cassette recorder will now start and the program lines will appear at the top of the screen. If the memory capacity is insufficient, the statement 'MEMORY FULL' will appear and the tape will stop. Similarly, if you press BREAK during the loading sequence, the statement 'BREAK IN PROGRAM' will appear and the tape will stop. If there is a reading error, then you will see 'CHECKSUM ERROR' appear. But, usually, no statement will appear, and the recorder will shut off when it reaches the trailer tone on the tape.

You now have the following options:

BREAK – KEY: The input of programming will stop and the ▷ prompting cursor will re-appear. You use this if there is a problem with the program you have just loaded.

ENTER – KEY: This line will be accepted into the memory.

BSPACE – KEY	This line will not be accepted into the memory, for example if there is a fault in the line-number.
SPACE – BAR	If you press this (or the ENTER key) repeatedly, the program is read in at full speed. This is useful when the program contains no faults.

The pressing of ENTER or BSPACE automatically brings up the next line of the program for inspection.

If there are control characters in the program (other than CR or ETX), or there is a reading error (stopbit not a "1" or bit 7 not a "1"), then the character in question will be underlined using a '-'. You can use this facility if you are trying to rescue a program from a very poor quality tape, in conjunction with the edit facility.

You can also load from a tape more than once, adjusting the playback level to different settings until a complete program is obtained. If the 'MEMORY FULL' statement appears, some of the program will have been lost, but what has already been put into the memory remains there, and can be used.

The saving routine for BASICODE

You do not need the special interface to save a program in the BASICODE format. You can begin the saving process by using the command **PUT**. The program is then saved onto cassette in the BASICODE format and the tape will start automatically. If the statement 'MEMORY FULL' appears, then the program was not completely stored in the memory, due to lack of space. What was stored will be put onto tape. You can only interrupt the saving process by pressing the reset button at the back of the computer. Note that control characters in the text, such as '/' will not be destroyed, so make sure that they do **not** appear in your BASICODE programs, or you may put other brands of computers into tilt. Read Chapters Two or Three for details of the guidelines to follow with the BASICODE. Note that the command GET will not affect other brands.

Chapter Eleven

THE FUTURE

Since details of BASICODE spread outside The Netherlands, the Dutch have received many thousands of letters. It became clear that the most popular computers in one country may not even exist in another. The idea of putting the entire BASICODE-2 protocol into Chapter Four is not only to explain it to users of computer makes already in the group, but it is hoped that experienced users of other makes will study the protocol and create translation programs for their computer. If you succeed, please **do not keep it to yourself!**

We will publish updates to this handbook with details of translation programs for other brands. Authors are always credited, and details of how to take part are given in the first chapter of this book.

BASICODE has grown from a simple idea for exchanging software, back in 1979, to a standard in regular use in many parts of the world. We hope you will contribute to make it even better known.

BASICODE 2 is just like Esperanto for microcomputers. It enables micros of different makes to 'talk' to each other. It is simple to use and has established itself as an international standard which can be used on some of the most popular home micros including:

Apple II and IIe
BBC Models A and B
Commodore 64, Pet and Vic 20
Sharp MZ80 A
Sinclair ZX81
Tandy TRS-80
Video Genie

BASICODE 2 will allow you to use programs broadcast in the BBC's radio series *The Chip Shop* and also to exchange programs with owners of these micros.

How does it work? The **BASICODE 2** kit comprises a handbook and a cassette. On one side of the cassette are the translation programs for the micros listed. The translation program when loaded will 'teach' your micro **BASICODE 2**. Once it 'knows' **BASICODE 2**, your micro can translate and use programs written for these other micros. You can also record and use the programs broadcast on Radio 4. On the second side of the cassette is a selection of games and scientific and educational programs to start you off with **BASICODE 2.**

ISBN 0-906965-14-4

BASICODE 2 kit £3.95

BSS Broadcasting Support Services

Published and distributed by Broadcasting Support Services for *THE CHIP SHOP*, BBC Radio 4
Cover designed by BBC Information Division 1984
Text designed by Adco