DAI

PERSONAL COMPUTER
HANDBOOK

PART I

INTRODUCTION

This manual is divided into two parts. The purpose of the first part is to allow you to use your new machine as soon as you have it set up as explained in the next few pages.

This part of the book assumes you are a complete novice to programming and will guide your first BASIC (Beginners All-purpose Symbolic Instruction Code - a programming language) steps while introducing you to features that are unique to the DAI Personal Computer and as such cannot be used straight off even by an experienced programmer.

On the other hand the purpose of this manual is NOT to give you a full course on BASIC programming.

The authors hope that after working through this book and having had but a hint of what you can make your computer do for you with proper programming, you will feel stimulated enough to want to learn more by studying one of the many available books on the subject (see Appendix B) to which this manual may in no way be considered a substitute.

The second part of the manual contains the information on the DAI implementation of the BASIC language to which you will often need to refer when programming on this machine.

Writing a manual that has to cater for a wide variety of users is no easy task. There is a danger of pleasing no-one by trying to please everyone. Please excuse us if we seem to be too pedantic at times and too superficial at others.

Indeed, if you have any suggestions that might help us improve this manual, please let us know.

ON KITCHEN FLOORS AND TV TUNING

The first thing to do upon arriving home with your new machine, is to find a quiet place near a power outlet and possibly a table. (Although your DAI will work just as well on the floor in the kitchen, this might prove a bit uncomfortable for you!)

In the carton where you have already found this manual you should also find the computer (the interesting-looking white box reminiscent of a typewriter), and three cables equipped with plugs.

Connect the coaxial cable to the VIDEO output on the back of the DAI and to the aerial input of the television set you intend to use as VDU (Visual Display Unit). The latter may be any model, b/w or colour, capable of receiving UHF, though you would be well advised to use a colour set in order to make use of one of the most impressive features of the DAI : COLOUR GRAPHICS.
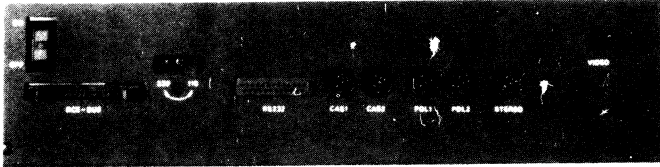
Connect the black power cable to the socket marked 220 (making sure first of all that the voltage selector is set to 220) on the back panel and to the socket in the wall.

The third cable is the cassette interface (computerese for connection). You simply plug that in the outlet marked CASS 1 on the computer and in the MICRO and EAR sockets respectively on your cassette recorder.

If you don't have a cassette recorder (nor any other tape recorder) in the house just now, DON'T PANIC you can still go through the whole manual (or most of it anyway) without one. But eventually you'll need a tape recorder to SAVE on tape the programs you'll have written and to LOAD (from tape into computer memory) both those programs and/or programs written by other DAI users as well as

commercially available programs. It needn't be an expensive model, but try to find one with a tape counter (invaluable for locating programs on a cassette).

SWITCHING ON

At this point we'll assume that you have connected all cables as required and are sitting in front of the DAI. The next thing to do is to switch on the TV and let it warm up. Now switch on the DAI (the red switch on the back) and check that both the switch and the small green lamp (on the right of the Keyboard) are lit up. Finally, you must now tune your TV set to UHF channel 36 to receive the signal from the DAI. When correctly tuned, you will see, on a green background, in large white capital letters centred in the top half of the screen the words:

DAI PERSONAL
COMPUTER

If instead of the above, you see a grey screen with in the top left-hand corner:
BASIC   V1.0
* _

that's because you have inadvertently done what we would have asked you to do in a moment, that is, pressed a key on the DAI after switching it on and before tuning the picture. In this case, so you can see the message on the green background and feel that you are following the instructions step by step, you must simply switch the computer off and on again.

There! Now the screen is green and displays the right message.

If you NOW press any key, you will get:

BASIC V1.Ø

*_

(On some TV sets, you might have to adjust the vertical and/or horizontal size of the picture, in order to see that properly, so please do that before blaming your PC should you not get the right picture.

READY

Your computer is now ready to accept you commands in the BASIC language.

We call the asterisk you can see under the B in BASIC the

PROMPT, because the computer will display it on a new line on the screen, following whatever may already be on the screen, whenever it wants to tell you - "prompt" you - that it is ready to accept your commands. Notice at this point, that immediately after the asterisk or PROMPT there is a blinking underline symbol. This is known as the CURSOR and shows where the next character will appear when you type it in.

Every time you see the asterisk and the blinking cursor, as the last line on the screen (not necessarily on the bottom of the screen), you are in CONTROL of the machine. That means that the computer is not executing any program and it is waiting for you to type a command or start typing in a program.

ON HAMMERS, SYNTAX AND A FEW OTHER THINGS

Before moving on, please take note of the following. It is VERY IMPORTANT.

There is no way that you can harm your computer by playing at the keyboard, short of typing with a hammer or TRYING TO REMOVE THE KEYS from their holders, either of which will cause permanent damage to your computer! Therefore, you can try out anything else you like just to see how the computer responds.

However, whether you type random letters or perfectly

correct English sentences, chances are that your only
reward (if and when you press the key marked RETURN,
explained later) will probably be a
SYNTAX ERROR
message, or some other error message, clearly indicating
that your computer only "understands" BASIC and not plain
human language.
This manual was written with the intention of giving you
the first elements of the language your computer
understands. Just as with any other language, human or
not, practice is the only way to learn.
We therefore urge you to TRY OUT ALL THE EXAMPLES given
in the following pages and even make up more of your own.
When trying your own, don't give up on the first ERROR
MESSAGE you get, you'll have to get used to seeing a lot
of them: the programmer who never gets an error message
has yet to be born!

    In order to make it absolutely clear when we want
you to type something in your DAI from this manual, the
text will be standing on a separate line and an arrow on
the left margin will point to it. You will then type the
text in EXACTLY as shown, including the spaces where they
appear in the manual. Very often SPACES play an
important role in the SYNTAX of the BASIC language, just

asi nEng lish. (OK, so you didn't have any trouble reading that, but remember that computers are very DUMB!) Whenever you get a SYNTAX ERROR after typing in one of our examples, the first thing you should check is that you have respected the spaces printed in the manual.


INTRODUCING THE DAI KEYBOARD

In order to communicate with your DAI you'll have to get familiar with its keyboard.

It is very much like any typewriter keyboard, but it also has a few keys that are new even to an experienced typist and some that do not perform the exact action a typist would expect.

If you have never used a typewriter before you might be less at a disadvantage than you think. Indeed, programming is not something you can really do at touch typing speed. On the other hand, when using a computer keyboard typists will initially have to watch out for mistakes caused by a habit they might have of typing l's (as in love) instead of 1's (as in 12345), and capital O's instead of ∅'s (zeroes, which on computer. keyboards look like funny capital O's slashed down the middle). In any event, for typists and newcomers alike here is a

list of the main keys not to be found on a regular typewriter:
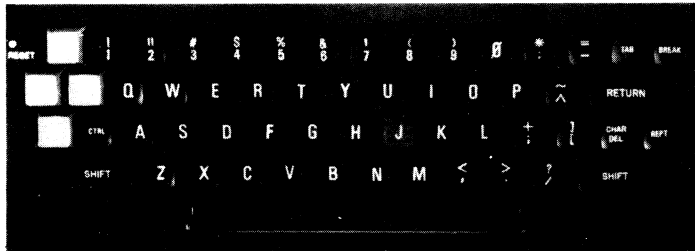
1.   the four grey CURSOR control keys in the upper left-hand corner of the keyboard;

2.   the CTRL key, just above the left SHIFT key;

3.   the RETURN key, above the CHAR DEL key;

4.   the CHAR DEL key, just above the right SHIFT key;

5.   the BREAK key, in the upper right-hand corner and

6.   the REPT key, to the right of the CHAR DEL key.

Moreover, the following keys perform a function as well as printing the symbol shown on them:

* is the MULTIPLICATION sign

/ is the DIVISION sign

< means LESS THAN

> means MORE THAN

! means TO THE POWER OF.

Finally, please note that in order to allow computers to distinguish between zeroes and capital O's the former are represented by the symbol Ø (a capital O with a slash).

Let's see what all these various keys do:

1. CURSOR control keys. They perform their main function in the EDIT mode, which will be explained later. The left arrow key is also used in the UTILITY (see Handbook). They normally have no other function, unless you decide to use them in one of your programs to perform any function you wish to assign to them (e.g. in games , where you wish to input the direction to be imparted to a moving object, or whatever).

2. CTRL key. This allows you to select whether all the alphabetic characters you type in after pressing it will be displayed as UPPER case (i.e. capital letters) or LOWER case.

When you first switch on the DAI, or after you press the RESET button (see later under point 5), any text you type in will appear on the screen as upper case, since that is

what you will normally need in order to write BASIC
programs.

However, when writing programs, there will be times when
you'll want the computer to display a few lines or a few
pages of text to serve as explanation for what the
program does or for whatever other reason.

Since the DAI can also display lower case letters, that
is what you'll presumably want your messages to be
written in.

There are two ways to get lower case letters on the
screen:

a) type the desired letter(s) while holding down one of
the two keys marked SHIFT on either side of the keyboard
or

b) press the CTRL key once and all letters typed
thereafter will appear as lower case.

Chances are you will choose option b) in which case the
keyboard will act just as a typewriter keyboard, giving
upper case letters when holding down the SHIFT key while
typing the letter(s) to be capitalised.

To revert to getting upper case letters without SHIFTing,
you will press CTRL once again and so on.

Notice that CTRL does not affect the non-alphabetic keys
which will print the upper symbol only when pressed while
holding down the SHIFT key, regardless of the number of

times you might have pressed the CTRL key. For example,
to type a ? (question mark), press the SHIFT key while
pressing the key which has the ? sign on top of the
sign. Try it.

A few minutes' practice will fix all the above in your
mind, so why don't you type a few lines, trying out the
SHIFT and CTRL keys. When you're finished you will
probably have a lot of junk filling up the screen.

Although when you have used up the 24 lines that can be
displayed at one time the DAI will still let you add more
lines at the bottom, by moving up (scrolling) the text so
that one line vanishes from the top of the screen for
every line added at the bottom, it won't hurt to tell you
right now HOW TO CLEAR THE SCREEN.

Press the key marked RETURN to make sure you're on a new
line (ignore any error messages this might cause),   then
type in:

➤ ?CHR$(12) ⮐

and press the RETURN key. Is the screen clear now? For
the moment just note how to clear the screen, without
trying to figure out how it is done. Try filling the
screen with garbage and clearing it a few more times to
get familiar with the procedure.

3. RETURN KEY

We've always felt that a more appropriate description of the RETURN key would be THE KEY OF NO RETURN. Indeed, once typed there is no way you can take back what you just told your computer to do (you soon lose count of the number of times you wished you hadn't pressed RETURN quite so soon!).

A word of explanation:

When you're typing something into the computer, there's no way it can guess when you have finished, unless you signify it in some way. The RETURN key is the way you tell the computer that it may now act upon what you have just typed in. Before and until you press RETURN you can change you mind a million times, but once you've pressed it...

It will take you some time before you get used to pressing the RETURN key every time you have finished typing in a command or a line of a program, as explained later. We shall remind you to do it by printing this special symbol ⏎ every time you must press the RETURN key and we shall also remind you by telling you to do it in plain English.

4.   CHAR DEL Key. Supposing you wanted to type
COMPUTET ...oops, that's where the CHAR DEL key comes   in
handy: press it once and COMPUTET becomes COMPUTE   then
type R and the end result is COMPUTER.   By repeatedly
pressing CHAR DEL you can even erase the whole of the
current line if you wish to take it all back, as long as
you haven't pressed RETURN.
The EDIT   mode permits you to correct any spelling
mistakes or syntax errors in the text of a program in a
much more convenient way, but we'll discuss that later.

5.   BREAK key.

When you first switch on the computer and   the
machine is executing no program (except, of course, the
program contained in ROM (Read Only Memory) that   **allows**
the microprocessor at the heart of the system to   make
sense of what you type in, and allows you to program in
BASIC) we say that you are in CONTROL of the computer.
When you type in a program and instruct the computer to
RUN it (meaning to execute the instructions contained in
the program), control of the machine passes to   the
program and you can no longer type your commands at the
keyboard.   The way to stop a program and regain   control
of the computer is to press the BREAK key.
However, sometimes th computer is so busy doing

something, that it cannot "feel" that you're pressing the BREAK key and will therefore not stop. When this happens (almost certainly because of a programming error or "bug" as we say, in your program), then the only (sad) thing left to do (short of switching the computer off and on again), is to press the RESET button on the left of the keyboard. BE WARNED that in so doing any program you might have painstakingly typed in for the past three hours will be lost unless and until you learn how to retrieve it as explained in the Handbook.

(Switching the computer off, however, completely wipes your program out of the DAI's memory).

While we're on the subject of lost programs and time, we would like to advise you that the way to avoid this is to SAVE on tape a program you're developing (or copying from a book or magazine) at regular intervals - say every ten or twenty minutes - even while you're typing it into the computer's memory and it's still incomplete.

You will start at the beginning of the tape and SAVE what's in memory at that stage then rewind the tape ten or twenty minutes later (or whatever time you seem appropriate depending on how fast you are adding new lines to the program) and so on. This way you will not waste an inconsiderate amount of tape and the latest

(most complete) version of the program will always be available at the beginning of the tape, where you can easily find it.

Doing so will protect you against power failures (like a pet or a bored husband/wife accidentally tripping over the power cable...) or some bug (if you think that's an unusual way of describing an error in a program, you'll soon change your mind!) or other causing your computer to go temporarily insane.

If either of those nasty things should happen, you'll be grateful you won't have to start typing from scratch, but simply LOAD the latest version from tape and continue from there.

For the moment though, we won't go into the details of how to SAVE and LOAD programs, since there is no program in memory and you're still not even supposed to know what a program is (you're a novice, remember?).

6. REPT key. REPeats the symbol or function of another key when and for as long as it is held down in conjunction with that key.

Try it out. Press every key in turn, while pressing the REPT key and fill the screen with pretty patterns of letters and other symbols. You can then clear the screen and go on to the next chapter.

WHERE YOU START BOSSING YOU COMPUTER AROUND

Now that you have a rough idea of how to use its keyboard, you can begin to use your computer.

You bought a computer capable of generating colour graphics and coloured text and background on your domestic TV set. Yet right now you're staring at a dull display of dark grey text on a light grey background (which is all you'll ever get unless you use a colour TV, but that's none of our business!).

Here's how you can put colour on the screen. Type the following BASIC command (please note that all commands to the computer must be issued in CAPITAL LETTERS):

➤ COLORT 5 Ø Ø Ø ⏎

Of course, nothing will happen until you press RETURN.

When you do, the text will be black and the background will turn green. Should you instead get a SYNTAX ERROR,

check you have entered the command exactly as shown, including the right spaces between the numbers.

You might prefer another text/background colour combination, though. So why don't you select your favourite one by trying out the various possiblities. To do so, remember that the first number following the COLORT command (5 in the example) will select one of the 16 available colours for the background, while the second figure (∅ in the example), which MUST be separated from the first one with a space, will determine the colour of the text.
(Notice that computers consider ∅ as a regular number, thus the 16 available colours on the DAI are numbered from ∅ to 15.)
The following two numbers (∅ ∅) must also be present, though they perform no apparent function. (The reason for this if you really want to know is that the COLORT command must have the same syntax as the COLORG command, explained later, or more simply, that's the way it works.)

It IS simple to change the colour of the text and the background, isn't it?

Now we feel that since you bought a computer you want
to see it do even the simplest job for you, especially if
it is a tedious one like having to type

COLORT something something $\emptyset$ $\emptyset$

256 times in order to see all the possible combinations of
text/background colour.

Wouldn't it be nice to have the computer do the job?

Well, it can and it's going to be your first programming
exercise. So try to resist the urge to skip to the page
where the final program is listed, but rather follow the
various steps as they are presented in order to get   to
grips with the numerous features of the BASIC language
and of the DAI Personal Computer that   even a short
program such as this one will allow us to introduce.

WHERE YOU FINALLY BEGIN PROGRAMMING YOUR COMPUTER

The first thing you should know about writing a program in BASIC, is that after you type RUN, and press the RETURN key, the computer executes a series of commands which you (or another programmer) have stored in its memory. The order of execution of each command is determined by the number preceding the command itself. This is known as the LINE NUMBER. Every time you issue a command to your PC without preceding it with a LINE NUMBER, it will be executed immediately after you press RETURN and no trace of it will remain in memory. If you precede the command with a LINE NUMBER (any whole number between 1 and 65535) then it is stored in memory on pressing RETURN and executed only when you RUN the PROGRAM.

So that you don't have to take our word for it, type:

➤ NEW ⏎

(and press RETURN), to tell the computer you want to start writing a new program, and then type the following line:

➤ 100 COLORT 10 5 0 0 ⏎

(remember to press RETURN). Nothing happened, right? Wrong, something DID happen: YOU'VE JUST ENTERED YOUR FIRST PROGRAM IN THE DAI ! (But we do agree that nothing seemed to happen)

Now type:

➡ LIST ⤶

and the computer will list the program currently in memory. In this case, the DAI will print:

1OO COLORT 1O 5 O O

which is the line you typed in. It is good practice to LIST a program after typing it in, to check it is correct before attempting to RUN it.

Now type:

➡ RUN ⤶

and see what happens. Is it what you expected?
The text went green and the background orange: your DAI executed the COLORT 1Ọ 5 Ọ O command in line 1OQ.
Encouraged by this success, let's continue writing the program that will eventually display in turn all possible text/background colour combinations.

Notice we picked 1QQ and not 1 as the first LINE NUMBER for our program. WHY? Because one of the nice facts about BASIC is that no matter in what order you type in a new line, the computer will insert it in the right place according to its LINE NUMBER.
So, if after typing our first line 1QQ COLORT 1Ọ 5 Ọ Ọ we should decide that the DAI must carry out another

instruction BEFORE changing the background and text colours, we would simply have to give that instruction a line number SMALLER than 100. If our first instruction had been:

1 COLORT 10 5 0 0

then since LINE NUMBER 0 is ILLEGAL (computerese for NOT ALLOWED), we would have had to retype both the old and the new lines.

It is therefore good programming practice to start numbering your lines with a reasonably high line number - say 100 - and increment that number by 10 for each new line number. This will leave plenty of room for adding lines in between when needed.

So far the computer hasn't really saved us any effort. On the contrary in order to get the green text and orange background, we had to type in more things than were necessary when we changed the colours on page 15. True, but bear with us.

VARIABLES

The next thing you'll have to learn, so we can continue writing our program is the use of VARIABLES.

It may not be very original, but we're going to ask you to imagine variables as being names for pigeonholes which

can contain a value you need to store for future use in your program. In order to create a pigeonhole, you simply have to think up a name for it and assign a value to it. For example, LET A = 1 would store the value 1 in the pigeonhole A. (Read "LET A = 1" as "let A be equal to one", which makes it clearer that it is YOU who have decided that the pigeonhole YOU named A must be assigned the value 1).

The name can be as short as one letter or as long as will fit in one line of the program. However, the DAI will only recognize the first 14 characters (much more than other versions of BASIC which only recognize the first two characters), so that in practice THISISAVERYLONGVARIABLE and THISISAVERYLONELYONE both refer to one and the same pigeonhole as far as the computer is concerned (because the first 14 characters are identical). But we hope you will hardly ever need to use such long variable names!

The reason one should preferably give whole names to variables as opposed to single letters of the alphabet is that it makes it easier to remember what use you want to make of the value contained in that variable. For example, in our program we're going to need variables to store the values for the background and text colour numbers to be used in the COLORT command. Although we

could decide to call these two variables B (short for background) and T (for text), we may as well call them BACKGROUND and TEXT which makes it clear for everyone what they're being used for. Type:

➡ LET TEXT = 5 ..

and don't forget to press RETURN. This causes two things to happen:

1.   Somewhere in the DAI's memory, a pigeonhole is labelled "TEXT", and

2.   the value 5 is stored in that pigeonhole.

As usual, we don't expect you to take our word for it. So how can you check that what we told you is true? Here's how you can ask your DAI to confirm that this is so:

        The PRINT command.

How do you normally ask something? With a question, of course! And that's exactly what you're going to do just now except in this case the question mark comes first. Type in:

➡? TEXT::

the question mark is equivalent to (but shorter than) typing the word PRINT and that's the way you ask the computer to print (on the screen or on a printer) the value of something stored in its memory. If you

remembered to press the RETURN key, the last line on the

screen should be:

5.0

which is the value stored in the pigeonhole   named

TEXT when you typed LET TEXT = 5

Just to see that you could equally have used the word

PRINT instead of the ? mark, type in:

➡ PRINT TEXT ↵

(need we still remind you to press RETURN?).

The reason why you got 5.0 as an answer to both ?   TEXT

and PRINT TEXT rather than a simple 5 is that unless

otherwise specified (by you) all variables are considered

to hold real (or floating point) numbers, as opposed to

integer (or whole) numbers. If you want a variable   to

contain only integer values then you must append a % sign

to the end of its name, for example, type:

➡ LET TEXT% = 5 ↵

Now

➡ ? TEXT% ↵

will give you:

5

The advantage of using integer instead of floating point

variables is that programs will RUN slightly faster,

because calculations with integers are easier to execute.

To spare you the effort of typing a % sign after every

integer variable name, DA1 BASIC allows you to imply
before you start writing a program, that a set of
variable names refer to integer variables. Say for
example you wanted names beginning with letters A-I to be
integer variables. You would then type:

➤ IMP   INT A-I ⏎

where   IMP is the IMPly instruction and INT is short for
integer.

There's a lot more you'll have to learn about variables
and variable types, but for the moment this will be amply
sufficient to allow you to understand what follows.

Now type:

➤ NEW⏎

(and press RETURN) to start afresh. Then type in the
following lines:

➤ 80 LET BACKGROUND = 0⏎

➤ 90 LET TEXT = 1⏎

➤ 100 COLORT BACKGROUND TEXT 0 0⏎

First of all, you should check that what you entered
in the computer corresponds to the lines printed in this
manual. To do that, type:

➤ LIST⏎

As soon as you press RETURN, the screen will be cleared
and the program will be LISTed on the screen.

Now try to figure out what will happen when this program
is executed. Then to see it happen type:

➤ RUN⏎

and don't forget to press the RETURN key.

This part of the program was meant to demonstrate that
using a VARIABLES instead of (CONSTANT) numbers produces
exactly the same effect.

(By the way, did you get blue text on a black
background?)

From now on, to change the colour of the text and/or the

background you simply have to change the numbers stored in the variables in lines 8Ø and/or 9Ø (without having to retype line 1ØØ) and RUN the program to obtain the desired colour change.

How do I go about changing the values in lines 8Ø and/or 9Ø, you might ask. For the moment, you'll have to do it "the hard way", that is, by retyping the whole line. We want you to try this, because it will introduce you to the idea that whenever you type in a new line having the SAME LINE NUMBER as an existing one, the old one is DELETED and the new one is substituted to it. (For the same reason, if you ever need to DELETE an entire line from your program, all you need to do is enter the number of the line you wish to delete and then press RETURN: since the new line thus created would be blank and the computer doesn't store blank lines, that line number will no longer be present in the program).

Take a few minutes now to practice changing the colours as explained, then come back for more.

The program we are writing together is supposed to show you all the possible colour combinations in sequence, so first of all we have to assign a starting colour to the variables BACKGROUND and TEXT (lines 8Ø and 9Ø of the program in memory do just that), then change

the screen to those colours (line 100 does that).

The next thing to do is to have the computer change the number contained in the variables TEXT and BACKGROUND instead of having you do it manually as we asked you to do a few lines ago. Since our aim is to have the computer show us all colours, we can start by having it add 1 to the variable BACKGROUND so that it contains the number corresponding to the next colour.

When you want the computer to add a number to an existing VARIABLE you tell it to LET the variable be equal to the (present value of the) variable itself plus the number you wish to add to it, or in other words (if the variable is BACKGROUND and the number you want to add is 1) you would say: LET BACKGROUND = BACKGROUND + 1. Just before going on, you should try this a few times before using it in the program to see that it works as you expect it to. You can practise in the following way:

- first you think of a variable name, say MOTHER or JOHN or TIMBUCTU or WHATHAVEYOU (by now you should know that you can give a variable any name you want up to 14 characters long);

- then store a value in your variable (to do that you type, for example:

� LET MOTHER = 30↵

- then check that MOTHER (in the example) contains the value 3Ø by typing:

➤ ?   MOTHER ⏎

to which the DAI will respond with:

3Ø.Ø

- then you can for example type:

➤ LET MOTHER = MOTHER + 4 ⏎

nothing seems to happen when you press RETURN but you can check that the DAI has indeed added 4 to the value already present in MOTHER by typing:

➤ ?   MOTHER ⏎

which this time will print:

34.Ø

just as expected. Try going through the various steps a few times, using different variable names of your choice, assigning various initial values to them and then adding numbers and checking that the results correspond to what you expect. This exercise should make you more familiar with what is possibly one of the hardest concepts to grasp when learning BASIC.

OK.   Now let's apply this new bit of knowledge about the way to add a value to a variable in the program we're writing.

The use of the word LET is optional, and is very   often

omitted in programs in order to save that extra byte of
memory space it would otherwise occupy. However, we
think that in the beginning of your programming career it
would be better for you to use the word LET in your
programs, to remind you of what exactly is happening. So
add the following line to the program in memory;

➤110 LET BACKGROUND = BACKGROUND + 1↵

When line 110 is executed, the value of the
expression on the right of the = sign will be calculated
and stored in the variable BACKGROUND. In the
calculation on the right of the = sign the value of
BACKGROUND is obviously the value currently stored in
that pigeonhole, which is set to Ø in line 8Ø. This
value will be added to 1 and the result stored in the
same variable BACKGROUND. From then on the value of
BACKGROUND will be 1 unless changed by some other line in
the program.

Before asking you to RUN the program in its present form
to see what happens, we would like you to check exactly
what is in memory now.

Type:

➤LIST ↵

and the computer will display the program currently in
memory. It should be something like this:

80 LET BACKGROUND = 0

90 LET TEXT = 1

100 COLORT BACKGROUND TEXT 0 0

110 LET BACKGROUND = BACKGROUND + 1

If you have been experimenting with different values for BACKGROUND and TEXT, as we suggested, lines 80 and 90 will probably need to be changed to match those in the above listing. You could retype both lines, as explained earlier, but that would be doing it the hard way again. Instead, we're going to show you how easily you can change one or more characters in a program by using

THE EDIT FACILITY

Type in:

➡ EDIT

If you remembered to press RETURN you should now see your program LISTed on the screen, with a few differences:
- on the left-hand side ther's a solid vertical stripe starting just below the last line of the program;
- at the end of each line you can see the symbol ⮐, which indicates the place where you pressed the RETURN key while originally writing the program;
- this time, the CURSOR which is flashing in the top left-hand corner on the first digit of the first line

number can be moved around the screen using the four
CURSOR control keys mentioned before, choosing the
appropriate arrow to move up, down, left, right;
- any characters you type will be INSERTED to the left of
the flashing character (which indicates the cursor
position);
- the CHAR DEL key will erase the character flashing at
the cursor position and close up the gap, moving all
characters right of the cursor one place to the left
every time the key is pressed.

All this may be very confusing to read but in
practice you will find the EDIT mode extremely powerful
(computerese again...meaning versatile, useful) and yet
so easy to use that it will soon seem indispensable.
Practice changing the values of BACKGROUND and TEXT a few
times as follows:
- place the cursor (to move it over long distances you
press both the desired arrow key and the REPT key to
speed the cursor to the right position) to the right of
the = sign after BACKGROUND so that it flashes on the
first digit of the value for BACKGROUND.
- then press CHAR DEL as many times as necessary to
delete the present value.
- and finally just type in the new value.

Once you have changed the program as desired, you must leave EDIT mode. There are two ways of exiting EDIT:
- if you're sure you made the right changes and you don't want to change your mind, then press the BREAK key ONCE followed by the SPACE BAR. With long programs, the prompt (*) will reappear after a few seconds at most during which time we advise you not to press any keys.
In the case of our short program the asterisk will reappear almost immediately after you press the space bar;
- if on the other hand you wish to restore the original program, ignoring any changes you made while in EDIT mode, all you have to do is to press the BREAK key TWICE.

Now try the whole procedure a few times changing the values of TEXT and/or BACKGROUND while in EDIT, exiting EDIT pressing BREAK and the space bar and RUNning the program to see the colours change. Mind that if you set text and background to the same colour, you'll be in trouble, since the text will seem to disappear. It is there, but have you tried using white chalk to write a letter on white paper?
When you've finished practising with the EDIT mode make sure, by LISTing it, that the program in memory corresponds to this one:

80 LET BACKGROUND = 0

90 LET TEXT = 8

100 COLORT BACKGROUND TEXT 0 0

110 LET BACKGROUND = BACKGROUND + 1

in order to continue writing our program. If it doesn't,
by now you should know how to change it to make it
correspond.

ON LOOPING

The last line we added (110) causes the variable
BACKGROUND to be incremented to the next colour number.
However, it's no use doing that if it isn't followed by a
COLORT command to effectively change the colour of the
background on the screen. So type:

→ 120 COLORT BACKGROUND TEXT 0 0

and RUN the program. See? This time you got grey text
on a blue background because line 110 added 1 to the
value of BACKGROUND (which was set to 0 in line 80) and 1
is the colour number for blue (see Appendix A). Now to
get grey text on a petunia red background you can add
these two lines:

→ 130 LET BACKGROUND = BACKGROUND + 1

→ 140 COLORT BACKGROUND TEXT 0 0

and RUN the program.

We could ask you to continue adding a line 15Ø identical to lines 11Ø and 13Ø and a line 16Ø identical to lines 12Ø and 14Ø and so on, in order to display all sixteen possible colours for the background. However, this would be a very repetitive and tiresome task, harder than changing the colours as you did earlier when you did it manually, as opposed to programming the computer to do it automatically.

Instead, we are going to introduce you to one of the most common programming techniques - the use of LOOPS to perform repetitive tasks.

One way of programming a LOOP in BASIC is to use the GOTO instruction, which causes the computer to continue executing the program from the line whose number follows the GOTO. Type:

➤ 12Ø GOTO 1ØØ ↵

Before asking you to RUN the program to see it work, let's take each line in turn and examine their function. Lines 8Ø and 9Ø respectively assign an initial value to the variables BACKGROUND and TEXT so that the first background/text colour combination to be displayed is grey text on a black background.

Line 100 actually performs the colour change using the values stored in BACKGROUND and TEXT.

Line 110 adds 1 to the value currently stored in BACKGROUND. The first time round it will add 1 to 0 and store the result (1) in BACKGROUND, the second time round it will add 1 to the current value of BACKGROUND which is 1 and store the result (2) in BACKGROUND, and so on.

Line 120 sends the program to continue at line 100 where the colour of background and text is changed according to the value contained in the variables BACKGROUND and TEXT. Since there is no instruction to change the value of TEXT the text colour will never change. Line 110 however adds 1 to the value of BACKGROUND every time it is executed and therefore the background colour does change through the whole range of available colours.

Now try to imagine what is going to happen when you RUN the program and then do it to see what happens.

Type:

➤ RUN ⌐⌐

and press RETURN

Is that what you expected? The background colour changed so rapidly through all sixteen colours that you did not even have time to see each colour and the end result is grey text on a white background spelling out the message:

NUMBER OUT OF RANGE IN LINE 100

So what happened?

What happened is that unless you slow down the computer, some of its actions are too fast for the human eye to perceive. Luckily, there is a ready-made instruction in the DAI version of BASIC that can be used to insert a pause in a program. So add the following:

➤1Ø5 WAIT TIME 1ØØ

(isn't it just as if you were talking English to your computer?

The value 1ØØ is the number of 20 millisecond intervals for which you want the DAI to pause. In this case, 1ØØ times 20 equals 2000 ms, i.e. two seconds.

As for the error message, it was caused by the fact that the computer attempted to execute line 100 with a value of 16 for the variable BACKGROUND. The range of colours is numbered, however, from Ø-15 and since there is no colour in the computer's memory corresponding to 16, your DAI says that there is a number out of range, i.e. higher than 15, and that it realised that while attempting to execute line 1ØØ.

How can you avoid getting that error message? Read on....

The IF statement.

One thing that makes computers look smart in the
eyes of people who have never tried to program them, is
their ability to take decisions based on the occurrence
of certain predetermined conditions.
One way to make use of this powerful feature is to use
the BASIC statement IF...THEN.
Let's use it in our program and see how it works.   Add
this line:
→12Ø IF BACKGROUND < 16   THEN GOTO   1ØØ
as you can see, instead of having line 120   send the
computer back to line 1ØØ UNCONDITIONALLY with a GOTO
1ØØ, this time it will only jump back to 1ØØ IF the
variable BACKGROUND is < (less than) 16 thus avoiding
incurring in the error that caused the message NUMBER OUT
OF RANGE IN LINE 1ØØ to be displayed last time the
program was RUN.
When BACKGROUND becomes 16 the condition is no longer
satisfied and the computer does not execute the
instruction following the THEN (in our case it does not
GOTO 1ØØ). Instead, it carries on executing the program
with the following line (in our case there is no
following line and the program ends.)

Now LIST the program once more before RUNning it:

80 LET BACKGROUND = 0

90 LET TEXT = 8

100 COLORT BACKGROUND TEXT 0 0

105 WAIT TIME 100

110 LET BACKGROUND = BACKGROUND + 1

120 IF BACKGROUND < 16 THEN GOTO 100

and check it is correct. Then type:

➤ RUN

Nice, isn't it? But why not ask the computer to tell you what the current BACKGROUND and TEXT colours are so that you can make a note of it?

This line will do the job:

➤ 101 PRINT "BACKGROUND = ";BACKGROUND,"TEXT = ";TEXT ⏎

As you will see when you RUN the program the computer will print BACKGROUND = because it is contained   in quotation marks, then next to it (because of the ; (semicolon) after the quotation marks) it will print the value of the variable BACKGROUND. A comma instead of a semicolon is used after the variable name BACKGROUND and this causes what follows to be printed at the beginning of the next field on the screen. (Every line on the screen is divided into five fields each 12 characters long.) That's why TEXT = (also contained in quotation

marks) is not printed right next to the value of
BACKGROUND but rather starting 24 spaces away from the
left edge of the line, i.e. at the beginning of the next
field.   The semicolon that follows the quotation mark
after TEXT = causes the value of TEXT to be printed right
next to the : sign and not at the beginning of the next
field. If this sounds incredibly confusing just try it
changing the semicolons to commas and viceversa to see
what happens. As usual it is much easier to understand
something you see happening on the screen than taking our
word for something explained in these pages.

As you can see, trying to write a program and teach even the most elementary notions of BASIC at the same time, takes quite a while and the program we set out to write still isn't complete.

So far, the program caused the screen background to go through the entire range of 16 colours while the text colour never changed. In order to finish our program and see all text/background combinations, we must change the text colour every time we've gone through the entire range of background colours.

Three more lines will do the trick so type them in:

➤130 LET TEXT = TEXT + 1 ⏎

➤140 IF TEXT ‹ 16 THEN GOTO 90⏎

➤150 COLORT 15 Ø Ø Ø : REM BLACK TEXT ON WHITE BACKGROUND .

Line 130 will only be executed when the condition for the computer to jump back to line 100 (in line 120) is not satisfied, that is when BACKGROUND is 16, indicating that we have been shown colours Ø – 15 for background with that particular colour for text. It is then time to change the text colour and line 130 does exactly that.

Line 140 checks that the variable TEXT never becomes 16 (which would cause a

NUMBER OUT OF RANGE IN LINE 1ØØ

to occur) and THEN sends the program to line 9Ø where
BACKGROUND is reset to Ø so as to go once more through
the range Ø - 15.
When TEXT is 16, the program would normally END leaving
you staring at a totally white screen since the last
COLORT in line 1ØØ was executed with BACKGROUND = 15 AND
TEXT = 15 (white text on a white background!)
Line 15Ø takes care of that by setting black text on a
white background, as explained in the line itself after
the BASIC statement REM.

REM is short for REMARK. Anything following a REM
in a program line is there for the sole use of humans
(i.e. you). Computers ignore REMs when executing
programs, but print them out in program listings; which
is useful to remind you why you used that particular
instruction when you need to revise the program weeks or
months after you originally wrote it.
Also notice in line 15Ø that a : (colon) separates TWO
instructions in the same line. It is indeed possible to
do that, but we would advise you, for the sake of easy
program interpretation both by you and others, not to put
more than one instruction per line (except of course for
REMs which actually help make programs easier to
understand).

Multiple instructions in one line are only useful if you have to conserve memory and/or you want your program to RUN slightly faster, but the trade off can be very aggravating when the time comes for you or anyone else to DEBUG a messy program.

So we finally have a working program which does everything we set out to accomplish. Let's LIST it just one more time in its complete form:

```
80 LET TEXT = 0
90 LET BACKGROUND = 0
100 COLORT BACKGROUND TEXT 0 0
101 PRINT "BACKGROUND = ";BACKGROUND,"TEXT = ";TEXT
105 WAIT TIME 100
110 LET BACKGROUND = BACKGROUND + 1
120 IF BACKGROUND < 16 THEN GOTO 100
130 LET TEXT = TEXT + 1
140 IF TEXT < 16 THEN GOTO 90
150 COLORT 15 0 0 0 :   REM BLACK TEXT ON WHITE BACKGROUND
999 END
```

Now RUN the program one or more times just to see it working and decide on the best text/background colour combination in your opinion.

If you feel you fully understand the way this program works, then you are well on the way to learning how to program your own applications on your brand new personal computer.

ON SAVING AND LOADING PROGRAMS

Before going on to other interesting things, it is time you learnt how to SAVE your programs on tape for later use.

Nothing could be simpler, but let's do it together step by step, just the same:

1 - you must naturally have a program in memory, and if you've followed this manual so far you should have one right now;

2 - put the cassette you intend to save the program on in the recorder and be ready to start recording when prompted by the computer;

3 - think up a name for the program you're about to save. You don't have to, but if you do you'll then be able to ask the DAI to LOAD only the program with that name from a tape containing several (differently named) programs.

Let's call this program "FIRST";

4 - type:

➤SAVE "FIRST" ↲

when you press RETURN, the computer will respond with:

SET RECORD,START TAPE, TYPE SPACE

so why don't you do what it says...

5 - when the prompt (*) comes back, you'll know the computer has finished recording the program on the cassette.

The next thing you should do at this point is to repeat steps 4 and 5, thus SAVEing the program once more after the first recording. This is called redundant recording and it is important if you want to have an extra chance of retrieving your program at a later time. Finally, if you really want to play it safe you may CHECK that the program was recorded properly. To do this, you first rewind the tape to the start of the first recording, then type:

➤CHECK ↲

(and press RETURN)

and start the tape in PLAY mode.

If everything is OK, a few seconds later the computer should print:

FIRST OK

otherwise, the message would be

FIRST BAD

If you do get the BAD message, then here are a few of the things that might have caused it:

- the head of the recorder needs to be cleaned;

- the quality of the cassette tape you used was not good enough;

- the volume setting during recording or playback was incorrect.

The cure for the first two problems is obvious. In the third case, you must repeat steps 4 and 5 with different volume settings (keeping the TONE control on maximum treble) and CHECK the recordings until you get an OK. When you do, make a note of the volume setting and you shouldn't have any more problems subsequently.

When you're sure you have SAVEd FIRST on tape, you can turn the machine off and on again (which assures you there is no trace left of the program in memory) and LOAD the program into computer memory just to see that it was indeed SAVEd on tape.

Here are the steps:

1 - rewind the tape to the beginning of the recording;

2 - type:

➤ LOAD ....

3 - press RETURN and start playing the tape.

When the prompt reappears, you can LIST the program
to see that it is really back.

That's all...

Now that you have a few fundamental notions about your computer and the language it understands, it is time for you to get acquainted with one of the features which most distinguishes this machine from the others: COLOUR GRAPHICS.

A RESOLUTE APPROACH

A graphic picture on a television screen, just like a photograph on paper, is made up of a number of dots. The finer the dots and therefore the higher their number in a given area of the picture, the better the quality of the image thus produced. In technical terms one says that a photographic film has a high RESOLUTION, when the number of distinct dots that make up an area of the picture is so high and each dot is so very small that the picture doesn't look "dotty" at all.

The DAI lets you draw pictures on your TV screen with a choice of three levels of resolution, which are:

72 dots across by 65 down

160 by 130, and

336 by 256.

At each of the three levels, you can choose to use

any four of the 16 available colours or all 16. You can
also decide whether to use the whole screen for the
graphic pictures or to leave room at the bottom of the
screen to display up to four lines of text. The total
number of options (or MODE's as we call them) at your
disposal is thirteen if you count MODE Ø which is the
all-text mode. We thought we'd include the table below for
your convenience:

Mode	Graphics size	Text size	Colours
0	-	24X60	any 2 of 16
1	72 X 65	-	16
1A	72 X 65	4 X 60	16
2	72 X 65	-	any 4 of 16
2A	72 X 65	4 X 60	any 4 of 16
3	160 X 130	-	16
3A	160 X 130	4 X 60	16
4	160 X 130	-	any 4 of 16
4A	160 X 130	4 X 60	any 4 of 16
5	336 X 256	-	16
5A	336 X 256	4 X 60	16
6	336 X 256	-	any 4 of 16
6A	336 X 256	4 X 60	any 4 of 16

If your machine has a full 48k (one K=1024 bytes) of RAM memory then all MODE's are available to you, if not you can easily find out which MODE's are not for you (until you decide to buy more RAM) by trying to select each MODE in turn. The way to select a graphic MODE could not be simpler: for example, to select MODE 1, type:

MODE 1

and the DAI will prepare the screen to display coloured dots rather than alphanumeric characters. Notice that the bottom part of the screen is still used to display up to four lines of text. According to the table above this should not be so, since it is MODE 1A and not MODE 1 that allows up to four lines of text at the bottom. However, it wouldn't do to have you stare at a totally graphic screen while in control of the machine. How could you possibly check that you where typing the right commands if the screen didn't show you the text as you typed it in? Therefore, the choice of selecting a MODE without text at the bottom only applies when a program is running (and you are no longer typing commands in the computer). Try for example:

10 MODE 1 ↵

20 GOTO 20 ↵

RUN

See? Now the entire screen is used to display graphics rather than alphanumerics. This program will never end of its own accord since line 20 causes it to loop continuously. Stop it by pressing the BREAK key and the graphic area will be pushed up to make room for up to four lines of text at the bottom. If the program you just stopped had been one that allows you to draw pictures on the screen, and had you drawn a beautiful landscape or what-have-you, you would now think you lost the top part of your masterpiece. Not so. When the DAI makes room for the text at the bottom, the top part of the graphics simply slides into a part of memory not displayed on your TV screen, but it can easily by pushed back down for further viewing with a simple trick (explained later).

Now try to see if you can get all MODE's by typing the word MODE followed in turn by numbers 2 to 6. If your computer does not have enough memory for one of the higher resolution MODE's it will tell you.

In any case, even if you can't get the higher resolution MODE's, you can still learn how to use them. In fact, all the examples which follow will be based on the lowest resolution MODE, available on all machines, but apply equally to all MODE's.

TWO BITS OR NOT TWO BITS

But what's all this business about four colours and 16 colours?

Before you think you've lost twelve colours on the way home from the shop, let us explain what's going on.

You're not expected to study or fully understand what follows right now. It is not essential in order to begin to use the graphics, but it's important you know why there are some restrictions in the use of the colour graphics. A more technical description of the graphic system is given in section 3.0 in the second part of the manual.

Basically the restrictions are due to the fact that the system we adopted allows you to work with sixteen colours in high resolution with HALF THE AMOUNT OF MEMORY which would be required for a totally unrestricted use of the colours.

We feel we adopted the only practical approach to give a home computer the spectacular graphic possibilities your DAI has.

CANVASSING

In order to display dots of colour on the screen, a special electronic circuit inside the DAI to which we will refer as "the video circuitry" continuously scans an area of the computer's RAM memory to be told what colour each dot on the screen must be. This part of RAM (known as

"screen refresh memory") stores an "image" of the dots on the screen, as a pattern may be printed on a canvas for young artists to paint upon. Many times per second the video circuitry in the DAI paints the picture on the screen using the pattern in memory as a model.

To start with the simple case, let's suppose we only wanted to produce two-colour pictures. You know perhaps that the smallest unit of memory in computers is the bit, which is a digit that can only have the value Ø or 1. The video circuitry will see the refresh memory as a pattern grid where every bit corresponds to the position of a dot on the screen. By convention the video circuitry will know to display a dot of one colour (say the background colour) if the bit in memory corresponding to that dot on the screen is a Ø. Every time it encounters a bit of value 1, our electronic painter will display the corresponding dot in the other available colour, which we could call the "foreground colour". It doesn't matter what the two chosen colours are, but you'll always be limited to them. What we just explained is in fact the way most black and white graphic systems work.

As you can see the number of bits of memory corresponds exactly to the number of dots on the screen. In this case, to display 86,016 dots (336 x 256), you need 86,016 bits of RAM memory. To express this number in terms of BYTES (the unit of measurement for memory in microcomputers = 8

bits), 10,752 bytes would be sufficient to hold the information required by the video circuitry to paint the picture even in the highest resolution MODE. This would not be an excessive amount of memory, if you consider that the DAI comes with at least 12,288 bytes (12K).

But we want the dots to be multicoloured.
On those painting patterns for children we referred to before, the artist is told what colour to use in any of the tiny squares (or dots, depending on how fine is the detail or resolution of the painting) that make up the picture by a number printed within the square itself. Exactly the same thing happens in the DAI. However, it is impossible to represent numbers greater than one with a single bit.
Two bits are necessary to represent numbers $\emptyset$ to 3 (allowing a choice of four different colours) and four bits are required to represent numbers $\emptyset$ to 15 (which allows any of 16 colours to be specified.)

So in order to specify which of 16 colours each dot on the screen should be, one would need to use four bits per dot. In the highest resolution this would require the use of a massive 344,064 bits of memory to store the information for the 86,016 dots (336 x 256). In other words, 43,008 bytes of RAM would have to be reserved for the graphic picture. Even in a DAI provided with the

maximum amount of 48k (49,152 bytes) of RAM there would be very little space left over for the programs you'll want to write to make use of the graphic display capability.

In order to reduce the amount of memory required for screen refresh, we had to reduce either the number of available colours or the resolution. We decided to keep the high resolution, so we somehow had to cut down the choice of colours. We came up with a system that on the one hand reduces the memory requirements by half and on the other will still allow you to work with sixteen colours albeit with some minor restrictions. Actually we came up with two different memory saving solutions, and we thought we'd let you decide on a case by case basis which of the two best suits the apllication on hand. The two ways of using the graphics on the DAI are:

- the 16-colour mode

and

- the 4-colour mode.

We shall discuss the latter first, bearing in mind that most of the commands we shall introduce apply equally in the former.

In this case memory consumption is limited by reducing the number of colours that can be shown AT ANY ONE TIME on the screen.

Two bits of memory are associated to each dot (requiring 21,504 bytes in the highest resolution)and tell the video circuitry which of four colours that dot must be. That does not mean, however, that your drawings in four-colour MODE will always be limited to the same four colours. YOU can determine what four colours to use at any one time, by choosing them out of the 16 available colours and placing their numbers in four special memory locations we call COLOUR REGISTERS. So at any one time there cannot be more than four different colours showing on the screen, but by loading a new colour in one of the registers all the dots whose two bits select that register (as opposed to selecting a fixed and predetermined colour) will immediately turn to that new colour. That means that in turn the same picture can be displayed in any of the sixteen colours just by changing the contents of one (or more) of the four colour registers. This can in fact be used for very interesting effects, including one we call animated drawing facility, whereby you can have smooth movement of graphic objects by not showing the object in the new position until it is fully drawn (see section 6.2.12.5 in the second part of the manual for details.

Since, admittedly, all the above is very confusing and harder to understand when presented in theory, let's experiment with the graphics.

All the examples will be based on the lowest resolution mode, available on all machines. If you have enough memory you can try out the same examples in the higher resolution modes, by simply selecting them with the appropriate number after the MODE command.

Type:

→ MODE 2 ⊐

and the screen will turn black except for the bottom where you can still see up to four lines of text and where the background colour will be independent of that of the dots on the screen are now set to the colour whose number is contained in the first of the four colour registers. Because you haven't changed it since you switched the computer on (or rather, we did not ask you to), that first register, just like the other three, contains the number Ø, which is why the screen turned black and not blue or orange (see Appendix A for list of colour numbers). The moment you change the number contained in the first register the whole screen will instantaneously change to the colour selected by the new number. To get a blue screen, type:

➤ COLORG 1 Ø Ø Ø

see?

THE DOT COMMAND

Now type:

➤ DOT 5,5 14. ⤶

and the computer will answer:

COLOR NOT AVAILABLE

Why? Because what you asked it to do was to place a yellow
dot on the screen, but yellow (colour number 14) is not
loaded in one of the four colour registers and therefore
not available for use at the moment. Instead you can try :
DOT 5,5 Ø

and you'll get a black dot on the screen (yes, it does
look more like a square than a dot, but don't forget
you're using the lowest resolution available on the DAI).
The position of the dot is determined by the two numbers
separated by the comma (5,5). If you remember your
cartesian geometry you'll know that 5,5 are respectively
the X and Y coordinates of the position of the dot, with
the origin Ø,Ø being in the bottom left-hand corner of the
graphics area.

Otherwise you can think of the screen as being divided
into a number of vertical columns and horizontal rows of
dots. DOT 5,5 tells the computer to place a dot in the
fifth column from the left edge of the screen, five rows

from the bottom of the graphic area. The colour of the dot is specified by the number separated by a space after the row number. To be used, the colour number MUST have previously been loaded into one of the four registers.

THE COLORG COMMAND

So how do you load the registers? Easy. To load the registers with yellow (14), blue(1), green (5) and white (15) type:

➡️COLORG 14 1 5 15⤶

The screen turned yellow because now the first register contains the number 14 for yellow. You should also see a dot five columns from the left and five rows up from the bottom. It's the dot that was black while the screen was blue before you changed the registers with the last command you typed in. Now that dot is blue because you loaded blue (1) in the second register which was previously black (∅). To put a green dot in the left corner at the bottom of the graphics area type:

➡️DOT ∅,∅ 5⤶

and you can put a white dot above the blue one at 5,5 by typing:

➡️DOT 5,6 15⤶

Can you put a blue dot on the right of the white one? Try it. We are not giving you the answer this time, so you're on your own. But do it, if it has to take you one minute

or one hour to figure it out. The only way you'll ever learn to master your DAI is to try things out yourself: neither this scanty manual nor the thickest book in the world could make an expert of you if you do not experiment with the computer.

XMAX AND YMAX

How can you put a dot in the right hand corner on the bottom?
You could look up the table on page 51 to see what the maximum column number is for the MODE you are in. Since you are now in MODE 2 you would find it is 71 (yes, there are 72 dots but don't forget they are numbered Ø-71 and not 1-72). So put a blue dot in that corner by typing:
➤DOT´71,Ø 1↵

There is however a much simpler way than having to remember or look up the maximum values for columns and rows in each of the three levels of resolution. Instead of typing the actual number type XMAX for the maximum (rightmost) column or YMAX for the maximum (topmost) row. This is important because it not only saves you having to remember or look up six different values, but it also allows you to write programs that will work independent of the level of resolution you later choose. A few examples:
➤DOT XMAX,Ø 14↵

will erase the blue dot in the right hand corner because it covers it with a yellow one which is the same colour as the background and therefore invisible. To place a dot in the centre of the screen at any level of resolution you can type:

`· ·>DOT XMAX/2,YMAX/2 5 ↵`

The green dot that appeared on the screen as you pressed RETURN is not really in the centre of the screen, is it? That's due to the fact that, as we explained earlier, in order to allow you to see up to four lines of text in the bottom part of the screen, the graphic area slides up when you use the computer in direct mode, i.e. when you are typing commands to be executed directly and not during a program run. We also told you there is a trick to SLIDE THE PICTURE DOWN for full viewing. Here it is. Type:

➡️60000 MODE 2 ↵

➡️60010 GOTO 60010 ↵

➡️RUN 60000 ↵

(Notice you can ask the DAI to start executing a program starting from any line number).

You now have a totally graphic screen and that green dot is in the centre of it. The trick simply consists in RUNning a "dummy" program that re-selects the SAME MODE your picture was made in (line 60000 will therefore need to be changed for the other MODE's) and then endlessly loops (line 60010) just in order not to give you back control of the machine (and with it the space at the

bottom of the screen). The program could have any line numbers at all, but placing it as high as 60000 assures you it will not be in conflict with the real program that might be in memory (you're not very likely to use such high line numbers in your programs).

When you're tired of watching (presumably pretty soon, since right now you're staring at a few dots here and there on the screen), press BREAK and the graphics will slide up again to make room for the PROMPT and up to four lines of text.

Apart from DOT you can use two more commands to help you create graphic pictures on the screen.

THE DRAW COMMAND

For example, to get a blue horizontal line to cut across the screen from column 0 (left edge) to the last column (XMAX) ten rows from the bottom, instead of placing a series of dots to make up that line, you can type:

➤DRAW 0,9 XMAX,9 1⏎

or you can cut the screen diagonally by typing:

➤DRAW 0,0 XMAX,YMAX 5⏎

In other words, to draw a line, you type the word DRAW followed by a space, the position of the dot from which you want the line to be drawn (given the same way as in the DOT command), then another space followed by the

position of the dot where the line must end. Finally, after another space you type the number of the colour you want your line to have.

THE FILL COMMAND

If you need to fill a square or rectangular area of the screen (or the whole screen for that matter) with a certain colour, you can do that with the FILL command. For example, say you want to fill with green a square having one corner in 7,7 and the (diagonally) opposite one in 20,20. Type:

➤ FILL 7,7 20,20 5 ▭

and you'll get it.

ON YOUR OWN

Try out the various commands we introduced. Make up pictures with dots, lines, squares and rectangles. Try moving to a higher resolution if your machine allows it. Select only the even numbered MODE's for the moment, i.e. the four-colour MODE's.
If you get a
SYNTAX ERROR
at any time, it means you either mis-spelled the command, or you did not leave the right spaces between the various

numbers. Check with the examples above to make sure you're using the correct syntax. If you get an

OFF SCREEN

error message, it means you tried to place a dot or part of a line or fill outside the boundaries of the graphic area.

When you feel confident enough with the various commands come back to hear all about the 16-colour MODE.

THE 16-COLOUR MODE

In this mode you can display all 16 colours at the same time. The only limitation here is that on the same horizontal line of dots you cannot have 16 dots one beside the other in 16 different colours. Here's how the system works:

Each horizontal row is divided into a number of segments, each containing eight dots. Depending on the level of resolution, there will be 9, 20 or 42 such segments, or fields as we call them, on every horizontal row.

Within each field only TWO different colours can be used AT THE SAME TIME.

These eight-dot fields act as we explained earlier for two-colour graphics: each of the eigth bits of memory that correspond to the position of the dots in the field will be either a 0 or a 1, telling the video circuitry to display one or the other of the two colours allowed within that field.
WHAT colours though?
The answer is ANY TWO COLOURS chosen from the 16 available ones.

Instead of adopting a system of registers where you load
the numbers of the colours you want to work with as in
4-colour mode, in 16-colour mode each field has its own
two "registers" independent from those of any other field.
So two bytes are reserved in memory for each eight- dot
field. In one of them, as we said, each bit corresponds to
one of the dots on the screen and tells the video
circuitry whether to display the background (0) or
foreground (1) colour.

The second byte is split into two four-bit segments.
Remember? With four bits you can represent numbers $0-15$,
i.e. sixteen numbers; these two halves of a byte are in
fact the "colour registers" for the field. What happens is
this:

one four-bit half of the byte holds the number of the
background colour for that field, while the other half
will hold the number for the foreground colour.

This time you are not required as for the 4-colour modes
to choose the colours you want to use in any field
beforehand by loading their numbers in the registers. The
selection in made dynamically as you place dots, lines and
squares on the screen.

To start with, when you first select one of the 16-colour
modes, the background will be one solid colour (which
happens to be the colour contained in the first colour
register of the 4-colour mode). That means that in each
field one of the four-bit halves of the colour byte is

already set to that colour number.

Now you can place a first dot of any colour anywhere you like on the screen (try it). You can also put dots of different colours right above and below your first dot (again, try it).

What you cannot do is place a dot of a third colour in any field where apart from the background colour (first colour) a dot is displayed in the foreground colour (second colour).

Though admittedly restrictive, this system does go a long way towards giving you truly high resolution graphics in 16 colours. We feel confident that you will soon find ways to work around the necessary limitations of the system and create brilliant 16-colour graphic pictures.

To practise in this mode you can apply all the commands that are valid in the four-colour mode. The only effect COLORG will have in 16-colour mode is that the colour number you load in the first register will determine the colour of the background when selecting a 16-colour mode for the first time with a MODE command. Changing any of the registers including the first one while in 16-colour mode will have no effect on the picture on the screen.

If a dot fails to appear or part of a line is invisible or a chunk is missing from an area you ordered FILLed.

remember that is due to the "field" system and not to a program error or a computer malfunction.

Have a go now, by selecting for example the low resolution 16-colour mode:

➡ MODE 1 ⤶

then if your machine allows them practise with the remaining two modes.

At this point there are still a great many features of the DAI for you to discover.

We feel that if you have followed this first part of the manual right through trying out all the examples and practising on your own as we suggested, you should now be able to make sense and use of the more detailed and technical part that follows.

Take a big breath now and when you're ready for it turn the page and take the big plunge to discover the full power of your machine...