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.Ø

* _
```

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 ø'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 (Ø 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 Ø as a regular number, thus the 16 available colours on the DAI are numbered from Ø to 15.)

The following two numbers (Ø Ø) 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 Ø Ø

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:

100 COLORT 10 5 0 0

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 10 5 0 0 command in line 100.

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 100 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 100 COLORT 10 5 0 0 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.Ø 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, DAI 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-T to be integer variables. You would then type:

➔ IMP INT A-T ↵

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 Ø Ø ↵

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 30 by typing:

➔ ? MOTHER↵

to which the DAI will respond with:

30.0

- 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:

34.0

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 Ø Ø

and RUN the program. See? This time you got grey text on a blue background because line 11Ø added 1 to the value of BACKGROUND (which was set to 0 in line 8Ø) 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 Ø Ø

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 Ø 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 12Ø sends the program to continue at line 1ØØ 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 11Ø 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 1ØØ

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:

```
→120 IF BACKGROUND < 16  THEN GOTO  100 ↵
```

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:

```
8Ø LET BACKGROUND = Ø
9Ø LET TEXT = 8
1ØØ COLORT BACKGROUND TEXT Ø Ø
1Ø5 WAIT TIME 1ØØ
11Ø LET BACKGROUND = BACKGROUND + 1
12Ø IF BACKGROUND < 16 THEN GOTO 1ØØ
```

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 13Ø will only be executed when the condition for the computer to jump back to line 10Ø (in line 12Ø) 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 13Ø does exactly that.

Line 14Ø 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:

```
8Ø LET TEXT = Ø
9Ø LET BACKGROUND = Ø
1ØØ COLORT BACKGROUND TEXT Ø Ø
1Ø1 PRINT "BACKGROUND = ";BACKGROUND,"TEXT = ";TEXT
1Ø5 WAIT TIME 1ØØ
11Ø LET BACKGROUND = BACKGROUND + 1
12Ø IF BACKGROUND < 16 THEN GOTO 1ØØ
13Ø LET  TEXT = TEXT + 1
14Ø IF TEXT < 16 THEN GOTO 9Ø
15Ø COLORT 15 Ø Ø Ø :  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 Ø to 3 (allowing a choice of four different colours) and four bits are required to represent numbers Ø 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 (Ø). To put a green dot in the left corner at the bottom of the graphics area type:

→DOT Ø,Ø 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:

→6ØØØØ MODE 2 ↵

→6ØØ1Ø GOTO 6ØØ1Ø ↵

→RUN 6ØØØØ ↵

(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 6ØØØØ will therefore need to be changed for the other MODE's) and then endlessly loops (line 6ØØ1Ø) 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, 2Ø 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 Ø 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...

PART II

TABLE OF CONTENTS

1.

GENERAL DESCRIPTION

The DAI Personal Computer is designed to provide the maximum capability that can economically be provided to an individual. The design is realised such that programs are loaded from a low cost audio cassette or a floppy disc. The results of program execution are output to the user via an antenna connector for PAL, SECAM or NTSC standard television receiver. The Graphical Sound Generation also outputs two tracks of separated sound for left and right stereo connections, and the sound channel of the television.

The resources of the DAI Personal Computer are partitioned into four segments; the Microcomputer Section, Programmable Graphical Video Section, the Sound Generator Section and the I/O Section. To optimise usage of components within the design, considerable overlay of logic usage exists within the system. Figure 1 is a logical block diagram of the DAI Personal Computer.

The resident software is comprised of six major modules, Basic Interpreter, Math Package, Screen Driver Module, Keyboard Scan + Encode Routine, the Machine Language Utility and the General House-keeping Module.

The Basic Interpreter incorporates most of the features found in other Personal Computers as well as special statements to control the video graphics and sound generator and interface with the Machine Language Utility as well as assist with generation and editing of source programs. In order to obtain the minimum possible execution time the design of the Basic System is such that it functions as a quasi-interpreter. When the user types in his source program it is compressed and encoded into a special "run-time" code so that the Execution Routine has the smallest possible amount of work left to do.

The Math Package is broken into an Integer Math Module and a Floating Point Math Module. The integer module performs only basic operations as +, -, multiply etc., while the Floating Foint Math Module provides these plus transcendental functions.
Integer variables are calculated to nine digit resolution and floating point variables to 6 digit resolution. The Math Package handles floating point numbers in the range $\pm 10^{-18}$ to $\pm 10^{18}$, and zero. When the Scientific Math option is inserted into its socket the Math Package automatically uses it for calculations instead of the software calculation modules.

The Screen Driver Module is responsible for arranging the data in memory to give a correct picture in all modes. It also handles the changing of screen colours, the drawing facilities (DOT, FILL, DRAW) and other screen-related facilities.
The Keyboard of the DAI Personal Computer is a simple matrix of 56 keys connected in an 8 x 7 matrix. The Keyboard Scan + Encode Routine scans the keyboard at fixed time intervals, detects key depressions and encodes a specific key according to a look-up table. Since the keyboard of the DAI Personal Computer has been constructed in this fashion it is possible to provide DAI Personal Computers with other configurations and codes. The keyboard driver software provides for a 3 key rollover mechanism.

The Machine Language Utility is a complete set of keyboard and subroutine callable functions that permit and assist the generation, loading, de-bugging, and execution of machine language programs and subroutines. The control subroutines and housekeeping subroutines of this module allow direct interface between BASIC programs and machine language program and subroutines. An unlimited number of machine language subroutines may be called by a BASIC program.
The General Housekeeping Module is a set of routines that are shared by other modules, providing for instance, the control of memory bank switching. This allows the 8080A microprocessor to operate with 72K bytes of memory instead of the 64K normally.

1.1
Summary of features

1.1.1
Microcomputer

8080A microprocessor running at 2MHz.
8K, 12K, 32K, 36K, 48K RAM memory configurations
24K PROM/ROM capability (software bank switched)
Memory mapped I/O
AMD 9511 math chip support logic
Hardware random number generator
Stack overflow detect logic.

1.1.2
I/O Devices

ASCII Keyboard
PAL/SECAM/NTSC/VIDEO TV connection via antenna input (color and B/W)
Sound channel audio modulated on TV signal.
Dual low cost Audio cassette input and output with stop/start control.
Stereo hi-fi output channels
Left and Right game paddle inputs (6 controls)
Interface bus (DAI's DCE-BUS) to:
- floppy disk controller
- printer controller
- standard interface cards (DAI's RWC family)
  - IEEE bus adaptor
  - communication interconnections
  - control connection
  - prom programming
  - special interfaces
  - analog input and output

RS232 Interface
- Programmable baud rates
- Terminal or modem function

1.1.3

Graphical Video

Character screen mode (66 characters x 24 lines normally 11/22/44/66 characters + 13 to 32 lines possible)

16 colors or grey scales

Multiple resolution graphics modes (software selectable)

65 x 88

130 x 176

260 x 352

(Intermixed mode screens of lines of characters and graphics are possible).

True"square" graphics.

1.1.4

Graphical Sound

3 independently programmable frequencies

1 programmable noise generator

Amplitude and frequency software selectable

smooth music

random frequencies

enveloped sound

vocal sound generator

1.1.5

Resident Software

Extended Highspeed BASIC interpreter

Full floating point scientific math commands.

Hardware scientific functions automatically used if math module present.

Graphical video commands

full graphic plotting

arbitrary line specification

arbitrary dot placement

filling of arbitrary rectangles

Graphical sound commands

predetermined volume envelope specification

individual specification of frequency

individual specification of volume

individual specification of tremolo

individual specification of glissando

Machine Language Utility.

1.1.6

Compatible System Software

DAI Assembler

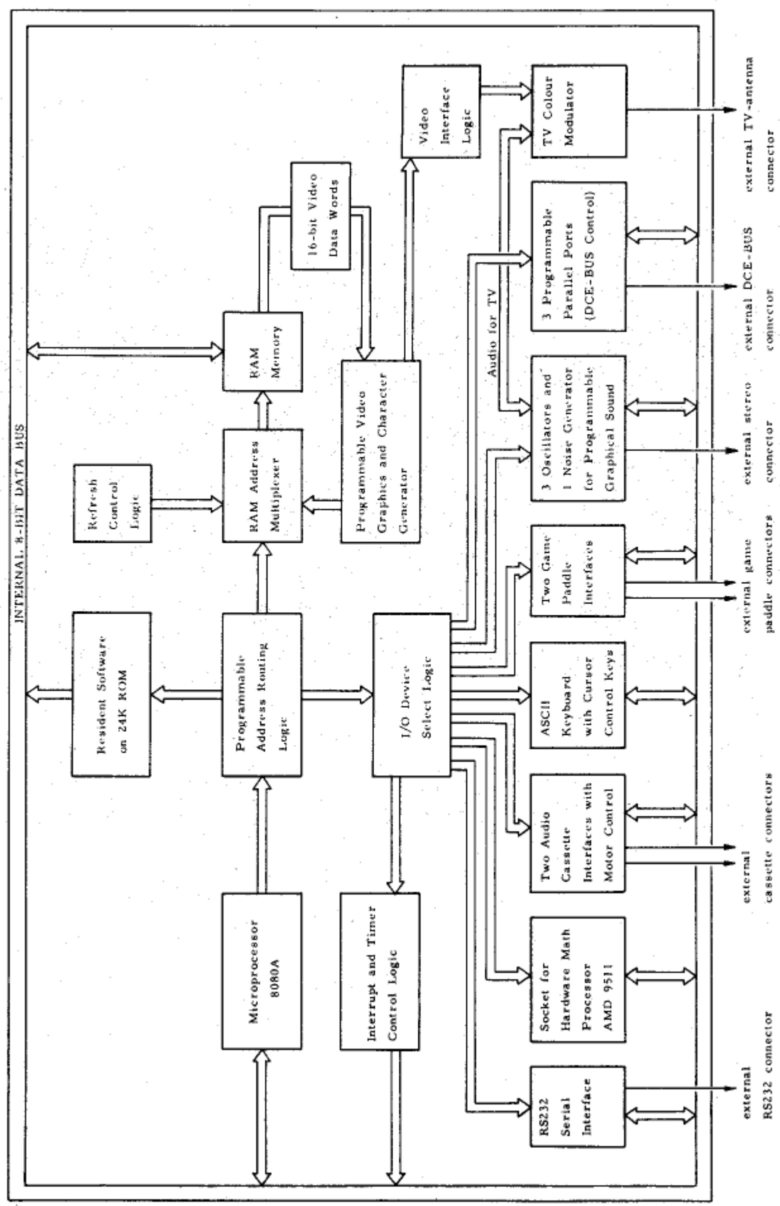
8080A Standard software support

FORTRAN Compiler support

MDS/Intellec non-disc software support.

1.1.7

Functional Block Diagram



2.0 MICROCOMPUTER

2.1 Introduction

The DAI Personal Computer's processor section is designed around the 8080A Microprocessor. The design is based upon the popular and economical high performance DCE microcomputer architecture. The microcomputer section consists of the microprocessor and timing circuitry; the ROM and Static RAM memory; Interrupt Control and Interval Timer logic; and the Master RAM memory. The Master Ram memory consists of a dynamic memory that is configurable from 8K bytes up to 48K bytes.

2.2 Memory Usage

The DAI Personal Computer's memory space is organised on the basis of memory mapped input-output which allocates normal memory addresses to all I/O operations alongside the RAM and ROM memory addresses that are required for normal system operation.

In the following descriptions the address space is described in terms of hexadecimal numbers where the available range of 64 kilobytes is represented by the address range 0000 to FFFF. Switched banks represent a duplication of addresses.

0000	-	003F	INTERRUPT VECTOR
0040			CONTROL OUTPUT IMAGE
0041	-	0061	UTILITY WORK AREA
0062	-	0071	UTILITY INTERRUPT VECTOR.
0077	-	00CF	SCREEN VARIABLES
00D0	-	00FF	MATH WORK AREA

0100 - 02EB		BASIC VARIABLES
02EC TO TOP OF RAM (VARIABLE BOUNDARIES)		HEAP(STRINGS + ARRAYS)
		PROGRAM (COMPILED BASIC)
		SYMBOL TABLE
		NOT USED RAM
		SCREEN DISPLAY
F800 - F8FF		uC STACK

The following two byte variables are maintained by the system.
Addresses are stored on low order byte, high order byte (8080A)

Address (Hex)	Variable
ø29B	START OF HEAP
ø29D	SIZE OF HEAP
ø29F	START OF PROGRAM BUFFER
ø2A1	END PROGRAM BUFFER AND START SYMBOL TABLE
ø2A3	END SYMBOL TABLE
ø2A5	BOTTOM OF SCREEN RAM AREA

2.3
Timer and Interrupt Control

The DAI Personal Computer has 5 interval Timers programmable from 64 µs to 16 ms, 2 external interrupts and 2 serial I/O interrupts. These are priority encoded with a masking system and allow an automatic or polled interrupt system to be used.

2.3.1
Interrupt Control

The 8 interrupt vector addresses provided by the 8080 are assigned the following functions:

Vector Address (Hex)	Allocated function
00	Timer 1
08	Timer 2
10	External interrupt
18	Timer 3
20	Receive buffer full
28	Transmit buffer empty
30	Timer 4
38	Timer 5/auxiliary interrupt

The external interrupt is connected to a signal which indicates that the address range F000 to F7FF has been accessed. This condition normally indicates a "stack overflow" condition.
The auxiliary interrupt is connected to a page signal from the TV picture logic. This provides a convenient 20 ms clock for timing purposes.
More complex features of this part of the logic are beyond the scope of this manual, and anyone needing such information should refer to the DAI publication "DCE MICROCOMPUTER SYSTEMS DESIGNER'S HANDBOOK".
The programming advice given on the TICC is valid also for Personal Computer systems. The access to the keyboard is also via the same logic, using the associated parallel input and output ports.

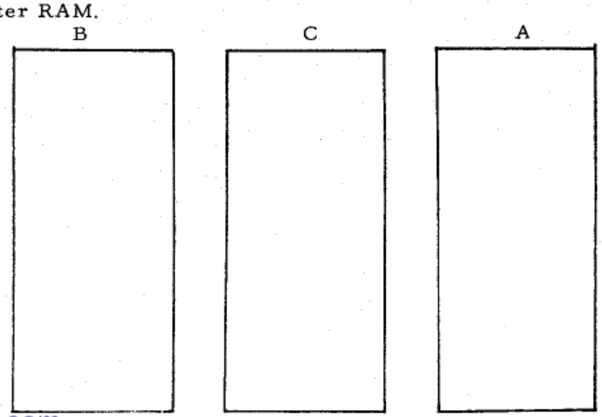
2.4

Master RAM Memory

The Master RAM memory is divided into three separate memory banks, called A, B, C. With one restriction each RAM memory may contain 4K or 16K dynamic RAM chips or they may be left empty. This yields a total RAM availability from 8K to 48K bytes.

The addressing of the dynamic RAM is controlled by a single PROM programmed to correspond to the physically present RAM configuration. The exchange of this chip and changing of a switch is the only operation, other than replacement of RAM chips, that is necessary to implement a configuration change.
The RAM memory is seen by the program as a continuous block of memory starting at (hex) address 0000 up to a maximum address which for 48K is BFFF.

The first RAM bank, (if present) starts at address 0000 and is available for program use only and may not contain display data. The remaining two banks which must both be present are arranged for 16 bit (two-byte) wide access by the display controller. Bank B contributes the low-order bits, and bank C the high-order bits of the 16 bit word. For processor access even-address bytes are in bank B and odd-address bytes are in bank C, e.g. : if bank A is 4K and occupies addresses 0000 to 0FFF then address 1000 is in bank B, address 1001 is in bank C etc. to the end of the Master RAM.



2.4.1

Programmable RAM select Logic

For each RAM configuration of the DAI Personal Computer it is necessary to define the address decoding. This is achieved using a single factory programmable ROM. These are supplied for each defined RAM configuration.

RAM configuration	Banks B+C address	Bank A
8K	0000 - 1FFF	not used
12K	1000 - 2FFF	0000 - 0FFF
32K	0000 - 7FFF	not used
36K	1000 - 8FFF	0000 - 0FFF
48K	4000 - BFFF	0 - 3FFF

No other aspect of the machine is altered by changes to the RAM configuration.

2.4.2

Master RAM Configurations VS Graphical Capability

Master RAM Configuration	Graphical Resolution	Display Color Modes	Required Picture Space	Available Prog. and Work space	Notes
8K	65 x 88	4 16	1.5K	6.5K	
	130 x 176	4 16	5.8K	2.2K	
12K	65 x 88	4 16	1.5K	10.5K	
	130 x 176	4 16	5.8K	6.2K	
32K	65 x 88	4 16	1.5K	30.5K	
	130 x 176	4 16	5.8K	26.2K	
	260 x 352	4 16	22.8K	9.2K	
36K	65 x 88	4 16	1.5K	34K	
	130 x 176	4 16	5.8K	30K	
	260 x 352	4 16	22.8K	13K	
	240 x 528	4 16	32K	4K	

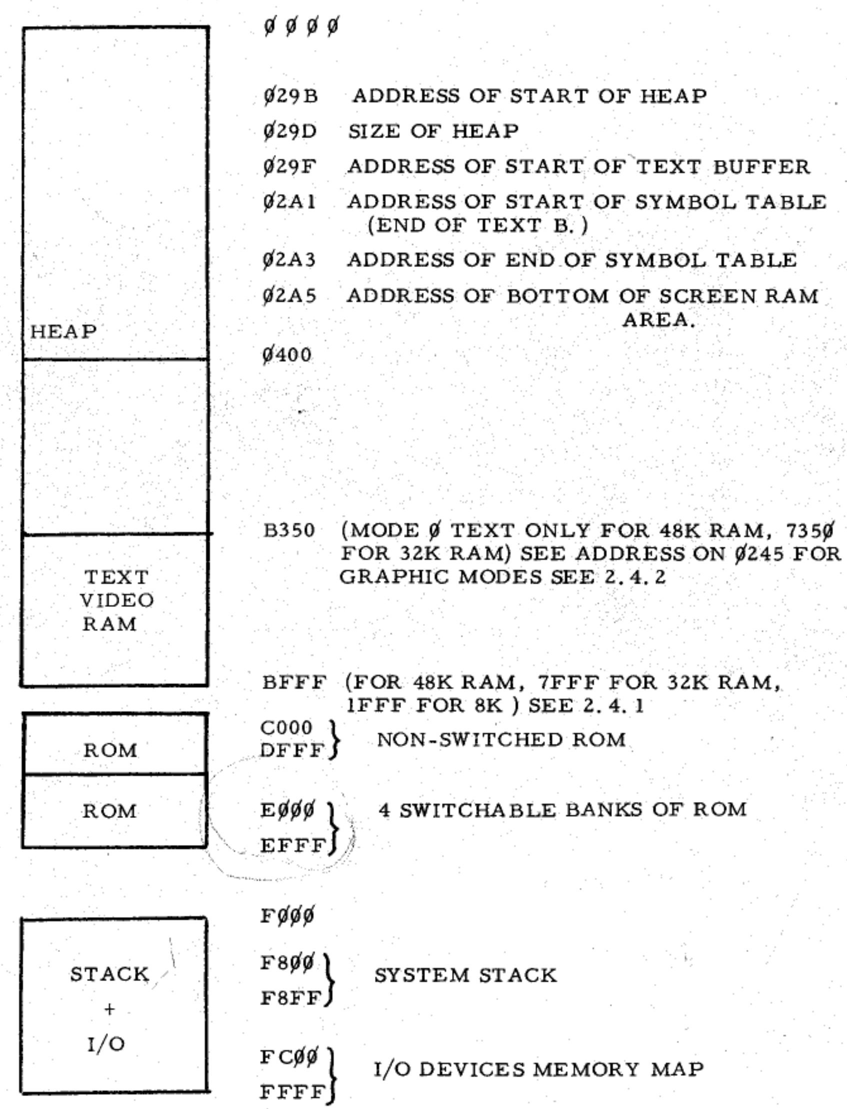
48K	65 x 88	4	16	1.5K	46.0K	
	130 x 176	4	16	5.8K	42.0K	
	260 x 352	4	16	22.8K	25.0K	
	240 x 528	4	16	32 K	16.0K	non-square

The above are examples of the RAM requirement for possible all-graphics screen configurations. Actual usage will be affected by the screen driver package used.

2.5 ROM and Static RAM Memory

The system software resides in mask programmed ROM'S starting at address C000 and extending to EFFF. Addresses C000 through DFFF are continuous program space while addresses E000 through EFFF have four switchable BANKS of program space. Total program ROM space is therefore 24K bytes. In the address range F800 to F8FF a bank of static RAM is included for use by the 8080A stack, and for a vector of jump instructions that allow the emulation of an MDS system.

2.5.1 Simplified memory map (48K RAM P.C.).



3.0
PROGRAMMABLE GRAPHICS GENERATOR

3.1
Introduction

The programmable video graphics + character system makes use of a scheme of variable length data to give efficient use of memory when creating pictures.
A few definitions are necessary before further examination of the scheme.

A "Scan" is:
One traverse of the screen by the electron beam drawing the picture. (there are 625 in a European television picture).

A "Line" is:
A number of scans all of which are controlled by the same information in the RAM.

A "Mode" is:
One of the different ways information may be displayed on the screen. For instance, in "character mode" bytes in memory are shown as characters on the screen, in "4 colour graphics" mode, bytes describe the colour of blobs on the screen.

A "Blob" is:
The smallest area on the screen whose color can be set (The physical size of a blob is different in different screen modes).

A "Field" is:
A set of 8 blobs whose colour is controlled by a pair of bytes from memory.

The picture is defined by a number of lines, one after another down the screen. Each line is independent of all others and may be in any of the possible modes.
At the start of each line two bytes are taken from memory which define the mode for that line, and may update the colour RAM two bytes. These are called respectively the Control and Colour Control bytes. The rest of each line is colour or character information, and the number of bytes used for it is a characteristic of the particular mode. (see example programs).
The screen can operate at a number of different definitions horizontally (e.g. blobs/scan). In the highest definition graphics mode there are 352 visible blobs across the screen. The two lower definitions have respectively 1/2 and 1/4 of this number. There are about 520 scans visible on a "625 line" television, and the screen hardware can only draw (at minimum) 2 scans per line, due to the interlacing. This gives a maximum definition of 260 by 352 which is close to the 3:4 ratio of the screen sides. Thus circles come out round !
Characters are fitted onto this grid by using 8 columns of blobs per character, the dot positions being defined for each character by a ROM. This allows 44 characters per line maximum (or 22/11 in lower definition modes).
A fourth horizontal definition provides for a "high density" character mode with 66 characters/line.
A total of 16 different colours, including white and black can be displayed by the system. Whenever a 4 bit code is used to describe a colour, it selects from this range of possibilities. In some modes (characters + or four colour graphics) a set of 4 of these colours (not necessarily distinct) are loaded into a set of "colour registers". Any 2 bit code describing a colour selects an entry from these registers.
Vertical definition is set by a 4 bit field in the control byte. In graphics modes this simply allows repetition of the information to fill any even number at scans from 2 to 32. In character mode it defines the number of scans occupied by each line of characters; thus the vertical spacing on the screen can be changed to allow anything between an 8 x 7 (the sensible minimum) and 8 x 16 character matrix, giving between 35 and

15 lines of characters on the screen.

Arrangement of information in memory

The first byte of information for the screen is located at the top of an 8K or 32K block of memory. Successive bytes follow at descending addresses. The screen takes memory and displays a picture on the screen accordingly until the whole screen has been filled. It then starts again at the first byte.

3.2 Screen Data Format

At the beginning of the data for each line, two bytes of data represent the lines control word. The control word defines the raster scan depth of the line, the horizontal graphical resolution of the line and selects the display mode of that particular line. Subsequent to this control word a number of data w ords are stored that represent the colour of pixels,or definition and colour of characters according to the selected display mode.

3.2.1 Control Word Format

3.2.1.1 High Address Byte (Mode byte)

Bits 7,6 — Display Mode Control
5,4 — Resolution Control
3,2,1,0 — Line Repeat Count

Line Repeat Count

The line repeat count controls the number of horizontal raster scans for which the same data will be displayed. Since interlace of the TV scan is ignored a minimum of two raster scans correspond to a line repeat count of zero. Thereafter, each additional repeat adds two scans to the line. The maximum programmable depth of any horizontal display segment is thus 32 scans. (European TV sets will show approximately 520 scans total for a full picture).

Resolution Control

The resolution control bits allow selection of one of four different horizontal definitions for display of data on the TV screen for each individual line.

Code (Bit 5, Bit 4)	Definition (pixels per screen width)
00	88 (Low definition graphics)
01	176 (Medium definition graphics)
10	352 (High definition graphics)
11	528 (Text with 66 characters per line) (Screendriver uses 60 characters for text). (Could be used for a very high definition graphics mode).

Mode Control

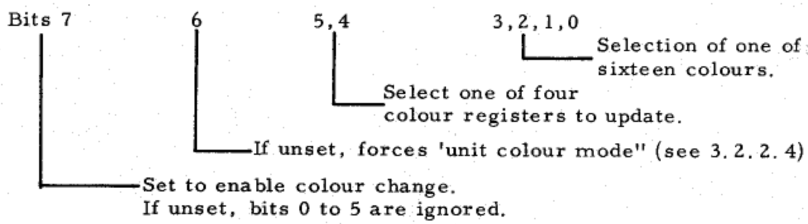
The mode control bits determine how data will be used to generate the picture for that particular segment.

Code (Bit 7, Bit 6)	Display mode
00	Four colour graphics
01	Four colour characters
10	Sixteen colour graphics
11	Sixteen colour characters

3.2.1.2

Low Address Byte (Colour type)

The Low Address control byte is used to store colours into a set of 4 "colour registers" for the four colour mode. Any one of the four colours in the registers can be changed at the beginning of any line of display data. Only the colours in these registers can be displayed in any 4 colour mode. The four colours are freely selectable from the sixteen colours defined in Colour Select Table.



Code	Code
0	Black
1	Dark blue
2	Purple Red
3	Red
4	Purple Brown
5	Emerand Green
6	Kakhi Brown
7	Mustard Brown
8	Grey
9	Middle Blue
10	Orange
11	Pink
12	Light Blue
13	Light Green
14	Light Yellow
15	White

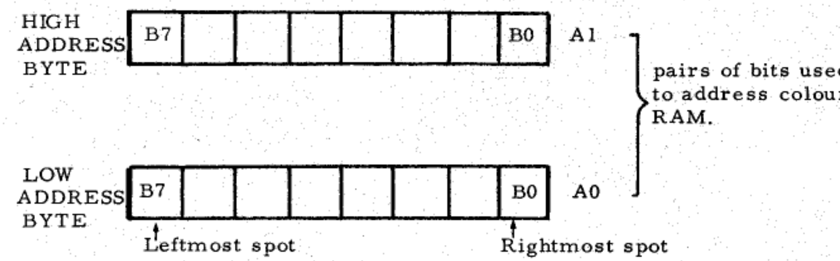
3.2.2

Data Mode

3.2.2.1

Four Colour Mode

In this mode only two bits of data are required to define the colour of a pixel. These data bits are obtained in parallel from the upper and lower bytes of each data word using the high order bits first.
The 2 bytes in a field are considered as 8 pairs of bits. Each pair sets the colour for one spot.



The 2 bits for each spot select one of the four colours which have been loaded into the colour RAM by previous Colour Control bytes. So on any line 4 colours are available. On the next line any one of these may be changed for another, and so on.

3.2.2.2

Sixteen Colour Mode

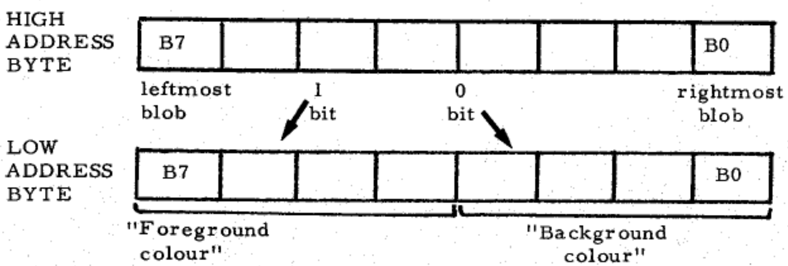
This graphics mode is designed to allow multi-colour high definition pictures in half the memory requirement of other systems.
The basic organization is that the low address byte selects two of the sixteen possible colours.

Bits 0 - 3 "Background" colour.
Bits 4 - 7 " Foreground" colour.

The high address byte than defines by each successive bit whether a colour blob should be foreground or background.

NB

The two bytes in the field serve different purposes, one being used to define two available colours for use in the field, and the other to choose one of these for each spot.



The bit for each spot can select either the "foreground" or the "back-ground" colour. However, what these colours are is totally independent of the preceding or following fields. So any line may use any and all of the total 16 colours. The contents of the colour RAM are irrelevant in this mode.

One additional feature is added to eliminate restrictions of the scheme. After each eight bit field of colour the background is extended into a new area, even if a new background colour is specified, until the new foreground is first used. It is therefore possible to create a required picture by suitable combination of foreground and background.

3.2.2.3

Character Mode

In this mode, characters are generated using a character generator ROM in conjunction with the four colour registers or using any 2 colours for each in the 16 colour character mode.

The usual character matrix is 6 x 9 bits out of a possible 8 x 16. Therefore the line repeat count should be at least eleven, to guarantee full character display plus line spacing.

Four colour characters are produced on the screen in a way similar to the four colour graphics mode, but with the character ASCIV data replacing the high address data byte used for four colours. The result is that characters are displayed using colours from the four colour registers. The data from the character generator ROM control the lower address bit and bits from the low-address byte determine the other. This allows characters on a single horizontal display segment to be in one of two colour combinations of character/background, or even with a vertical striped pattern controlled by the low address byte. However, note that as compared with four colour mode information (but not the low-address byte) is subject to a one character position delay before appearing on the screen.
In character mode the height of the characters is a set number of horizontal scans. The character width is determined by the definition selection in the control byte. A definition of 352 yields 44 characters per line, 528 hields the normal 66 characters per line. Other definitions are possible and they yield wide characters, useful as large capitals in applications such as the power-on message. However, this feature is not supported by the resident BASIC.

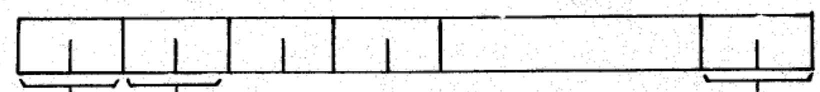
Special characters:

- CR — Terminates a line of characters and positions the cursor at the first position of next line. If necessary, the screen is "rolled up" to make room.
- FF — Fills the character area with spaces and positions the cursor at the start of the tope line on the screen.
- BS — If the current line has some characters on it, then the cursor is moved back to the previous position and the character there is replaced by a space.

- A line of characters on the screen can be extended up to 4 screen widths. Continuations are indented a few characters, and a letter "C" is displayed in the first position of these lines.
- When a third continuation line is full any character except CR, FF and BS is ignored.
- Attempts to backspace past the beginning of the line are ignored.
- If the screen is in "all graphic mode" and character output is necessary then a mode change will be to an appropriate mode including a character area. First the corresponding "split" mode will be tried e. g. if the screen is in mode 1, then mode 1A. If in mode 1 a program claims all free memory (e. g. by using "CLEAR") then mode 1A, which requires more memory than mode 1, will not be possible and the default is to mode 0. In this case the program is deleted by an automatic "NEW" command.

CHANGING LINE BACKGROUND OR LETTERS COLOR ON ONE LINE

Line 1 Control byte is located at address XFEF and line 1 Color Control byte address at XFEE (X being 1 for 8K machine, 7 for 32K machine, B for 48K machine). The first character byte of line 1 is located at line 1 Control byte address minus 2, and the character Colour Control byte at line 1 Control byte address minus 3. Each of the 66 positions of the screen is located at line Control byte - (2 * position of character on the line) for the character and at line Colour Control byte - (2 * position of character on the line) for the Colour Control byte of the character. Remember that there are 66 character positions on the screen but that the first and last three characters are kept blank for the margins. Therefore the Control byte for the next line is located at Control byte of previous line (i. e. XFEF) less 134 bytes (#86. So if the Control byte of line 1 is a BFEF, the Control byte of line 2 will be at #BFEF - #86 = #BF69.



Examples:

Control Byte Line 1 #BFEF
Control Byte Line 5 #BFEF - (# 86 * 5) = #BDD7
Colour Control byte Line 5 = #BDD6
Character N° 6 on Line 5 # BDD7 - 6 * 2 = #BDCC
Colour Character 6 of Line 5 = #BDCB
(see VIDEO RAM TABLE and examples 1 and 2)

Use the POKE in your program for changing line background, letter colour, or letter, and Utility 3 for checking the location you intend to POKE (when you return to BASIC the colour changes you made in Utility mode are erased if you enter MODE 1, RETURN, MODE 0.

Example

```
COLORT 8 0 5 10
POKE #BA2D,#DA    (Will change colour of letter from black 0 to
                   colour 10 on line 12)
POKE#BA2D,#C3     (Will change background from 8 to 3)
```

The locations from #x350 to #x35F and #xFF0 to #xFFF
x = 1 FOR 8K RAM, x = 2 FOR 12K, x = 7 FOR 32K, x = B FOR 48K
control the screen background and foreground colours

Example
COLORT 0 15 7 8

```
00 00 B8 3F 00 00 A7 3F 00 00 9F 3F 00 00 80 3F

00 00 B8 36 00 00 A7 36 00 00 9F 36 00 00 80 36
```

*POKE#735A,#90:POKE#7FFA,#90:POKE#735E,#80:POKE#7FFE,#80

You will see the screen black and the letters black
the # numbers 90 and 80 can be replaced by any # number
from #90 to #9F and #80 to # 8f

Changing colour of background and text

Example 1

```
10    MODE 0
15    REM START AT #BEE2 for 48K, #7EE2 for 32K, #2EE2 for 12K, #1EE2 for 8
20    COLORT 3 0 5 15
25    FOR A%=1 TO 23:PRINT A%,:FOR B=0.0 TO 40.0:PRINT "+";:NEXT:PRINT :NEX
30    REM YOU FIND IN    LINE  1 - 2 TEXT COLOUR  0 BACKGROUND  8
35    POKE #BEE2,#CF:REM LINE  3 - 7 TEXT COLOUR  0 BACKGROUND 15
40    POKE #BC44,#DF:REM LINE  8 - 9                15            15
50    POKE #BB38,#D8:REM LINE 10      (no taxt)      8            15
60    POKE #BAB2,#D0:REM LINE 11 -12                 2            15
70    POKE #B9A6,#DF:REM LINE 13 -14 (no text)      15            15
80    POKE #B89A,#D5:REM LINE 15                     5            15
90    POKE #B814,#D0:REM LINE 16                     0            15
92    POKE #B78E,#DF:REM LINE 17 -18 (no text)      15            15
93    POKE #B682,#C6:REM LINE 19 -21                15             6
94    POKE #B4F0,#C8:REM LINE 22 -24                15             8
95    GOTO 95
```

Example 2

```
10    E%=#FF
20    COLORT 8 0 0 8
25    REM START AT #BEE2 for 48K, #7EE2 for 32K, #2EE2 for 12K, #1EE2 for 8
30    B%=#BFEF
40    FOR A%=1 TO 23
50    D%=B%-3
60    FOR C%=0 TO 65
70    POKE D%,E%
80    D%=D%-2:NEXT
90    B%=B%-#86:NEXT
93    E%= INOT E% IAND #FF
95    GOTO 30
```

VIDEO RAM TABLE

Line N°	Start Address of Line (in Hex)	Line Colour Control byte Address (in Hex)
1	XFEF	XFEE
2	XF69	XF68
3	XEE3	XEE2
4	XE5D	XE5C
5	XDD7	XDD6
6	XD51	XD5Ø
7	XCCB	XCC4
8	XC45	XC44
9	XBBF	XBBE
10	XB39	XB38
11	XAB3	XAB2
12	XA2D	XA2C
13	X9A7	X9A6
14	X921	X920
15	X89B	X89A
16	X815	X814
17	X78F	X78E
18	X7Ø9	X7Ø8
19	X683	X682
20	X5FD	X5FC
21	X577	X576
22	X4F1	X4FØ
23	X46B	X46A
14	X3E5	X3E4

X = 1 FOR 8K MACHINE, X = 2 FOR 12K, X = 7 FOR 32K, X = B FOR 48K

3.2.2.4

Unit colour mode

This mode is available for space saving during uniform scans of the picture. A horizontal band of constant colour (or repeated pattern) can be drawn using only one control word and one data word. The data for this mode should be in high speed format.

Using this mode a full screen of data need be no more than 40 bytes of ram.

3.3

Video Interface

The television interface is realized such that a separate adaptor module plugs into the fundamental logic to realize normal Black and White interface, standard colour modules of PAL, SECAM or NTSC and video monitors. Other video interfaces are easily realizable by construction of an adaptor that plugs into the video interface connector of the DAI personal computer.

4.0
PROGRAMMABLE GRAPHICAL SOUND GENERATOR

4.1
Introduction

The sound generator of the DAI Personal Computer has considerable flexibility because every frequency is generated by digital oscillators that yield precise results. Additional random noise generation and digital volume controls complete the system.

4.2
Programmable Oscillators

The Programmable Graphical Sound Generator is realised via three independent programmable oscillators and a random noise generator. Each oscillator is connected as an I/O device to the microprocessor and is programmable to any frequency within the range 30 HZ to 1MHZ. Obviously the higher frequencies are not interesting for audio work but since the three oscillators are added together before modulation of the audio channel of the TV interesting effects can be obtained by beating together various possibilities.
The programmable oscillators are used for sound generation and game paddle interfaces.

4.2.1
Frequency Selection

In order to program a frequency into one of the channels a 16 bit number must be sent to one of the following addresses:

Oscillator Channel	Device Address
1	FC00 or F001
2	FC02 or F003
3	FC04 or F005

Prior to sending a frequency to a channel, address FCØ6 must be loaded with the following 8-bit data words:

1	36 Hex
2	76 Hex
3	B6 Hex

The 16 bit frequency data is sent as two 8-bit transfers to the specified address sending least significant byte first.

4.2.2
Volume Control

The amplitude of the oscillator output as well as that of the noise generator is digitally controllable by writing a control word to the address specified in I/O device allocation section.

4.3
Random Noise

A noise generator circuit is included within the sound generation circuitry. The purpose of this device is to simulate as near as possible white noise for the purpose of complex sound generation and to provide a time random sequence for random number generation. Random events generated by this circuit provide the basis for information input on an I/O port to generate a true random number.

4.4
Frequency Mixing

All sound channels as well as the output of the noise generator are added together before modulation of the audio channel. Channels 1 and 2 and 2 and 3 are added together for left and right stereo output. For the stereo configuration noise is inserted in Channels 1 and 3.

4. 5

Frequency Calculator Formula

To output a frequency of nHz from a given oscillator, program it with an integer equal to 2×10^6 divided by n. A special BASIC function (FREQ.) performs this calculation when required.

5. 0

INPUT-OUTPUT SECTION

5. 1

Introduction

All input-output of the DAI Personal Computer is arranged on a memory mapped basis. I/O is thus directly accessible to BASIC programs, however care is necessary to avoid conflict with the BASIC interpreter activity when using POKE commands.

5. 2.

Game Paddle Interface

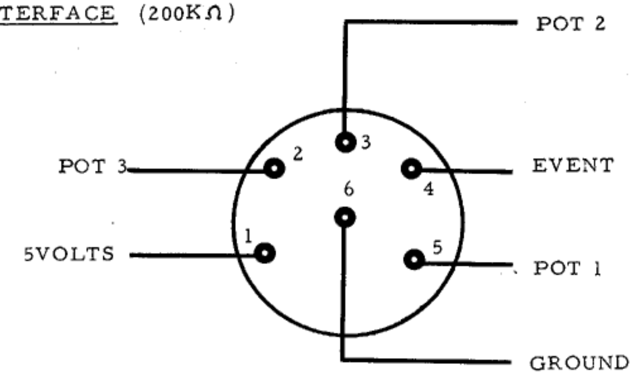
The Personal Computer is equipped with circuitry required to connect two game paddles as input devices. Each paddle contains three variable resistors whose positions are read as values and one on-off event (single contact switch).

The position of any paddle resistor is found by putting its binary address onto the 3 bits in port FD06. Then channel 0 of the sound generator is put into a mode such that it operates as a counter. The read of the positions is triggered by reading location FD01. The value is read out and mapped onto an 8 bit range for a result.

DIN PLUG CONNECTIONS FOR DAI PERSONAL COMPUTER

(6 PINS DIN PLUG 240° VIEWED FROM INSIDE OF THE PLUG OR TO THE COMPUTER PLUG)

PADDLE INTERFACE (200KΩ)

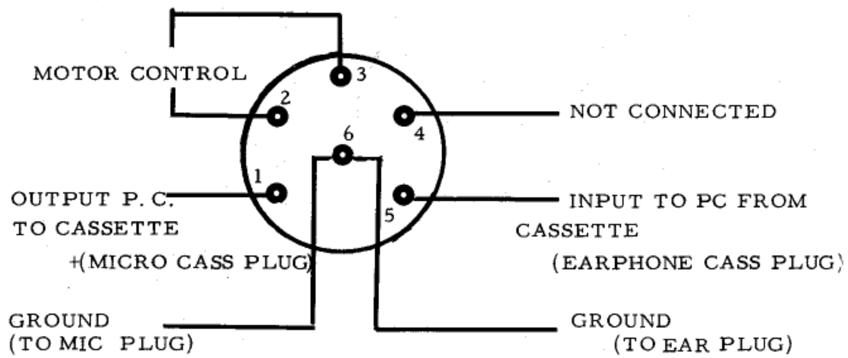


5.3

Audio Cassette Interface

The Personal Computer of DAI contains the entire logic and interface circuits needed to connect a low cost audio cassette for the input and output of data and programs.
The Personal Computer input from the cassette should be made via the crystal ear phone outlet or the external speaker outlet. In these cassettes that have no such outputs simply connect the speaker wires to the Personal Computer input.

DIN PLUG CONNECTIONS FOR DAI PERSONAL COMPUTER
(6 PINS DIN PLUG 240° VIEWED FROM INSIDE OF THE PLUG OR TO THE COMPUTER PLUG)
CASSETTE RECORDER INTERFACE

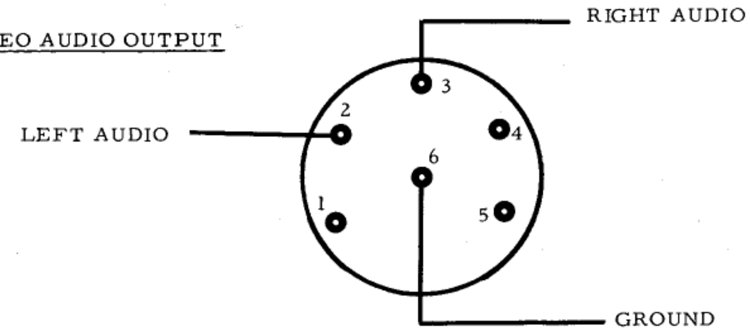


5.4

Stereo Output

The DAI Personal Computer Graphical sound Generator is connectable to the left and right channels of a stereo set. Channels 0 and 1 and channels 2 and 3 are summed to make the left and right channel respectively.

STEREO AUDIO OUTPUT



5.5

Scientific Math Peripheral

As an option for high speed calculations the logic of the DAI Personal Computer supports the Scientific Math Chip of Advanced Micro Devices (9511).
The device is addressed at locations FB00 (data) and FB02 (command and status). The "PAUSE" signal is correctly used to make the CPU wait for data. Note that the SHLD and LHLD instructions are not usable with this device for double byte transfers.

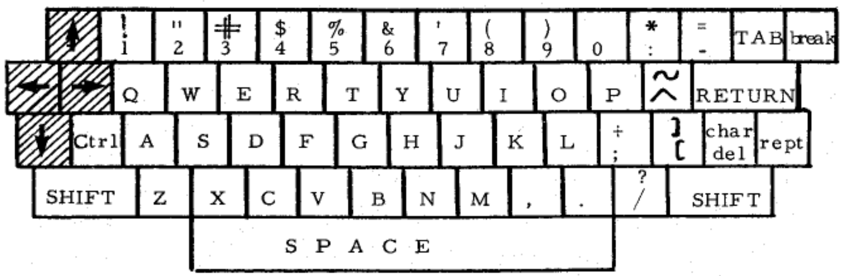
5. 6

ASCII Keyboard

The ASCII keyboard is scanned as a matrix of switches. Encoding, debouncing and roll-over are realized via a software routine.

5. 6. 1

Keyboard Layout



The keys are assigned to rows and columns.

ROWS Output lines (FF07)	0	1	2	3	4	5	6
0	0	8	re-turn	H	P	X	↑
1	1	9	A	I	Q	Y	↓
2	2	:	B	J	R	Z	←
3	3	;	C	K	S	[	→
4	4	,	D	L	T	^	Tab
5	5	-	E	M	U	space bar	ctrl
6	6	.	F	N	V	rept	break
7	7	/	G	O	W	char del	shift

COLUMNS

Input lines (FF01)

5. 6. 2

Keyboard Scan Logic

The Personal Computer contains a software keyboard scan and encoder. This can be used by other programs which may use the standard key encoding tables, or supply their own.
All keys are scanned periodically, and action is taken when a key is noticed to have been newly pressed. Alternatively, if the repeat key is pressed, then periodically all currently pressed down keys are acted on. The repeat speed is fixed.

The actual code for the key is obtained from a table. The "shift" system selects which of two possible tables to use. By setting a flag byte the keyboard handler can be made to scan only for the "BREAK" key which obviously takes less time.

On initialisation the alphabetic keys (A - Z) give capital letters if unshifted, and small when shifted. Pressing the "CTRL" key inverts this arrangement to give a "type-writer-like" effect. Successive uses invert each time.

The standard codes returned by each key: see decimal/characters table end of this book.

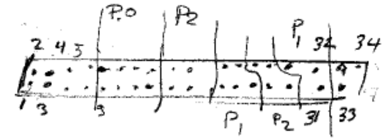
5.7

DCE-BUS

The DAI Personal Computer provides the possibility of external connection by flat cable of a DCE standard bus. The provided logic drives the bus exactly as a standard DCE Processor with the same addressing and characteristics including reset and interrupt lines. * The DCE bus can be connected directly to external equipment.
Included in the Personal Computer are routines to communicate with DAI Real-World-Cards. Note that the interface to these routines is different from that in some other DAI software.
Example routines follow in 6.2.15 third page. Note that the internal logic of the routine is subject to changes. Only the interface is guaranteed.

EXAMPLE OF ROUTINE TO DRIVE A PARALLEL PRINTER THROUGH DCE-BUS

```
10  CLEAR 1000 : REM MUST BE SET FOR YOUR PROGRAM
20  DIM PRI (10)
30  INPUT "TYPE J IF YOU WANT A PRINT" ; A$ : PRINT
40  IF A$<> "J" GOTO 100
50  FOR X = #400 TO 419
55  READ C
60  POKE X,C
65  NEXT X
70  POKE # FEØ3,# AC
75  POKE # 2DD, # C3
80  POKE # 2DE, # ØØ
85  POKE #2DF, # 4
90  DATA 229,213,197,17,2,254,6,16,33,1,254
95  DATA 119,43,54,Ø,54,1,26,160,194,11,4,193,209,225,201
100 PRINT CHR$ (12)
110 IF A$ <> "J" GOTO 200
120 IF A$ = "J" THEN POKE #131,3 : REM OUTPUT TO DCE-BUS
                                                    ONLY
```

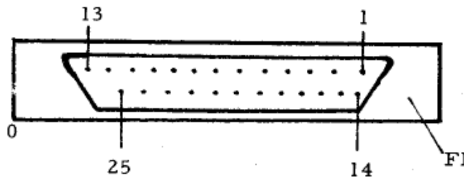



5.7.1

DCE-BUS Pinout

SIGNAL NAME	DESCRIPTION		pin on real-world card	pin on personal comp. card.
P0B0	General Interface PORT 0	Bit 0	24	16
P0B1	data bus	Bit 1	26	14
P0B2		Bit 2		12
P0B3		Bit 3	28	10
P0B4		Bit 4	29	9
P0B5		Bit 5	27	11
P0B6		Bit 6	25	13
P0B7		Bit 7	23	15
P1B0	General Interface PORT 1	Bit 0	12	30
P1B1	CARD SELECT	Bit 1	10	31
P1B2		Bit 2	8	32
P1B3		Bit 3	7	25
P1B4		Bit 4	9	24
P1B5	INTERNAL CARD ADDRESSING	Bit 5	11	23
P1B6		Bit 6	13	22
P1B7		Bit 7	15	21
P2B0	BUS EXPAND General Interface PORT 2	Bit 0	18	26
P2B1	WRITE	Bit 1	17	27
P2B2	READ	Bit 2	16	28
P2B3		Bit 3	14	29
P2B4		Bit 4	19	20
P2B5		Bit 5	20	19
P2B6		Bit 6	21	18
P2B7		Bit 7	22	17
EXINTR+	External Interrupt		4	6
IN7+	Parallel input Bit 7(aux. interrupt)		3	5
EXRESET	External Reset (Ground for Reset)		5	7
+12V	+12V DC		2	2
+5V	+5V DC		1	1
-5V	-5V DC		6	3
INTR	INTERRUPT PIN 14 OF CPU 8080		-	33
IN7+			-	34
NOT CONNECTED			-	8
ground				4

PERSONAL COMPUTER RS-232 CONNECTOR:



FEMALE CONNECTOR
(OUTSIDE VIEW)

PIN	FUNCTION	
1	GND	
2	SERIAL OUT	OUTPUT DATA FROM P. C.
3	SERIAL IN	INPUT DATA TO P. C.
4	DATA TERMINAL RDY	INPUT READY HIGH (5V), NOT READY LOW (ØV)
5	+12V *	
6	+12V *	
7	GND	
8	+12V *	
9 ↓ 25	N. C.	

Note: This connector is wired as fo a terminal and signals to pins 2 and 3 may have to be swapped if it is to send data to a terminal/printer.

* 12V THROUGH 220Ω1/4W.

5.8
RS232 Interface

The Personal Computer has an RS232 compatible interface giving a serial input line, serial output line and a status line to halt output (DTR). These are available on a CCITT standard connector at the rear of the machine. The DTR signal allows synchronisation of the output with a printer. If unused, then output will be unimpeded.
Interrupts to locations 20 and 28 can be set up for receive and transmit ready. The BASIC interpreter however uses the locations for other purposes.

5.9
I/O Device Address (Allocation Reference)

5.9.1
Master Control Device Address (Hex)

F900 - F9FF	Spare	
FA00 - FAFF	Spare	
FB00/1	Data	Scientific Math Chip
FB02/3	Command	Scientific Math Chip
FC00/1	Channel 0	Graphical Sound Generator
FC02/3	Channel 1	Graphical Sound Generator
FC04/5	Channel 2	Graphical Sound Generator
FC06/7	Command	Graphical Sound Generator
FDXX	See 5.8.2	
FE00/1/2	I/O ports 0/1/2	DCE-BUS
FE03	Command port	DCE-BUS
FFXX	See 5.9.3	

5.9.2

Discrete I/O Device Address (Hex)

ADDRESS	NOTES	IN/OUT	BIT ALLOCATION	
FD00	1	IN	0	-
			1	-
			2	Page Signal
			3	Serial output ready
			4	Right paddle button (1 = closed)
			5	Left paddle button (1 = closed)
			6	Random data
			7	Cassette input
FD01	3	IN	Single pulse used to trigger paddle timer circuit.	
FD04	2	OUT	0, 1, 2, 3	Volume, oscillator Channel 1
			4, 5, 6, 7	Volume, oscillator Channel 2
FD05	2	OUT	0, 1, 2, 3	Volume, oscillator Channel 3
			4, 5, 6, 7	Volume, random noise

Cont. . . . .

ADDRESS	NOTE	IN/OUT	BIT ALLOCATION	
FD06	3	OUT	0	Cassette data out
			0, 1, 2	Paddle channel select code
			3	Paddle enable bit
			4	Cassette motor 1 control (0 = run)
			5	Cassette motor 2 control (0 = run)
			6, 7	ROM bank switch

Notes:

1 User may read from or write to any of these addresses at will. No harm can result.

2 Reading from these locations does nothing.
Writing to them will modify the appropriate volume settings, but if the BASIC system accesses the channel the effect may be lost, as it has an internal memory of its own last set value.

3 These locations should not be written into.

5.9.3

Serial I/O, timer & interrupt control

The detail given here is sufficient to allow use of the serial I/O. All these facilities are given by one LSI component, and the BASIC interpreter uses many of the facilities itself. So care must be taken not to disturb the normal running of the system.

ADDRESS	NOTE	FUNCTION
FF00	1	Serial input buffer Contains the last character received on the RS232 interface.
FF01	1	Keyboard input port Bottom 7 bits are data input from the keyboard. Bit 7 is the IN7 line from the DCE-BUS and is attached to the page blanking signal for the TV.
FF02	2	Interrupt address register
FF03	1	Status register Bit allocations: 7,6,5 Not useful 4 Transmit buffer empty Set if RS232 output ready to accept another character. 3 Receive buffer loaded Set if a character has been received 2 Overrun Set if a character has been received but not taken by the CPU. 1 Frame error Set by a "BREAK" on RS232 input
FF04	2	Command register
FF05	3	RS232 Communications rate register

Send (Hex)	for	
1/81	110 baud	2/1 stop bits
2/82	150 "	"
4/84	300 "	"
8/88	1200 "	"
10/90	2400 "	"
20/A0	4800 "	"
40/C0	9600 "	"

Underlined is usual one to use.
Other combinations not useful

ADDRESS	NOTE	FUNCTION
FF06	3	Serial output Write byte to this location to send it on RS232 output. Use only when address FFØ3 bit 4 HIGH
FF07	4	Keyboard output port Data output to scan keyboard. Not useful to user.
FF08	2	Interrupt Mask register
FF09, FF0A, FF0B, FF0C, FF0D	2	Timer addresses

Notes:

1 May be read but not written to by user

2 Should not be accessed by user

3 May be written but not read by user

4 May not be read, writing is harmless and useless ! System keyboard scanner will overwrite user data.

6.0
RESIDENT SYSTEM SOFTWARE

6.1
Introduction

The resident software is comprised of major modules, Basic Interpreter, the Machine Language Utility, and the General Housekeeping Module. Under normal system operation they work together to allow use of BASIC programs from cassette. For machine code programs major functions available as subroutines.

6.2
Resident DAI BASIC

6.2.1
Alphabetic Index of DAI BASIC Statements

6.2.1.1
BASIC Commands

6.2.1.2

BASIC Functions

ABS	6.2.14.1	LOG	6.2.14.15
ACOS	6.2.14.2	LOGT	6.2.14.16
ALOG	6.2.14.3	MID$	6.2.14.17
ASC	6.2.14.4	PDL	6.2.7.4
ASIN	6.2.14.5	PEEK	6.2.7.5
ATN	6.2.14.6	PI	6.2.14.18
CHR$	6.2.14.7	RIGHT$	6.2.14.19
COS	6.2.14.8	RND	6.2.14.20
CURX	6.2.12.10	SCRN	6.2.12.8
CURY	6.2.12.10	SGN	6.2.14.21
EXP	6.2.14.9	SIN	6.2.14.22
FRAC	6.2.14.10	SPC	6.2.14.23
FRE	6.2.11.3	SQR	6.2.14.24
FREQ	6.2.13.6	STR$	6.2.14.25
GETC	6.2.8.2	TAB	6.2.14.26
HEX$	6.2.14.11	TAN	6.2.14.27
INP	6.2.7.12	VAL	6.2.14.28
INT	6.2.14.12	VARPTR	6.2.11.5
LEFT$	6.2.14.13	XMAX	6.2.12.6
LEN	6.2.14.14	YMAX	6.2.12.7

6.2.1.3

Arithmetic and Logical Operators

+, -, *, /, MOD, ↑, =, <, >, <>, <=, >=, IOR, IAND, IXOR, INOT, SHL, SHR, AND, OR.

6.2.2

Format rules and constraints

6.2.2.1

Variables and Numbers

DAI BASIC recognises 2 types of numeric value, integer, and floating point. Integers are whole numbers only, and of restricted range. ± 2 ↑ 32 - 1 (e.g. about 9 digits). However, integer arithmetic is exact and gives no rounding errors. Floating point numbers include non-integer values, and allow numbers whose size is in range 10^{-18} to 10^{18}, with 6 digit printout resolution. (32 bit floating point format).

Various DAI BASIC commands expect either an integer or a floating point value. For example:

a) DRAW A, B C, D X. All of parameters A, B, C, D and X are expected to be integers.

b) LET A = SQRT (B). The parameter B is expected to be a positive floating point number.

DAI BASIC obeys the following rules regarding numerical values:

1) When a floating point value is found where an integer value is required, it is truncated (e.g. 2.3 → 2, -1.7 → -1).

2) When an integer value is found where a floating point value is required, it is converted automatically.

3) Where an integer representation (e.g. "3" not "3.0") is typed in, it will be encoded as a floating point or integer value as the context demands, or if neither is defined, e.g. in "PRINT", as the type set by the "IMP" command.

Variable names have from 1 to 14 characters, of which the first must be alphabetic, and the rest either alphabetic or numeric. Alphanumeric characters after the 14th are ignored. If no type letter ($, %, !) is appended then the type depends on the IMP command. Initially all such variables are floating point.

Numeric variables in DAI BASIC may be either floating point or integer type. Integer variable names are terminated by the character "%", and floating point by " ! ". String variables have "$" as a terminator. But see examples for influence of IMP command.

Examples:

Initially

I, A, S	are floating point, because they are abbreviations of I!, A!, S!
I%, A%, S%	are integer and distinct from I, A, S.
I!, A!, S!	are floating point, and are the same variables as I, A, S.
I$, A$, S$	are string variables.

So if the IMP command is never used, floating point variables can be indicated by leaving off the "type" letter, integer variables by using %, and string by using $.

After IMP INT I-N
IMP STR S-S

I	is an abbreviation for I%, or integer variable
A	is an abbreviation for A! or floating point variable
S	is an abbreviation for S$ or string variable

However any variable with a type letter (I$, A%, S!) is totally unaffected by the IMP command. When the Personal Computer is LISTING a program, it uses the shortest form for a name. In other words after the example above, the variable I% would be printed as just I, S$ as just S, and A! as just A. If the IMP command is used in the form "IMP INT" or "IMP FPT", without a range of letters, then all variable names are defaulted to that type. In addition integer number representations e.g. "3", are interpreted as the required type.

Command	Means same as	"3" is interpreted as	and A as
IMP INT	IMP INT A - Z	Integer 3	A%
IMP FPT	IMP FPT A - Z	Floating point 3.0	A!
IMP STR	Not allowed	-	-

At power on the system does an initial "IMP FPT".

6.2.2.2

Strings

1) A string may be from 0 to 255 characters in length.
2) String arrays may be dimensioned exactly like numeric arrays. For instance, DIM A$ (10,10) creates a string array of 121 elements, eleven rows by eleven columns (rows 0 to 10 and columns 0 to 10). Each string array element is a complete string, which can be up to 255 characters in length.
3) The total number of characters in use in strings and associated control bytes at any time during program execution cannot exceed the amount of string space requested, or an error message will result.
4) Strings cannot contain the character double quote (Hex 22). It can be printed using CHR$ (#22).

Examples of String Usage (Do not forget to make first a CLEAR).

DIM A$(10,10)

Allocates space for a pointer in string space for each element of a string matrix. No furhter string space is used at this time.

A$ = "FOO" + V$

Assigns the value of a string expression to a string variable, requiring string space equal to the number of characters plus one.

IF A$ = B$ THEN STOP

String comparison operators. Comparison is made on the basis of ASCII codes, a character at a time until a difference is found. If during the comparison of two strings, the end of one string is r reached, the shorter string is considered smaller. Note that "A " is greater than " A" since trailing spaces are significant.

INPUT X$

Reads a string from the keyboard. String does not have to be in quotes, but if not leading blanks will be ignored and the string will be terminated on a "," character.

READ X$

Reads a string from DATA statements within the program. Strings do not have to be in quotes, but if they are not they are terminated on a "," character or end of line, and leading spaces are ignored.

PRINT X$

PRINT "F00"+A$

Prints the result of the string expression.

6.2.2.3
Operators

It is obvious that the result of adding I% + J% when I% contains 3 and J% contains 4 should be the integer 7. It is also reasonable to expect I + J where I contains 3.0 and J contains 4.0 to give the floating point result 7.0. Thus some BASIC operators do different things depending on the types of their operands. It is always permitted to give operands of either type to any operator. However the operator may convert either or both operands to another type before use.
Relational operators and the operators "AND" and "OR" produce results of type "logical". These results cannot be assigned to any variables and are only used in "IF" statements.

6.2.2.4
Statements

In the description of statements, an argument of V or W denotes a numeric variable, X denotes a numeric expression and an I, J or K denotes an expression that is truncated to an integer before the statement

is executed. A, B indicate array names without any parameters.
An expression is a series of variables, operators, function calls and constants which after the operations and function calls are performed using the precedence rules, evaluates to a numeric or string value.

A constant is either a number (3.14) or a string literal ("F00").

6.2.2.5
Expressions

The cardinal principle behind the evaluation of expressions by DAI BASIC is that if an expression contains only integer values or variables and operators which work on integers, then at no time is floating point arithmetic used. This gives fast integer arithmetic where it is needed for industrial control and graphics applications.

Order of Evaluation

Expressions in Brackets
↑
*   /   MOD
+   -
SHL   SHR
IOR   IAND   IXOR
>   <   =   < >   <=   >=
AND   OR

Operators on the same level are evaluated from left to right.
E.g. 3 * 5 MOD 2 = 1

6.2.3

Error Reporting

6.2.3.1

Error Report Format

When an error is encountered a message is printed giving details. Under certain circumstances, other information will be given.

(i) If an immediate command has just been input, than no other information is given.

(ii) If a stored program line has just been input, then a reflection of the line with a "?" near the error will be printed.

(iii) If an immediate command is being run, no other information is given.

(iv) If a stored program line is being run, the words "IN LINE NUMBER" and the line number are given.

In case (ii), the line goes into the program with a "***" on the front. (Internally coded as an ERROR LINE)

6.2.3.2

Error Messages Dictionary

CAN'T CONT

There is no suspended program to be "CONTinued".

COLOUR NOT AVAILABLE

A colour has been used in 4 colour mode when it has not been set up by a COLORG command.

COMMAND INVALID

This command cannot be used in a non-stored program line, or in a stored program line, whichever was attempted.

DIVISION BY 0

Integer or floating point divide by 0.

ERROR LINE RUN

A line which gave an error message when it was input has been run without first correcting it.

INVALID NUMBER

The parameter given to a VAL function was not a valid floating point number.

LINE NUMBER OUT OF RANGE

A line number greater than 65535 or zero has been used. (or negative)

LINE TOO COMPLEX

Line typed in would generate more than 128 bytes of encoded program.

LOADING ERROR 0 , 1 , 2 or 3

The program or data requested could not be loaded.

For cassette:

0 means Checksum error on program name.

1 means Insufficient memory

2 means Checksum error on program.

3 means Data dropout error.

NEXT WITHOUT FOR

A "NEXT" statement has been executed without a corresponding "FOR" statement.

NUMBER OUT OF RANGE

Some number has been used in context where it is too large or small.

OFF SCREEN

A point has been referred to which does not exist in this mode.

OUT OF DATA

A "READ" statement has tried to use more DATA than exists.

OUT OF MEMORY

Some attempt has been made to use too much space for the program, symbol table, screen, heap (strings + arrays storage) or edit buffer.

OUT OF SPACE FOR MODE

This message occurs if a program is running in modes 1 or 2, with insufficient free space to run mode 0, 1A or 2A, and attempts to print a message. The system deletes the program by a NEW and prints this message.

OUT OF STRING SPACE

More string space has been used than was allowed for.

OVERFLOW

Integer or floating point overflow.

RETURN WITHOUT GOSUB

A "RETURN" statement has been executed with no corresponding "GOSUB"

STACK OVERFLOW

A line too complex has been typed in, or, too much stack space has been used by a running program.

STRING TOO LONG

A string of over 255 characters has been created.

SUBSCRIPT ERROR

A subscript has been evaluated which is outside the declared range for the array, an array name has been used with the wrong number of parameters, or a dimension of 0 has been requested.

SYNTAX ERROR

Some error in the line just typed in, or the line of data read by an INPUT or READ.

TYPE MISMATCH

Some expression gives a result of an incorrect type for its position. Can occur on input or while a program is running.

UNDEFINED ARRAY

A reference has been made to an array which has not yet been "DIMensioned".

UNDEFINED LINE NUMBER

A reference has been made to a non-existent program line.

6.2.4

Interacting with DAI BASIC

6.2.4.1

Facilities of the Character Screen

When the Personal Computer first prints the message "BASIC" and the prompt, the screen is in what is known as mode 0. That is 24 lines of 60 characters. At any time the screen can be returned to this mode with the command "MODE 0".

The next position where a character will be displayed is indicated by a flashing underline cursor.

Lines on the screen are obviously physically 60 characters long. But when characters are being output the line can be extended with up to 3 "continuation" lines. These have the letter C in column 0 and the first character of those coninuation lines are indented 7 spaces to the right.

The cursor is moved forward when a character is output, and backwards for a backspace (#8) character. Carriage return (#D) ends a line.

The form feed character (#C) has the special effect of entirely clearing the character area (in any mode) and placing the cursor at the top left position.

The tab (#9) character has no special function.

When the third continuation line is used up, further characters output to the screen are ignored, until a carriage return, backspace or form feed. When BASIC is expecting input it only notices characters in positions after the prompt character. If the prompt is deleted with backspaces, then any character put in that position will be ignored, probably causing a syntax error. The colours used for characters are initially set at power on. They can be changed using the COLORT Command.

6.2.4.2

Input of programs and data

When the Personal Computer expects input, it always types a "prompt" character, normally a "* ", but during INPUT commands a "?". The user can then type in characters at will. To delete the last entered character, the "CHAR DEL" key is used. If more information is input than fits across the screen, then it is continued on the following line, indented and with a "C" (for continuation) in column 0. Up to 3 continuation lines may be used, giving a line length of 59 + 53 + 53 + 53 = 218 characters.

Pressing BREAK while typing in commands causes a " " to be printed, and the line is ignored. However during input for an INPUT command, it causes suspension of the program.

6.2.4.3

Amending and running of programs.

When the Personal Computer is ready to accept instructions, it prints a prompt character.

The user can then type in a line of one or more commands, separated by the character ":", and terminated by a "RETURN". The commands will be encoded immediately, and if they have the right syntax, will be run. If the line has a number on the front, it will be encoded as before and placed into the stored program in the machine, according to its line number. It replaces any previous line with that number. If the line is not syntactically correct, an error message will be printed. If there was no line number, no other action is taken. If there was, then a   is is inserted as a dummy first command on the line, and the first 121 characters of the line are encoded as if the line were a REM statement. Attempted execution of the line yields the message "ERROR LINE RUN". A question mark is inserted near the point where the error was detected. The line is then inserted into the program as before.

When the user wishes to run a stored program, he types "RUN", to start at the first line or "RUN 22" to start at line 22.

(for example). The program will then run until some error, or one of the following, occurs:

(i) If an END statement is executed, the program stops. It prints the message: END PROGRAM. The program can only be restarted using RUN.

(ii) If a STOP statement is executed, the program stops. It prints the message: STOPPED IN LINE X with X the appropriate line number. The program is then said to be "suspended".

(iii) If the BREAK key is held down, one of two results will occur:
a) In most circumstances the message BREAK IN LINE X will be printed immediately. The program is then suspended.
b) Under some circumstances, after a pause the system will print: ***BREAK. The program cannot now be restarted.

When a program is suspended, it can be restarted by use of the CONT command. This restarts the program just as if it had never stopped. However any variables etc. changed by the user during the suspension are not restored to their old values.
If the system has cause to report any run-time error to the user, or if the user RUNs any other program or does a SAVE, LOAD, EDIT, CLEAR or NEW, then the suspended program is no longer valid and cannot be CONTinued. If the user tries to do so a message will be printed: CAN'T CONT. When a RUN, SAVE, CLEAR, LOAD, EDIT or NEW command is executed, all variables are reset to 0 (if arithmetic) or a null string (if string). All space assigned to arrays is returned, and any subsequent reference to an array before running a DIM statement for it will give an error.
To delete the stored program the command NEW is used. After this there are no stored lines in the machine and no variables are set to any values.

When a program is suspended the STEP command may be used to continue the program one line at a time. Before each line is executed it is listed to the screen and the machine waits for a space to be typed in on the keyboard.

At power on DAI BASIC defaults into the floating point variable mode where integer variable names must be concluded by the (%) character. A facility to allow this to be switched is provided by the IMP statement. The operator must type in any IMP switches that he desires before he enters his program.

6.2.4.4
Merging of BASIC Programs

CLEAR 10000
LOAD SEGMENT 1 OF PROGRAMS TO BE MERGED
EDIT + BREAK + BREAK
LOAD SEGMENT 2 OF PROGRAMS TO BE MERGED
(THE LINE NUMBERS CANNOT BE THE SAME IN SEGMENTS 1 AND 2)
POKE#135,2

6.2.4.5
Merging of BASIC and machine Language Programs (or routine)(MLP/R)

a) Prepare of the MLP/R and save it after the BASIC program you intend to use with this MLP/R.

EXAMPLE SAVE FIRST YOUR BASIC PROGRAM (see example under of program)

MLP/R
```
10 CLEAR 2000
20 DIM A (20,20)
30 FOR I% = Ø TO 9
40 READ B% : POKE (#2F1 + I%), B% : NEXT
50 SAVEA A "TEST" : STOP
60 DATA #F5,#3E,#FF,#32,#50,#BE,#F1,#C9,Ø,Ø
```
N. B. The size of a one dimension array is (256 x 4) bytes maximum.
In this example the size is (20 x 20 x 4) = 1764 bytes.
The basic program you intend to use must have:

- a CLEAR - a DIM (of the same name and the same array size as the MLP/R - a LOADA (of same name than the MLP/R)
  EXAMPLE of BASIC program that you have on cassette before the MLP/R

```
10 CLEAR 2ØØØ
20 DIM A (2Ø,2Ø)
30 LOADA A
40 CALLM #2F1
50 STOP
```

This program will load the MLP/R after you make a RUN and execute the MLP/R by the CALLM of line 40. You should now RUN 40 each time for calling the MLP/R. You can also delete the first 3 lines by typing 10, RETURN, 30, RETURN.

Important: When the MLP/R has been loaded by the BASIC program do not use the EDIT mode, nor RUN the lines containing the CLEAR, DIM and LOADA commands (in this example you must RUN 40), nor use somewhere in the BASIC program a CLEAR command or a DIM statement with the same array name used for the MLP/R.

When using an MLP/R with a BASIC program (if you have not been locating this MLP/R at any location of your choice) you will find the # location of the begin of the MLP/R by PRINT HEX$ (VARPTR (A(Ø,Ø))). This location is usually 2FØ for the first MLP/R for a one dimension array and #2F1 for a 2 dimension array (when the discs are not used, as the DOS moves the Heap).

6.2.5.
User Control Statements

6.2.5.1
EDIT

EXAMPLE(s)

(i) EDIT
Moves entire BASIC program into edit Buffer for possible modification and display

(ii) EDIT 100
Moves only the BASIC program line number 100 into the edit buffer for possible modification and display.

(iii) EDIT 100 -
Moves the BASIC program line numbers 100 until the end of the BASIC program into the edit buffer for possible modification and display.

(iv) EDIT 100-130
Moves the BASIC program line numbers 100 to 130 into the edit buffer for possible modification and display.

(v) EDIT - 130
Moves the BASIC programs from the first line to line number 130 into the edit buffer for possible modification and display.

Functional Explanation

The Edit statement provides a simple means to modify or type-in aprogram into the DAI Personal Computer. A number of program lines are placed into an internal edit buffer. The first 24 BASIC program lines in the edit buffer are displayed on the screen. The cursor is positioned at the first character of the first line on the display.

The cursor can be moved around the screen by use of the cursor control keys. (↑ ↓ → ←). If the operator attempts to move the cursor off the screen

the part of the document which can be seen on the screen is moved to keep the cursor visible. The visible area of the document is known as the "window". The window can also be changed by using the cursor control keys plus the "shift" key. The cursor stays in the same place in the document, unless moving the window would take it off the screen.
The CHAR DEL key deletes the character at the cursor. It has no effect to the right of a carriage return. Any other character typed in is inserted before the cursor position, if the cursor is left of the carriage return on the line.
When all editing is finished, the BREAK key should be pressed. If it is followed by a second BREAK, then the whole effect of the editing is ignored. If followed by a space, then the original version of the edited text is deleted, just as if it were typed in from the keyboard.
Any necessary error messages will be put on the screen, andfollowed by a prompt. The Edit command is also used to achieve Program merges from different cassettes.

Special note:
Avoid pressing BREAK or any other key after typing the end of the EDIT command and before the program has been displayed on the screen.
See "Edit Buffer Program" in appendix.

6.2.5.2
IMP

EXAMPLES
See examples given in paragraph 6.2.2

6.2.5.3
LIST

EXAMPLE(S)
(i) LIST
Displays the entire BASIC program. During display the output can be made to pause by pressing any character key. Then pressing of the space bar will continue the listing display output.

(ii) LIST 100
Displays BASIC program line number 100 only.
(iii) LIST 100 -
Displays BASIC program starting at line number 100 until the end of the program.
(iv) LIST 100-130
Displays BASIC program line numbers 100 to 130.
(v) LIST - 100
Displays BASIC program starting at first line of program and until line number 130.

6.2.5.4
NEW

EXAMPLE(S)
(i) NEW
Deletes current BASIC program that is stored in memory and resets all variables to the undefined state. The HEAP reservation is is not changed. (See 6.2.11).

6.2.5.5
RUN

EXAMPLE(S)
(i) RUN
Starts execution of the BASIC program currently in memory at the lowest line number.
(ii) RUN 100
Starts execution of ten BASIC program currently in memory at line number 100. If line 100 does not exist, an error message occurs.

6.2.6

Program control Statements

6.2.6.1

END

EXAMPLE(S)

(i) END

Terminates the execution of a BASIC program. The program cannot be further continued without a RUN command. An "END PROGRAM" message is displayed.

6.2.6.2

FOR......NEXT

EXAMPLE(S)

(i) FOR V = 1 TO 9.3 STEP .6

(ii) FOR V = 1 TO 9.3

(iii) FOR V = 10* N TO 3.4/Q STEP SQR(R)

(iv) FOR V = 9 TO 1 STEP - 1

(v) FOR W = 1 TO 10 : FOR W = 0 TO 3 : NEXT : NEXT.

The variable in the FOR statement is set to the first expression given. Statements are executed until a NEXT statement is encountered. Action at this point depends on the rest of the FOR statement. When the FOR statement is executed the "TO" and "STEP" expressions are also calculated. The step defaults to 1 if it is not explicitly given. Then the range is divided by the step to calculate a repeat count for the loop. This must be within the ranges 0 to 2↑23-1 for a floating point loop and 0 to 2↑31-1 for an integer one. The loop is run this number of times irrespective of anything else, and is always run at least once.

If the STEP is not explicitely given then the NEXT statement uses a special fast routine to increment the variable value. If it is explicitely given it is added to the variable. Loops using integer variables run faster than those using floating point ones.

Special cases:

a) The interpreter will terminate an unfinished loop if a NEXT statement for an outer one is encountered. E.g.

FOR A = 1 TO 10 : FOR B = 0 TO 3 : NEXT A

is allowable.

b) The interpreter will terminate all loops up to the correct level if a loop is restarted. E.g.

```
10 FOR A = 1 TO 10
20 FOR B = 0 TO 3
30 GOTO 10
```

is allowable.

c) FOR loops inside a subroutine are separate from those outside for purpose of special cases (a) and (b)

d) A FOR loop may be abandoned by a RETURN statement. E.g.

```
10 GOSUB 10
20 STOP
30 FOR A = 1 TO 10
40 RETURN
```

is allowable.

e) after a FOR loop finishes, the variable has the value it would next have taken.

E.g. 10 FOR I = 0 TO 10 : NEXT

20 PRINT I

Will print 11.Ø.

6.2.6.3

GOSUB

EXAMPLE

(i) GOSUB 910

Branches to the specified statement, i.e. (910). When a Return statement is encountered the next statement executed is the statement following the GOSUB. GOSUB nesting is limited only by the available stack memory. A program can have 10 levels of GOSUB or 15 levels of FOR loops without difficulty.

6.2.6.4
GOTO

EXAMPLE
GOTO 100
Branches to the statement specified.

6.2.6.5
IF.....GOTO

EXAMPLES
(i) IF X = Y + 23.4 GOTO 92
Equivalent to IF ... THEN, except that IF ... GOTO must be followed by a line number, while IF ... THEN is followed by another statement, or a line number.
(ii) IF X = 5 GOTO 50:Z = A
Warning: Z = A will never be executed.
6.2.6.6
IF ... THEN

EXAMPLE
(i) IF X < 0 THEN PRINT "X LESS THAN 0" : GOTO 350
In this example, if X is less than 0, the PRINT statement will be executed and then the GOTO statement will branch to line 350. If the X was 0 or positive, BASIC will proceed to execute the lines after this one.
(ii) IF X = Y + 23.4 THEN 92
IF ... THEN statement in this form is exactly equivalent to IF ... GOTO example (1).

6.2.6.7
ON ... GOSUB

EXAMPLE(S)
(i) ON I GOSUB 50, 60

Identical to "ON ... GOTO", except that a subroutine call (GOSUB) is executed instead of a GOTO. RETURN from the GOSUB branches to the statement after the ON ... GOSUB.

6.2.6.8
ON ... GOTO

(i) ON I GOTO 10, 20, 30, 40
Branches to the line indicated by the I'th number after the GOTO.
That is: IF I=1 THEN GOTO LINE 10
IF I=2 THEN GOTO LINE 20
IF I=3 THEN GOTO LINE 30
IF I=4 THEN GOTO LINE 40
If I is <= Ø or > (number of line numbers) then the following statement is executed.
If I attempts to select a non-existent line, an error message will result. As many line numbers as will fit on a line can follow an ON ... GOTO.
(ii) ON SGN(X)+2 GOTO 40, 50, 60.
This statement will branch to line 40 if the expression X is less than zero, to line 50 if it equals zero, and to line 60 if it is greater than zero.

6.2.6.9
RETURN

EXAMPLE(S)
(i) RETURN
Causes a subroutine to return to the statement that follows the most recently executed GOSUB.

6.2.6.10

STOP

EXAMPLE(S)

(i) 100 STOP

BASIC suspends execution of programs and enters the command mode. "STOPPED IN LINE 100" is displayed. To continue program with next sequential statement type in "CONT".

6.2.6.11

WAIT

EXAMPLE(S)

(i) WAIT I, J, K

This statement reads the status of REAL WORLD INPUT port I, exclusive OR's K with the status, and then AND's the result with J until a result equal to J is obtained. Execution of the program continues at the statement following the WAIT statement. If the WAIT statement only has two arguments, K is assumed to be zero. If waiting for a bit to become zero, there should be a one in the corresponding position for K. I, J and K must be > = 0 and < = 255.

(ii) WAIT MEM I, J, K

WAIT MEM I, J

As example (i), but I is a memory location, which of course may be a memory-mapped I/O port.

(iii) WAIT TIME I

Delays program execution for a time given by the expression I. The result should be in the range 0 to 65535.

Time is measured in units of 20 milliseconds.

6.2.7

Physical Machine Access Statements

6.2.7.1

CALLM

EXAMPLES

(i) CALLM 1234

Calls a machine language routine located at the memory locations specified.

(ii) CALLM I, V

Calls a machine language routine located at the memory locations specified by I. Upon entry to the machine language program the register pair H, L contains the address of the variable specified by V. The machine language subroutine must preserve all of the 8080 registers and flags and restore them on return.

If V is a variable, the pointer is to V. If V is a string, the pointer is to a pointer to the string. The string consists of a length byte followed by characters. If V is a matrix, pointer is as though V is a normal variable.

6.2.7.2

INP (I)

EXAMPLE

A = INP (31)

Reads the byte present in the DCE-BUS CARD 3 PORT 1 and assigns it to a variable A. The port-number should be = 0 and = 255.

6.2.7.3

OUT I, J

EXAMPLE

OUT 91, A

Sends the number in variable A to the DCE-BUS card 9 PORT 1. Both I and J must be = 0 and = 255.

6.2.7.4

PDL (I)

EXAMPLE

A = PDL (I)

Sets the variable A to a number between 0 and 255 which represents the position of one of the paddle potentiometers. I must be or = Ø and or = 5.

6.2.7.5

PEEK (I)

EXAMPLES

(i) A = PEEK (#13C2)

The contents of memory address Hex 13C2 will be assigned to the variable A. If I is 65536 or 0 an error will be flagged. An attempt to read a memory location non-existent in a particular configuration will return an unpredictable value.

Displays the value in the decimal memory address 258.

6.2.7.6

POKE

EXAMPLE(S)

(i) POKE I, J

The POKE statement stores the byte specified by its second argument (J) into the memory location given by its first argument (I). The byte to be stored must be > = 0 and < = 255, or an error will occur. If address I is not > = 0 and < 64K, an error results. Careless use of the POKE statement will probably cause BASIC to stop, that is, the machine will hang, and any program already typed in will be lost. A POKE to a non-existing memory location is usually harmless.

Example of POKEs (see also the ASSEMBLY section of the book)

POKE #131,Ø	OUTPUT TO SCREEN AND RS 232
#131,1	OUTPUT TO SCREEN ONLY
#131,2	OUTPUT TO EDIT BUFFER
#135,2	READ (INPUT) FROM EDIT BUFFER
#13D,#10	SELECT CASSETTE 1,#20 FOR CASSETTE2
#40,#28	CASSETTE MOTOR CONTROL 1 ON
#40,#28	CASSETTE MOTOR CONTROL 2 ON
#40,#30	CASSETTE MOTOR CONTROL 1 AND 2 OFF
#730,#30	FLOPPY DRIVE Ø ACTIVATED
#730,#31	FLOPPY DRIVE 1 ACTIVATED

See also useful POKES in paragraph (5.9.1 + 2 + 3)

6.2.7.7

UT

EXAMPLE

UT

Calls the Machine Language Monitor.

6.2.8

BASIC System Data & I/O Statements

6.2.8.1

DATA

EXAMPLES

(i) DATA 1, 3, -1E3, -0.4.

Specifies data, read from left to right. Information appears in data statements in the same order as it will be read in by the program.

(ii) DATA "F00", "Z00"

Strings may be read from DATA statements. If the string contains leading spaces (blanks), or commas (,), it must be enclosed in double quotes.

6.2.8.2
GETC

EXAMPLE(S)

(i) A = GETC

The ASCII value of the last character typed on the keyboard. If no character has been typed in since the last GETC statement zero value is returned. Note that GETC forces a scan of the keyboard. Scanning the keyboard too often will cause "key bounce" and keys may appear to be pressed twice when they were only pressed once.

6.2.8.3
INPUT

EXAMPLE(S)

(i) INPUT V, W, W2

Requests data from the terminal (to be typed in). Each value must be separated from the previous value by a comma (,). The last value typed should be followed by a carriage return. A "?" is typed as a prompt character. Only constants may be typed in as a response to an INPUT statement, such as 4.5E-3 or "CAT". If more data was requested in an INPUT statement than was typed in, another "?" is printed and the rest of the data should be typed in.

If more data was typed in than was requested, the extra data will be ignored. The program will print a warning when this happens. Strings must be input in the same format as they are specified in DATA statements.

(ii) INPUT "VALUE";V

Optionally types a prompt string ("VALUE") before requesting data from the terminal.

Typing CONT after an INPUT command has been interrupted due to the BREAK key will cause execution to resume at the INPUT statement. If any error occurs, the INPUT statement will restart completely.

6.2.8.4
PRINT (can be replaced by "?")

EXAMPLES

(i) PRINT X, Y, Z
(ii) PRINT
(iii) PRINT X, Y
(iv) PRINT "VALUE IS", A
(v) ? A2,B

Prints the numeric or string expressions on the terminal. If the list of values to be printed out does not end with a comma, (,) or a semicolon (;), then a new a new line is output after all the values have been printed. If a semicolon separates two expressions in the list, their values are printed next to each other. If a comma appears after an expression in the list, the cursor is positioned at the beginning of the next column field. If there is no list of expressions to be printed, as in example (ii), then the cursor goes to a new line.
There are 5 fields on the line in positions Ø, 12, 24, 36, 48.

6.2.8.5
READ

EXAMPLE

READ V,W

Reads data into a specified variables from a DATA statement. The first piece of data read will be the first not read by any previous data statement. A RUN or RESTORE statement restarts the process from the first item of data in the lowest numbered DATA statement in the program. The next item of data to be read will be the first item in the second DATA statement of the program. Attempting to read more data than there is in all the DATA statements in a program will cause an error message.

6.2.8.6
RESTORE

EXAMPLE

(i) RESTORE

Allows the re-reading of DATA statements. After a RESTORE, the next item of data read will be the first item listed in the first DATA statement of the program, and so on as in a normal READ operation.

6.2.9
Cassette and Disc I/O Statements

Additional Cassette and Disc commands are available using the Resident Machine Utility Program (See Section 6.3).

6.2.9.1
CHECK

The CHECK command scans a cassette tape or disc and examines all the files. The type and name of each is printed followed by the word "OK" or "BAD" depending upon the file checksumming correctly. For cassettes the command does not stop of its own accord, but will stop if the BREAK key is held down.

6.2.9.2
LOAD

EXAMPLES

(i) LOAD "FRED"

Loads the program named "FRED" from the cassette tape or disc. When done, the LOAD will type a prompt as usual. The file name may be any string of printable characters.

(ii) LOAD

Loads the first program that is encountered on the tape. If the recorder motor is under automatic control it will be started. Otherwise the recorder should be started manually.
If a LOAD command is executed directly, not as part of a program, then as each data block or file is passed on the tape, its type (0 for a BASIC program) and its name will be printed. When the load is finished succesfully, a prompt is printed. If the LOAD is unsuccessful, then a message "LOADING ERROR" is printed. It is followed by a number giving details of the problem. The flashing of the cursor will cease while the data is being read from the tape.

6.2.9.3
LOADA

Loads ARRAY or Machine Language programs stored as arrays.
Example LOADA A$ "FRED" or LOADA F$ + "J"
FRED or J are the array names.

```
10 DIM A$ (Ø,Ø)          100 DIM A$ (Ø,Ø)
20 INPUT A$              110 LOADA A$
30 SAVEA A$ "INFO"       120 GOTO 100
40 GOTO 10
```

6.2.9.4
SAVE

EXAMPLE

(i) SAVE "GEORGE"

(ii) SAVE A$

Saves on cassette tape or disc the current program in the memory. The program in memory is left unchanged. More than one program may be stored on one cassette/disc using this command. The program is written on the cassette under the name given.

(iii) SAVE

The program is written on the cassette under a null name.

The system replies to the command with the message "SET RECORD, START TAPE, TYPE SPACE". Place the tape recorder into the right state for recording (note that if the motor control is connected to the Personal Computer, the motor will not yet start). Then press the space key. When the motor will stop (if automatically controlled) a prompt character will appear on the screen. If the cassette is working manually, then it should now be stopped.

6.2.9.5

SAVEA

EXAMPLE

(i) SAVEAG "GEORGES"

(ii) SAVEA A$

Saves an array on cassette or disk.

(iii) SAVEA A

EXAMPLE

```
20 INPUT A$
30 SAVE A$
40 GOTO 10
```

After typing RUN and pressing RETURN key the tape recorder will start automatically to record the input you enter in line 20 (the tape recorder must have a remote control and must be in recording mode).

COPY OF A PROGRAM FOLLOWED BY AN ARRAY (OR MACHINE LANGUAGE ROUTINE) WITH 2 TAPE RECORDERS (1 BEING ON PLAY, 2 ON RECORD).

POKE #40, #28 : LOAD : POKE #40, #18 : SAVE : POKE #40, #28 : PRINT "SAVE ENDED" : CLEAR 2000 : DIM A (20, 20) : LOAD A :

POKE 40, 18

SAVEA A POKE 40, 28

PRESS RETURN: the array is named A.

6.2.10

Program Debug and Comment Statements

6.2.10.1

CONT

EXAMPLE

CONT

Continues BASIC program execution with the next statement following the "STOP" Statement or "BREAK" position.

6.2.10.2

REM

EXAMPLES

(i) REM NOW SET V=0

Allows comments inside BASIC programs. REM statements are not executed, but they can be branched to. A REM statement is terminated by end of line, but not by a (:) character.

(ii) REM SET V=0:V=0

The V=0 statement will not be executed.

(iii) The V=0 statement will be executed.

6.2.10.3

STEP

Command to allow single step execution of BASIC programs. After "BREAK" or "STOP" the operator types in STEP and then each depression of the space bar allows execution of the next sequential BASIC line. The line to be executed is displayed before execution of that line.

6.2.10.4
TRON

EXAMPLE

(i) 100 A = 0
105 TRON
106 A = 1
107 A = 2
108 TROFF

When you RUN, and after the TRON (TRACE ON) is executed the lines 106 and 107 will be executed and displayed at the same time until the TROFF (TRACE OFF) is reached and executed.

6.2.10.5
TROFF

EXAMPLE SEE 6.2.10.4

6.2.11
Array and Variable Statements

6.2.11.1
CLEAR

EXAMPLE

(i) CLEAR 999

Resets all variables to ∅ or the null string, and returns all space assigned to arrays. The size of the HEAP (array and string storage) is than set to the number specified by the CLEAR statement. The minimum size is 4 (no space would be available) and the maximum is 32767

6.2.11.2
DIM

EXAMPLE

(i) DIM A(3), B(10)
(ii) DIM R3(5,5), D$(2,2,2)

Allocates space for arrays. Arrays can have more than one dimension. All subscripts start at zero (0), which means that DIM X (100) really allocates 101 matrix elements. The maximum size for a dimension is 254 Dimensions may be specified as variables or expressions.
DIM statements may be re-executed to vary the size of an array. The space used for arrays is in the same part of RAM as that for strings, the size of which is set by the CLEAR command.

6.2.11.3
FRE

EXAMPLE(S)

(i) A = FRE

The variable A is set to the number of memory bytes currently unused by the BASIC program. Memory allocated for string and arrays is not included in this count.

(ii) PRINT FRE

The amount of remaining memory space will be displayed.

6.2.11.4
LET

EXAMPLE(S)

(i) LET W = X
(ii) V = 5.1

Assigns a value to a variable. The word "LET" is optional.

6.2.11.5
VARPTR (V)

EXAMPLE(S)

(i) A = VARPTR (B)

Variable named (A) is set to the memory address of the variable named (B).

(ii) A = VARPTR (B(3,4))

Variable named (A) is set to the memory address of the array element B(3,4).

6.2.12

GRAPHICS AND DISPLAY STATEMENTS (See Example program "TOWER OF HANOI")

6.2.12.1

MODE

EXAMPLE(S)

(i) MODE 0

Places display in character only mode.

(ii) MODE 1A

Places display in split mode. Low resolution graphics with 16 colours and a four line character display at the bottom.

The Personal Computer has 3 different graphic definitions available for the graphics display and at each definition there are 4 possible configurations of the screen. Two of these have only graphics on the screen, and the others are exactly the same except that the graphics area is moved up the screen to make room for four lines of characters. The graphics hardware has 2 different ways in which it can be used. That is why at each definition there are 2 different types of display. The display types are known as 16-colour, and 4-colour modes. In the 16 colour modes each point on the screen can be set to any of the 16 colours. However each field of 8 dots horizontally (positions 0 to 7, 8 to 15 etc.) can only have 2 or sometimes 3 separate colours in it. For exact details of the restrictions on what can be drawn. (See 3.2.2.1) At any time the 4 selected colours can be altered, and the existing picture changes colour immediately. This allows interesting effects. (see for instance "ANIMATE").

MODE DEFINITION TABLE

Number	Graphics size	Text size	Type of graphics
0	-	24 X 60 CHAR	-
1	72,65	-	16 colour
1A	72,65	4 X 60	16 colour
2	72,65	-	4 colour
2A	72,65	4 X 60	4 colour
3	160,130	-	16 colour
3A	160,130	4 X 60	16 colour
4	160,130	-	4 colour
4A	160,130	4 X 60	4 colour
5	336,256	-	16 colour
5A	336,256	4 X 60	16 colour
6	336,256	-	4 colour
6A	336,256	4 X 60	4 colour

6.2.12.2
COLORG

EXAMPLE
COLORG 1 2 3 4
Sets the colours available in any four colour graphics mode to 1,2,3 and 4.
If the screen is already in a 4 colour mode, then the colour change will be immediate. Any area which was in the first-named colour of the previous COLORG statement, is now displayed in colour 1, and so on.
If the screen is in a 16 colour mode, no immediate effect is visible.
In any event, the next time a new graphics mode is entered, the initial colour of the graphics area will be the first colour given in the COLORG command. This applies both for 4 and 16 colour modes.
If COLORG has not been used, then after a 4 colour mode command (i.e. mode 2) the colours available will be Ø, 5, 1Ø, 15.

6.2.12.3
COLORT

EXAMPLE
COLORT 8 15 0 0
Sets up colour number 8 as the background colour for the text screen and colour 15 as the colour of the characters. The other two colour numbers are not normally used. However they define an alternative set of colours which can be used by POKE access, or machine code routines.

6.2.12.4
Drawing Facilities

Points on the graphic screen are specified by an X,Y co-ordinate with 0, 0 located at the bottom left corner of the display screen. An attempt to draw out of the maximum area for a particular graphics mode will result in an error.

It is possible, however, to draw in the invisible top section of the graphics area in split screen modes. The drawing facilities provide statements to draw dots, lines and rectangles on the graphic display screen. The DOT statement places a single dot of a specified colour at any allowable X, Y coordinate on the display statement allow the drawing of a line and the colouring of a rectangular area specified by two X, Y coordinates. See color codes paragraph 3.2.12.

6.2.12.4.1
DOT

EXAMPLE(S)
(i) DOT 10, 20 15
Places a dot of colour 15 at the position X = 10 and Y = 20. The size of the dot will depend upon which graphic resolution was selected.

6.2.12.4.2
DRAW

EXAMPLE
DRAW 91,73 42,77 15
Draws a line in colour 15 between 91, 73 and 42, 77. There is no restriction on the order of the coordinates. Line width will depend upon which resolution was selected.

6.2.12.4.3
FILL

EXAMPLE
FILL 91,73 42,77 15
Fills the rectangle with opposite corners at 91, 73 and 42, 77 with the colour 15. There is no restriction on the order of the points. The physical size of the rectangle depends upon the resolution selected.

6.2.12.5

Animated Drawing Facility.

With the screen in a 4 colour mode each point is described by 2 bits. The binary value of these 2 bits selects which of the four available colours should be displayed. Normally a DOT, DRAW or FILL sets both of these bits to their new value. However, a facility is available to set or clear only one of the two. This is accomplished by specifying colour numbers 16, 17, 18 or 19. It is emphasized that these are not real colours, but an extra facility.

For example:

MODE 2A
COLORG 6 9 12 15

These commands set all points on the screen to colour 6. The two bits for each point on the screen are both Ø. (Binary Ø Ø).

DOT 10, 10 17

This sets the lower bit only for point 10, 10. Thus the point changes to colour 9 (Binary 0 1).

DOT 10,10 19

This sets up the upper bit only. The point changes to colour 15 (binary 11 = 3)

DOT 10,10 16

This clears the lower bit, and gives colour 12 (binary 10 = 2).

DOT 10, 10 18

This clears the upper bit, and gives colour 6 (binary 00). The usefulness of this system is that by the COLORT command two pictures can be independently maintained and altered on the screen. This allows one pattern to be changed invisibly while the other is displayed. The pictures can be swapped instantaneously and the invisible one changed.

Example program:

```
5   MODE 2
10  COLORG Ø Ø Ø Ø
20  FOR Q = 1 TO XMAX
30  DRAW Ø,Ø Q,YMAX 17+2* A:REM COLOR = 17 OR 19.
40  COLORG Ø 15 - 15* A  15* A  15:REM COLOR = 18 OR
                                                16.
```



```
50 DRAW Ø,Ø Q - 1, YMAX 18-2 A : A = 1 - A : NEXT
```

"ANIMATE"

When the screen is in a 4 colour mode, each point on the screen is described by 2 bits. A facility is provided for drawing using only one bit from each pair, without affecting the other.

Drawing using the number	has effect of
17	set lower bit
19	set upper bit
16	clear lower bit
18	clear upper bit

This allows two totally independent pictures to be maintained and separately updated. They simply appear to overlap. If the SCOLG entrypoint is used to make only 1 visible at a time, then animation effects can be achieved.

If the colours set by the SCOLG command are numbered 0,1,2,3 in order as given, then the colour seen on the screen is selected by the two bits for each point in the natural way.

E.g.

If SCOLG sets up red, yellow, green and blue, in that order

Upper Bit	Lower Bit	Visible Colour
0	0	Red
0	1	Yellow
1	0	Green
1	1	Blue

"Colours 20 to 23"

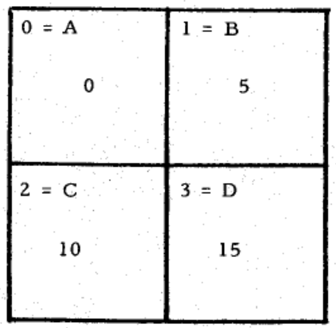
In 4 colour mode only, the colour numbers 20 to 23 may be used to request the 4 colours set up by the last SCOLG call. Colour 20 always refers to the first colour given irrespective of what it is. Similarly 21 is the second colour, and so on.

The "animate" facility using colours 16 to 19 can be explained as a 4 boxes square where a colour is assigned to a box.

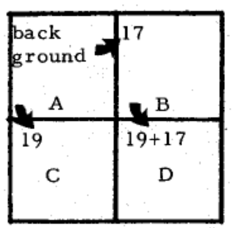
Number 0 1 2 3 of the

COLORG A B C D command assigning a color to each box.

A DOT, DRAW or FILL Command with a 16 to 19 colour definition will move the background and foreground colours as indicated by the arrows.



16 ←
17 →
18 ↑
19 ↓



COLORG 0 0 15 15

COLORG 0 15 0 15

6.2.12.6

XMAX

EXAMPLE

A = XMAX

Sets the variable A to the maximum allowable X value for the current graphics mode.

6.2.12.7

YMAX

EXAMPLE

A = YMAX

Sets the variable A to the maximum allowable Y value for the current graphics mode.

6.2.12.8

SCRN (X,Y)

EXAMPLE

(i) A = SCRN (31,20)

Sets the variable to a number corresponding to the colour of the screen at coordinate 31,20.

6.2.12.9

CURSOR

EXAMPLE

(i) CURSOR 40,20

Moves the cursor to the fourtieth character position of the twentieth line from the bottom of the screen.

The cursor can be moved to any position on the screen by using the CURSOR command. The positions are given by X, Y coordinates where the bottom left corner of the screen is 0,0.

6. 2. 12. 10

CURX

EXAMPLE

A = CURX

Sets the variable A to the X position of the cursor (character position). Value returned will be < = 60.

6. 2. 12. 11

CURY

EXAMPLE

A = CURY

Sets the variable A to the Y position of the cursor (line position). Value returned will be < = 24.

6. 2. 13

Graphical Sound Statement.

6. 2. 13. 1

Programmable Sound Facility

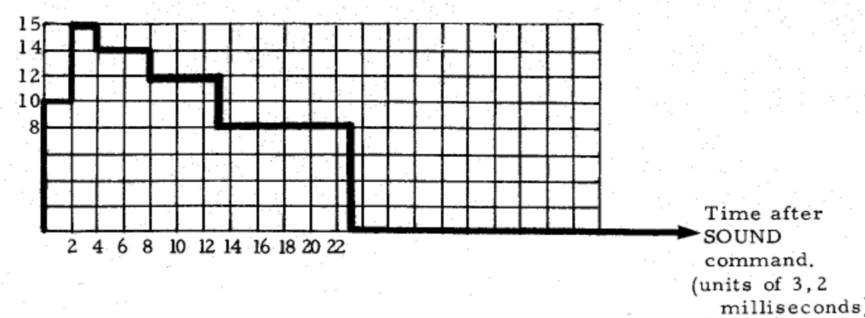
The Graphical Sound Generator of the DAI Personal Computer is supported by the BASIC to give a set of commands that allow program control of the sound system, 3 oscillator channels plus a white noise channel. The SOUND command is the primary method of control. The SOUND command specifies a channel to which it applies, an envelope to be used, the required volume and requency. A simple sound command would be:

SOUND 0 1 15 0 FREQ (1000)

This would set channel 0, using envelope number 1, at a volume of 15 and frequency 1000 Hz. The ENVELOPE statement allows the volume of a note to be rapidly changed, in the same way as that of a musical instrument. Thus the rise and fall in volume for a note can be specified. The command specifies a set of pairs of volume and time. The volume constants are in the range 0 to 15 and the time is in units of 3. 2 milli-seconds. For example the command:

ENVELOPE 0 10,2;15,2;14,4;12,5;8,10;0

This sets a volume envelope like this:



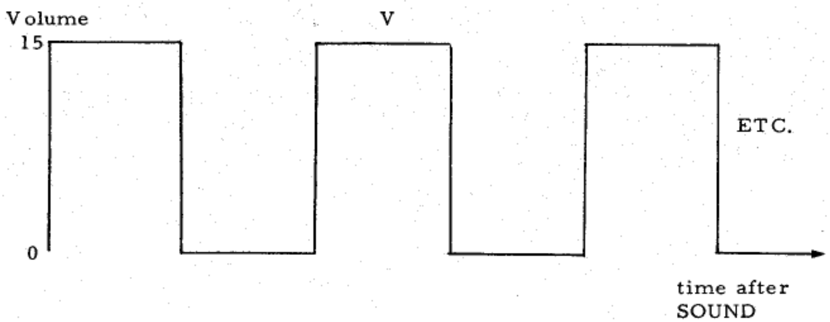
So every time a SOUND command is given it produces a short burst of sound whose volume is as shown above. Varying the envelope varies the quality of the sound heard.

The volume given in a SOUND command is effectively multiplied by that in the envelope. So if the SOUND command requests a volume of 8 units, which is 8/15 of full volume, and the envelope requests 4 units, which is 1/4 of the maximum figure, then the volume used is 2/15 of the maximum. (as 1/4 x 8/15 = 8/60 = 2/15.)

The envelope command can end, as above, in a single volume, in which case that volume continues for ever, or in a pair of volume and time, in which case the envelope is repeated indefinitely. For example:

ENVELOPE 0 15,10;0,10;

Sets an envelope like this:



That would give a series of "blips" of sound.

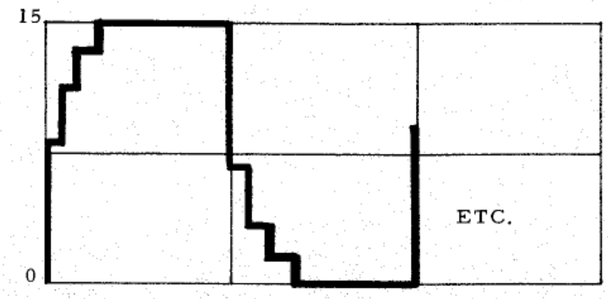
The simplest envelope is obviously:

ENVELOPE 0 15

Which then has no audible effect on SOUND commands, as all volumes are multiplied by 15/15.

Special note:

The BASIC Interpreter limits the rapidity with which the volume on any channel is allowed to change. The maximum change is d/2 + 1, where d is the difference between the requested and current volumes. Thus the actual volume output for the envelope above would be:



This helps reduce spurious sound caused by volume changes.

The noise generator is controlled by a NOISE command that controls the audible output of the white noise generator. Only its volume and envelope can be set. e.g. :

NOISE 0 15

Turns on the noise channel using envelope 0 and overall volume 15.

In addition to the facilities already described, the SOUND command controls 2 others. They are TREMOLO and GLISSANDO.

Tremolo is simply a rapid variation of volume by ± 2 units. This gives a "warbling" effect to the sound. Glissando is an effect where the new note on a channel does not start immediately at the requested frequency, but "slides" there from the previous frequency. The effect resembles a Hawaiian Guitar or Stylophone. Glissando + Tremolo are controlled by one parameter in the SOUND command. Setting the bottom bit requests Tremolo and the next bit Glissando. E. g. :

(i) SOUND 0 0 13 1 FREQ (1000)

(ii) SOUND 0 0 15 2 FREQ (5000).

The first example sets channel 0, using envelope 0, at volume 13 and with tremolo. The volume put will vary rapidly from 11 to 15.

The second example increases the volume to 15, and slides the frequency "GLISSANDO" up to 5000 Hz. The flexibility and facilities of the Graphical Sound Generator have been illustrated fully and their capabilities exploited with the three commands previously discussed.

Due to the flexibility of change in volume and frequency it is quite feasible to explore the possibilities of vocal sound generation. The BASIC of the DAI Personal Computer gives full control to the programmer who wishes to develop experimentally a burst of sound and frequencies that result in audible words.

6.2.13.2

SYNTAX : SOUND

(i) SOUND <CHAN><ENV> <VOL> <TG> FREQ <PERIOD>
(ii) SOUND <CHAN> OFF
(iii) SOUND OFF

<CHAN> is an expression in the range 0 to 2. It selects programmable oscillator 0, 1 or 2.
<ENV> is an expression in the range 0, 1. It selects which of the 2 previously defined envelopes should be used.
<VOL> is an expression in the range 0 to 16. It selects the volume for this particular sound. It is multiplied by the volumes in the ENVELOPE specified.
<TG> is an expression in the range 0 to 3.

0 selects	no tremolo + no glissando
1 selects	tremolo + no glissando
2 selects	no tremolo + glissando
3 selects	tremolo + glissando

<PERIOD> is an expression in the range 2 to 65535. It sets the period of the required sound in units of 1/2 microseconds.

6.2.13.3

SYNTAX: ENVELOPE

(i) ENVELOPE <ENV> {<V> , <T> ;} <V> , <T> ;
(ii) ENVELOPE <ENV> {<V> , <T> ;} <V>

ENV is an expression in the range 0 to 1. It selects which of 2 envelopes is being defined.
V is an expression in the range 0 to 15. It selects a volume level by which that in a SOUND command is to be multiplied.
T is an expression in the range 1 to 254. It selects the time for which the volume V applies. It is in units of 3.2 milliseconds.
Note: The parts of the command in curly brackets are optional and may be absent or repeated as many times as required.

6.2.13.4

SYNTAX: NOISE

(i) NOISE ENV VOL
(ii) NOISE OFF

ENV is an expression in the range Ø to 1.
VOL is an expression in the range 0 to 15.
This represents a 4 bit binary number. The top 2 bits of this number (when modified by the ENVELOPE specified) control the volume of the noise. The bottom 2 bits control the frequency.

6.2.13.5

FREQ

EXAMPLE

A = FREQ (1000)

Sets the variable A to a number that can be sent to a Graphical Sound Generator channel to result in a 1000 hertz rate.

6.2.13.6

Synthesing Vocal Sound.

6.2.13.6.1

TALK

TALK ADDRESS

CODE	DATA
0	2 BYTES FREQ. CODE CHANNEL 0
2	" " 1
4	" " 2
8	1 BYTE VOLUME CHANNEL 0
9	" " 1
A	" VOLUME W. NOISE GENERATOR
C	2 BYTES DELAY IN UNITS OF MSEC
D	CALL MACHINE CODE
FF	END

DATA BLOCK

	location	content	
# 2000	20 00	09C4	set channel 0 freq. 800
	20 02	1AØA	set channel 1 freq. 300
	20 08	0F	set maximum volume ch Ø
	20 09	0F	set maximum volume ch 1
	20 0C	FEFE	set + listen to it for ---- msec
	20 08	00	turns volume down
	20 09	00	
	20 0D	0050	machine codes at 5000
	20 FF		End.


```
#5000  ØØ  [LXI H, VARPTR (Q(Ø))]  21 ØØ 20
 5004       RETURN                  C9
Ex.    3    CLEAR 1000
       4    DIM Q (100)
       5    B% = VARPTR (Q(Ø))
      1Ø    READ A%
      2Ø    POKE B%, A% : B% = B% + 1
      3Ø    IF A%<>#FF GOTO 1Ø
      4Ø    TALK VARPTR (Q(Ø))
     (5Ø    WAIT TIME 1Ø)
      6Ø    GOTO 4Ø
      8Ø    DATA Ø, 9,#C4, 2,#1A,#A, 8,# F, 9,# F
      9Ø    DATA# C, #FE,#FE, 8, Ø, 9, Ø,# FF
```

6.2.14

Arithmetic and String Functions

The following is a list of the mathematical + character handling functions provided by BASIC. Each takes a number of expressions (arguments) in brackets and works on them to return a result. This result may be used in just the same way as a variable or constant in expressions.

EXAMPLES

(i) A = 3.0 + 2.1

(ii) A = SIN (3.0) + 2.1

6.2.14.1
ABS(X)

Gives the floating point absolute value of the expression X. ABS returns X if X > = 0, -X otherwise. For example ABS(-253.7) = 253.7.

6.2.14.2
ACOS(X)

Returns arc cosine of X. Result is between -PI/2 and PI/2.

6.2.14.3
ALOG(X)

Returns antilog base 10 of X.

6.2.14.4
ASC(X$)

Returns the integer ASCII value of the first character of the string X$.
E.g.: ASC("ABC") returns 65 since A has code 41 Hex or 65 decimal.

6.2.14.5
ASIN(X)

Returns the arcsine of X in radians. Result is between -PI/2 and +PI/2.
X may be any value between + 1 and - 1 inclusive.

6.2.14.6
ATN(X)

Returns the arctangent of X in radians.

6.2.14.7
CHR$(I)

Inverse of ASC. Returns a 1 character string whose ASCII value is I.
I must be between 0 and 255.
E.g.: CHR$ (65) returns the character "A".

6.2.14.8
COS(X)

Gives the cosine of the expression X, measured in radians. (X) may be any value between 0 and 2π inclusive.

6.2.14.9
EXP(X)

Returns the value "e" (2.71828) to the power X, (e ↑ X). "e" is the base for natural logarithms. The maximum argument that can be passed to EXP without overflow occurring depends on whether the software or hardware maths option is being used. For hardware
$-32 < X < 32$ exactly.
For software $-43 < X < 43$ approximately.

6.2.14.10
FRAC(X)

Returns the floating point fractional part of the argument.
e.g.: FRAC (2.7) = 0.7, FRAC (-1.2) = -0.2

6.2.14.11
HEX$ (I)

EXAMPLE(S)
Returns a string of characters representing the hexadecimal value of the number I. I must be between 0 and 65535.

6.2.14.12
INT(X)

Returns the largest integral floating point value less than or equal to its argument X. For example:
INT(.23) = 0, INT(7) = 7.0, INT(-2.7) = -3.0, INT(1.1) = 1.0
INT (43.999) = 43,0
Note: INT(-1) = -2.0.

6.2.14.13
LEFT$(X$,I)

Returns a string which is the leftmost I characters of the string X$.
E.g.: LEFT$("DOGFISH",3) equals "DOG"

6.2.14.14
LEN(X$)

Returns an integer giving the length in characters of the string X$.
E.g.: LEN("HELLO") equals 5.

6.2.14.15
LOG(X)

Calculates the natural logarithm (base e) of the argument (X).

6.2.14.16
LOGT(X)

Calculates the logarithm base 10 of X.

6.2.14.17
MID$(X$,I,J)

Returns (J) characters starting at position I in the string (X$). The first character is position 0.
E.g. : MID$ ("SCOWL", 1,3) returns "COW".

6.2.14.18
PI

Returns the floating point value 3.14159

6.2.14.19
RIGHT$(X$,I)

Returns the rightmost (I) characters of string (X$).
E.g. : RIGHT$("SCOWL", 3) returns "OWL".

6.2.14.20
RND(X)

Generates a hardware or software generated random number.
E.g.
If X < 0 Starts a new sequence of software numbers with X as seed. The same negative X produces the same sequence of numbers. The number returned is between 0 and X
If X > 0 Returns the next pseudo-random number from the current sequence. The number is in the range 0 to X
If X = 0 Returns a hardware generated random number in the range 0 to 1.
Ex.
```
5       CLEAR 1000
10      DIM B% (100)
20      INPUT C%
30      FOR A% = 1 TO 20
```

```
40      B% (A%) = RND (C%)
50      PRINT B% ( A%)
60      NEXT A%
```

6.2.14.21
SGN(X)

Returns 1.0 if X > 0, 0 if X = 0, and -1.0 if X < 0.

6.2.14.22
SIN(X)

Calculates the sine of the variable X. X is in radians.
Note: 1 Radian = 180/PI degrees = 57.2958 degrees; so that the sine of X degrees = SIN(X/57.2958).

6.2.14.23
SPC(I)

Returns a string of the number of spaces given by I. I ≤ 255.

6.2.14.24
SQR(X)

Gives the square root of the argument X. An error will occur if X is less than zero.

6.2.14.25
STR$(X)

Returns a string which is the ASCII representation of the number X.
E.g.: STR$ (9.2) returns the string "9.2".

6.2.14.26
TAB(I)

Returns a string of the number of spaces necessary to move the screen cursor right to the column given by I. The cursor can only be moved to the right.

6.2.14.27
TAN(X)

Gives the tangent of the expression X, X must be expressed in radians.

6.2.14.28
VAL(X$)

Returns the floating point value of the number represented by the string variable X$.
E.g. : VAL ("9.2") returns 9.2
X$ must represent a valid floating point number.

6.2.15

Arithmetic and Logical Operators

Operator	Usage		Type of Result
+ (addition)	int + int		int
	fpt + int	(Note 1)	fpt
	int + fpt		
	fpt + fpt		
	str + str		str
-/* (subtract, divide, multiply)	as +, except no string version		
↑ (power (^ on keyb.)	as		always fpt
IAND	int ... int		integer (Note 2)
IOR	int ... fpt		
IXOR	fpt ... int		
MOD	int ... int		
SHL			
SHR			
INOT	int		integer
= equal	str ... str		logical
greater than	fpt ... fpt		
smaller than	fpt ... int	(Note 1)	
different from	int ... fpt		
= greater than or equal to	int ... int		
= smaller than or equal to			
AND OR	logical		
	... logical		logical

Note 1: The integer values are converted to fpt before use.
Note 2: The fpt values are truncated to integer before use.

EXAMPLE(S)

(Numbers without decimal parts represent integers)

(i)

Operation	Result	Type of Result	
1 + 2	3	integer	
1.0 + 2.0	3.0	fpt	
1.0 + 2	3.0	fpt	
3 * 4	12	integer	
3 ↑ 4	81.0	fpt	NB
12.0/4.0	3.0	fpt	
12.0/4	3.0	fpt	
12/4	3	integer	
11/4	2	integer	NB
3 IAND 2	2	integer	
3.0 IAND 6.0	2	integer	
3.14 IAND 6.72	2	integer	
3 SHL 2	12	integer	
3.2 SHL2.1	12	integer	
7 = 4	FALSE	logical	
3.0 > 2.1	TRUE	logical	
"FRED" < "FREDA"	TRUE	logical	
"A" = "A"	TRUE	logical	
7.1 = 7	FALSE	logical	
7.0 = 7	TRUE	logical	NB
3 < 4 OR 7 = 8	TRUE	logical	
3 = 7 AND 9 < 10	FALSE	logical	

(i) (In all of the cases below, leading zeroes on binary numbers are not shown).

63 IAND 16 = 16	Since 63 equals binary 111111 and 16 equals binary 1000 , the result of the IAND is binary 1000 or 16.
15 IAND 14 = 14	15 equals binary 1111 and 14 equals binary 1110, so 15 IAND 14 equals binary 1110 or 14.
-1 IAND 8 = 8	-1 equals binary 11 .... 11 and 8 equals binary 1000, so the result is binary 1000 or 8 decimal.
4 IAND 2 = 0	4 equals binary 100 and 2 equals binary 10, so the result is binary 0 because none of the bits in either argument match to give a 1 bit in the result.
4 IOR 2 = 6	Binary 100 IOR'd with binary 10 equals binary 110 or 6 decimal.
10 IOR 10 = 10	Binary 1010 IOR'd with binary 1010 equals binary 1010, or 10 decimal.
- 1 IOR -2 = -1	Binary 11.... 11 (-1) OR'd with binary 11.... 10 (-2) equals binary 11.... 11 or -1.

The following truth table shows the logical operations on bits:

Operator	Arg. 1	Arg. 2	Result
IAND	1	1	1
	0	1	0
	1	0	0
	0	0	0
IOR	1	1	1
	1	0	1
	0	1	1
	0	0	0
INOT	1	-	0
	0	-	1

A typical use of the bitwise operators is to test bits set in the REAL WORLD input ports which reflect the state of some REAL WORLD device.

Bit position 7 is the most significant bit of a byte, while position 0 is the least significant.

For instance, suppose bit 1 of REAL WORLD port 5 is 0 when the door to Room X is closed, and 1 if the door is open. The following program will print "Intruder Alert" if the door is opened:

10 IF (INP(5)IAND 2) = 2 THEN 10

This alert will execute over and over until bit 1 (masked or selected by the 2) becomes a 1. When that happens, we go to line 20.

20 PRINT "INTRUDER ALERT"

Line 20 will output "INTRUDER ALERT".

However, we can replace statement 10 with a "WAIT" statement, which has exactly the same effect.

10 WAIT 5,2

This line delays the execution of the next statement in the program until bit 1 of REAL WORLD port 5 becomes 1. The WAIT is much faster than the equivalent IF statement and also takes less bytes of program storage.

7.0

Machine Language Utility

7.1

Introduction

The Utility provides a set of facilities to develop and debug programs in machine-code. It has the ability to keep a safe copy of the registers for a program being debugged. These can be displayed and modified, as can the mode of operation of the Real World Bus, and the Timer and Interrupt controller. The memory contents can also be displayed and changed, and can be stored on, or loaded from, disc or cassette. A machine code program can be debugged using breakpoints, or an instruction - by - instruction tracing facility.

7.2

User Interface

When the Utility is entered from BASIC by means of the UT command it prints its sign-on message: P.C. UTILITY V3.3

The message is followed by the prompt character ">". Whenever the Utility prints this character, it is waiting for another command. The format of commands is always a single letter followed possibly by one or more numbers. No separator is required between the letter and the first number. Numbers are always in hexadecimal, and are terminated by a space or carriage return. The utility always uses the last hex characters typed in, two or four depending on the required range of the number. So G12345678 is equivalent to G5678, because a 4 digit hex number is required

F0000 FFFF 5566 is equivalent to:

F0000 FFFF 66 as the third number is required to have 2 digits.

Any 2 or 4 digit number can be terminated early and the Utility will use the number of digits typed. So:

G0003
G003
G03
G3

} These are all equivalent.

NT

When there is any kind of an error, the Utility prints the character "?". This is the only possible error message.

When the utility is tracing a program or printing memory contents the display can be halted by use of the BREAK key.

Some functions require the use of a terminator apart from space or carriage return. This is called an "ESCAPE", and the key used is the "cursor Left" on the far left of the keyboard.

During the description of commands, some special signs will be used. They are:

␣	for	SPACE
↵	for	CARRIAGE RETURN
←	for	ESCAPE (LEFT ARROW)

Characters typed in are underlined in the examples.

You will return to BASIC by typing "B".

7.3
Utility Commands

This section describes in detail the four classes of commands that assist the user in his program development in the utility mode. Abbreviations used in the text are defined as follows:

adr :	ADDRESS
ladr :	LOW ADDRESS
hadr :	HIGH ADDRESS
dadr :	DESTINATION ADDRESS
badr :	BASE ADDRESS of PROM Reference

The address is a string of four hexadecimal numbers. If the string is longer than four digits, the utility accepts the four rightmost digits as the address. This feature provides the advantage that if a mistake is made while entering an address, one can disregard the mistaken figures and keep entering figures until the four rightmost digits are correct. Command arguments can be separated by either space or comma.

The four classes of commands are:

Memory Commands: These commands enable the user to trace his program while it is running, or single-step it. He can also display blocks of memory bytes, and insert user's program or data.

Register Commands These commands afford the facility to examine and modify the 8080 registers, and the vector and initialization bytes. In general these commands allow the user to initialize the DCE card before transferring control to the user program.

Hexadecimal I/O Commands With these commands the user can read file, write file.

CLASS 1. MEMORY COMMANDS

7.3.1
LOOK: L adr ladr hadr

When the sequence is terminated with the "RETURN" key the command initiates transfer to the user mode. The program counter is loaded with the address specified. After each instruction execution, the contents of all the CPU registers are displayed on the console:

I = 1043 A = 02 F = 02 B = 00 C = 00 D = 00 E = 05 H = 00 L = 00
S = P = 1045

Where "I" is the address of the instruction just executed, all the instructions between the low and high address specified will be traced. To temporarily abort program execution, press and hold the "BRAK" key during the last desired trace line, until the line is completed. To continue program execution after the break, just type "L" followed by the "RETURN" key. Tracing will continue with the command whose address is equated to "P" on the last trace.

While under the control of the Utility during the break, all functions, may be used without affecting subsequent LOOK restart. The programmer is thus free to access and modify the entire register and memory area during the break.
Before restarting execution, the "trace window" can be changed from the one originally specified with this command. To alter the trace window continue program execution by typing:

L ladr hadr

followed by a return. The LOOK function restarts with the new trace limits. Whenever the LOOK function is initiated by typing all three arguments, the system is initialized as described in Section 4. 1. However, when LOOK is restarted by just typing L, or L with the new trace window arguments, only the CPU registers are restored. No other states are modified. This allows normal continution of a program after the BREAK.

The BREAK key abort feature is always active, even when the program is running outside the trace window. This feature allows escape from a program loop while saving the Program Counter.

7. 3. 2
DISPLAY: D ladr hadr

When terminating the sequence by the "RETURN" key, the console displays consecutive memory bytes in hexadecimal starting with the one specified by the low address and ending with the one specified by the high address. Each line is preceeded by the memory address of the first byte on the line.

Example: D1000, 110A
Pressing and releasing the BREAK key aborts printout.

7. 3. 3
GO: G adr

When the sequence is terminated with the "RETURN" key, the command initiates transfer to the user mode. The system is initialized, and program execution starts. The user program stored in the memory controls the CPU until control is returned to the utility. The address in the command is optional; if no address is given, only the 8080 registers are restored from the save area, and not the GIC and TICC initialization bytes. Execution starts with the saved P (program counter) value. Entering "G" without address allows restarting the system after a breakpoint without reinitializing.
Example: G1040
This command transfers control to the program segment starting at the memory location 1040H.

7. 3. 4
FILL: F ladr hadr byte

When terminating the sequence with the "RETURN" key, the memory space defined by and including the low and high addresses is filled with the constant byte given. If no constant value is given the memory space will be filled with zeroes.

Example:	F1010	101A	FF	fill area from 1010 to 101A with FF
	F1010	101A		fill area from 1010 to 101A with ØØ

7.3.5

SUBSTITUTE: S adr

When terminating the sequence with space, or the "RETURN" key, the screen displays the content of the byte specified by the address given. A new value can now be typed in. This value will replace the current content of the addressed byte when the next separator, space or comma or "RETURN", is entered. At the same time, the content of the next higher order byte is displayed for substitution. To leave a byte unchanged the space bar or "RETURN" is used after the display of the byte.

Example: S1000 3D-8F 1A = CB-3F 81-AE 78-FA

In the example above, digits entered by the user are underlined, and the space bar was used as separator. To return to the utility, press the "LEFTCURSOR" key. After escaping the sequence, the memory locations starting from address 1000 to 1004 will have the following contents:

1000: 8F, 1001: 1A, 1002: 3F, 1003: AE, 1004: FA

7.3.6

MOVE: M ladr hadr dadr

The MOVE command, when terminating the sequence with the "RETURN" key, moves a block of memory specified by the low and high addresses to a destination beginning with the destination address.

Example: M1000, 100A, 1100

After executing the above command, the program segment starting at address 1000 and ending at address 100A has been moved to a starting address at 1100, and it will occupy all the bytes up to and including address 110A. The original program segment at location 1000 is not destroyed.

The MOVE command is useful during program development when an instruction must be inserted into the program already stored in the RAM memory. For example, assume that three bytes must be inserted into a program field ranging from RAM location 1040 through 1075. The new bytes must occupy locations 1046, 1047, and 1048.

Using the MOVE command, the program segment ranging from 1046 through 1075 can be shifted right three bytes:

M1046 1075 1049

The three new bytes can now be inserted. Caution: the MOVE command does not adjust reference addresses within instructions.

CLASS 2. USER REGISTER COMMANDS

7.3.7

EXAMINE: X

When the above command is terminated by pressing the "RETURN" key, the screen displays the following CPU registers: Accumulator, Flags, Registers B through L, Stack Pointer, and the Program Counter.

Example:

X

A = 00 F = 46 B = 20 C = 44 D = 10 E = BF H = 11 L = 7A S = 11BE
P = 1040

The bit assignment of the flag-byte is as follows:

B7	SIGN
B6	ZERO
B5	ALWAYS ZERO
B4	AUXILIARY CARRY
B3	ALWAYS ZERO
B2	PARITY
B1	ALWAYS ONE
B0	CARRY

7.3.8

EXAMINE REGISTER: X reg

This command is exactly like the substitute command except that it allows substitution or initialization of the user-register copy area.

Example: Suppose we wish to initialize the accumulator to the value of 35 and register B to the value of FF. We can do this task in either of the following ways:

XA 00-35 46- 20-FF
or
XA 00-35
XB 20-FF

The digits entered by the user are underlined. In the first example the space bar was used as separator, and the value of the flags remained unchanged, since no replacement value was entered. In the second example the first substitution was terminated by the "LEFT ARROW" key.

7.3.9

VECTOR EXAMINE: V

When the "RETURN" key is pressed after the command, the console displays the contents of the user initialization and interrupt-transfer vector bytes.

Example:
V
0 = 00 M = 00 T = 10 G = 20 1 = 106F 2 = 1089 3 = 0040 4 = 0040
5 = 0040 6 = 0040 7 = 106F.

7.3.10

VECTOR EXAMINE BYTES: V byte

The function of this command is the same as that of the substitut or examine register commands. It allows changing the contents of the transfer vector or initialization bytes.

Example: V2 1089-1100

When the "CURSORLEFT" key is pressed after the sequence above, the interrupt 2 vector address is changed from 1089 to 1100.

CLASS 3 HEXADECIMAL I/O COMMANDS

7.3.11

READ: R adr

The address in the command is optional.
Pressing the "RETURN" key after the command, initiates action. The READ function will start reading the binary file from tape or disc as soon as the tape recorder or disc drive is turned on. While reading the tape, the utility checksums each record. If a read error occurs, the error exit is taken, the reading stops, and the control is returned to the user. In this case the tape may be read again by backing it up at least one record. The reading continues until the end of file record is read.

7.3.12

WRITE: W ladr hadr

After pressing the "RETURN" key the hexadecimal content of the memory range specified by the low and high addresses is output to the tape or disc. The format of this output is the packed hexadecimal format described below.

```
W600 60F
10060000B7C8CD380523C300060E0DCD3805C50188
|  |   | |                            |  |
|  |   | |                            |  └─ Check sum
|  |   | └────────────────────────────── Data
|  |   └── Code: 00 = Data
|  |             01 = End
|  └───── Starting load address
└──────── Number of bytes in datafield expressed in hexadecimal
```

W0⊔FFF⊔GEORGE↵

Writes the area of memory from 0 to FFF to disc or cassette under the name "GEORGE".

W0⊔1F↵

Writes the area 0 to 1F on cassette with no name. Unnamed files should not be used on disc. It is loaded back into exactly the same addresses as it was written from.

R1000⊔FRED↵

As above, but the data is read into addresses 1000 hex bytes higher than it was written from.

R↵

The next binary file on the cassette is read into memory. No offset is used. Note that unnamed files should not be used with discs.

The files created by the W and read in by the R command have a file type of 1. They cannot be accessed by, and will be ignored entirely by the LOAD, LOADA commands of BASIC. Similarily R will not read in files of types other than 1.

File names include every character typed between the space and the carriage return. There is no "character delete" facility, so great care should be taken.

Decimal	Character	Decimal	Character	Decimal	Character
000	NUL	031	US	062	>
001	SOH	032	SPACE	063	?
002	STX	033	!	064	@
003	ETX	034	"	065	A
004	EOT	035	#	066	B
005	ENQ	036	$	067	C
006	ACK	037	%	068	D
007	BEL	038	&	069	E
008	CH DEL	039	'	070	F
009	TAB	040	(	071	G
010	LF	041	)	072	H
011	VT	042	*	073	I
012	FF	043	+	074	J
013	CR	044	,	075	K
014	SO	045	-	076	L
015	SI	046	.	077	M
016	↑ CURS	047	/	078	N
017	↓ CURS	048	0	079	O
018	← CURS	049	1	080	P
019	→ CURS	050	2	081	Q
020	Shift+↑	051	3	082	R
021	Shift+↓	052	4	083	S
022	Shift+←	053	5	084	T
023	Shift+→	054	6	085	U
024	CAN	055	7	086	V
025	EM	056	8	087	W
026	SUB	057	9	088	X
027	£	058	:	089	Y
028	¢	059	;	090	Z
029	GS	060	<	091	(
030	RS	061	=	092	\

Decimal	Character	Decimal	Character	Decimal	Character
093	)	123	{		
094	↑	124	|		
095	←	125	}		
096	`	126	~		
097	a	127	DEL		
098	b				
099	c				
100	d				
101	e				
102	f				
103	g				
104	h				
105	i				
106	j				
107	k				
108	l				
109	m				
110	n				
111	o				
112	p				
113	q				
114	r				
115	s				
116	t				
117	u				
118	v				
119	w				
120	x				
121	y				
122	z				

LIST OF SOME USEFUL POKES

POKE #2C4, # FF   FORCE A BREAK

OUTPUT

POKE # 131,0 OUTPUT TO SCREEN + RS 232
,1 OUTPUT TO SCREEN
,2 TO EDIT BUFFER
,3 TO DISC — ⇒C E

INPUT

POKE #135,0 INPUT FROM K. B./SCREEN
,1 INPUT FROM STRING
2 INPUT FROM EDIT BUFFER TO PROGRAM AREA

TAPE CONTROL

POKE # 40, # 28   TAPE 1 ON
40, # 18   TAPE 2 ON
40, # 30   TAPE 1 AND 2 OFF

POKE # 13D, # 10   CASSETTE PORT 1 ACTIVATED
13D, # 20   "   " 2 "

SWITCH FLOPPY DRIVE

POKE # 730, # 30 FLOPPY DRIVE 0 ACTIVATED
730, # 31 FLOPPY DRIVE 1 ACTIVATED

AM 9511

UT
>SF BØØ
>
>B

UNIT FLOPPY DISK

UT
> Z3
> XA   30   USE DRIVE N° Ø
31   "   "   "   1
> G B6
> B

TOP OF STACK   #F900
BOTTOM OF STACK #F800

POKE # 2C4, # FF : FORCE A BREAK IN PROGRAM

ON TAPE "ACTIVATE"

TO ACTIVATE FLOPPY (2C5 TO 2E2)

2C5 C3 58 Ø5 C3 F2 Ø5 C3 12 Ø6 C3 A1
2DØ Ø5 C3 FB Ø5 C3 FC Ø6 C9 ØØ ØØ C3 75 Ø6 C3 29 Ø6
2EØ C3 5C Ø6 (2E2)
2A0 08 5D 08 5E 08

TO ACTIVATE CASSETTE (2C5 TO 2E2)

2C5 C3 B8 D2 C3 F1 D2 C3 27 D4 C3 25
2DØ D3 C3 40 D3 C3 45 D4 C3 A2 D3 C9 ØØ Ø¢ C9 ØØ ØØ
2EØ C3 B4 DD (2E2)

2AØ 33 ED 03 F6 03 50 B3 C5 E8

SOFTWARE PROTECTION

1. Write program in BASIC (Avoid putting REM)
2. UT
3. D2A1 2A4 (Pointers) ↵
   2A1 # # # #
   Low High Low High
   VAL 1 VAL 2
4. SAVE ON CASSETTE BY
   W (VAL 1 + 1) (VAL 2) FILE NAME (without double quote)
5. Protect by
   F(VAL 1+1) (VAL 2) C↵ C(C = Hex code for form feed)
6. B (return to BASIC)
7. SAVE ON CASSETTE (SAVE "FILENAME")
   When loading from cassette you cannot LIST nor EDIT anymore as all information is scrambled.

WHAT TO DO IF AN ACCIDENTAL RESET HAPPENED DURING PROGRAM KEYING OR AT END OF PROGRAM

1. Push on BREAK
2. Type UT return
3. Type S29F and 6 x Space bar, result is b a x x x x
4. Note b a x x x x
5. Cursor (←)
6. Type S a b space bar, result is x x
7. Note x x
8. Cursor (←)
9. Press B (BASIC)

If you accidentally RESET

1. Type UT return
2. Type S29F press 6 times space bar; result is x y &&&&
3. Change the 6 positions if different to what you noted.
4. S a b change the 2 " " " " " " "
   Cursor
5. Press B
6. Type EDIT press and BREAK Space

SAVING AND RELOADING A DRAWING

After you draw the picture for saving
Press on BREAK
Type MODE ? A (? being the mode in which you draw the picture)
Type UT Return
Type W XXXX BFFF PICTURE 1

To reload the picture
Type MODE ?A (? being the mode in which the picture was drawn)
Press UT Return
Type R

MODE 1
2 A B350 TO BFFF
3A A440 TO BFFF
4
5 5670 TO BFFF
6

```
                +
                ;
C003                    ORG     0C003H
                ;
C003            XMINIT: DS      3         ; PACKAGE INIT
                ;
C006            XFINM:  DS      3         ; INCR FPT NUMBER IN MEM
C009            XFDCM:  DS      3         ; DECR FPT NUMBER IN MEM
                ;
C00C            XFCOMP: DS      3         ; FLOATING POINT COMPARE
                ;
C00F            XIINM:  DS      3         ; INCR INT NUMBER IN MEM
C012            XIDCM:  DS      3         ; DECR INT NUMBER IN MEM
                ;
C015            XICOMP: DS      3         ; INTEGER COMPARE
                ;
C018            XPUSH:  DS      3         ; SAVE FPAC ON STACK
C01B            XPOP:   DS      3         ; RETRIEVE FPAC FROM STACK
                ;
                ; IO FUNCTIONS
                ;
C01E            XFCB:   DS      3         ; INPUT A FPT NUMBER TO FPAC
C021            XFBC:   DS      3         ; CONVERT A FPT NUMBER FOR OUTPUT
C024            XICB:   DS      3         ; INPUT INTEGER NUMBER TO IAC
C027            XIBC:   DS      3         ; CONVERT INTEGER FOR OUTPUT
C02A            XHCB:   DS      3         ; INPUT HEX NUMBER TO IAC
C02D            XHBC:   DS      3         ; CONVERT IAC TO HEX FOR OUTPUT
C030            XPRTY:  DS      3         ; PRETTIES UP FPT OR INTEGER NUMB
                ;
C033            DECBUF: DS      2         ; LOCATION OF OUTPUT BUFFER
                ;
                +       PAGE
```



```
                +
                ;
                ; MEMORY + IO MAP
                ;
                ; DEFINES WHERE TO FIND THE HARDWARE
                ;
FB00            MTHAD   EQU     0FB00H  ; MATH CHIP (IF FITTED)
                ;
FC00            SNDAD   EQU     0FC00H  ; 8253 ADDRESS (IF FITTED)
                ;
FC00                    SND0    EQU     SNDAD   ; CHAN 0
FC02                    SND1    EQU     SNDAD+2 ; CHAN 1
FC04                    SND2    EQU     SNDAD+4 ; CHAN 2
FC06                    SNDC    EQU     SNDAD+6 ; CONTROL
FC00                    PDLCH   EQU     SND0    ; PADDLE READING CHANNEL
                ;
                        ; 8253 MODE BYTES
                ;
0032                    COM1    EQU     032H    ; CHAN 0, MODE 1, 2 BYTE OPERA
                ;
0036                    COM3    EQU     036H    ; CHAN 0, MODE 3, 2 BYTE
0076                    C1M3    EQU     076H
00B6                    C2M3    EQU     0B6H
                ;
0030                    COM0    EQU     030H    ; CHAN 0, MODE 0, 2 BYTE OP
                ;
0000                    COFIX   EQU     0       ; FIX COUNT ON CHANNEL 0
                ;
FD00            PORI    EQU     0FD00H  ; INPUT PORT
                ;
0004                    PIPGE   EQU     04H     ; PAGE SIGNAL
                ;
0008                    PIDTR   EQU     08H     ; SERIAL OP READY
                ;
0010                    PIBU1   EQU     10H     ; BUTTON ON PADDLE 1
                ;
0020                    PIBU2   EQU     20H     ; BUTTON ON PADDLE 2
                ;
0040                    PIRPI   EQU     40H     ; RANDOM BITS
                ;
0080                    PICAI   EQU     80H     ; CASSETTE INPUT DATA
                ;
FD01            PDLST   EQU     0FD01H  ; PADDLE SAMPLING START
                ;
FD04            PORO    EQU     0FD04H  ; VOLUME OUTPUTS CHANS 0, 1
                ;
FD05            POR1    EQU     PORO+1  ; VOLUMES CHAN 2 AND NOISE
                ;
```

```
FD06        PORO    EQU     0FD06H  ; OUTPUT PORT
            ;
0001                POCAS   EQU     01H     ; CASSETTE OUTPUT BIT
0007                PDLMSK  EQU     7       ; PADDLE SELECT BITS
            ;
0008                POPNA   EQU     08H     ; PADDLE ENABLE BIT
            ;
0010                POCM1   EQU     10H     ; CASSETTE MOTOR CONTROL 1
0020                POCM2   EQU     20H     ;     "         "       " 2
            ;
                    ; TOP 2 BITS ARE BANK SWITCHING
            ;
FE00        GIC     EQU     0FE00H  ; RWBUS GIC ADDRESS
            ;
0080                RWMOP   EQU     080H    ; RW OUTPUT MODE
            ;
0090                RWMIP   EQU     090H    ; RW INPUT MODE
            ;
FFF0        TICC    EQU     0FFF0H  ; TICC ADDRESS
            ;
F900        STTOP   EQU     0F900H  ; TOP OF STACK RAM
            ;
F800        SRBOT   EQU     0F800H  ; BOTTOM OF STACK RAM
            ;
           +        PAGE
```



```
           +
            ;
            ; VARIABLES:-
            ;
0100                ORG     0100H
            ;
            ; USER STATE:
            ;
            ; FOLLOWING ARE SAVED BY SOFT BREAK
            ;
            SYSBOT:
            ;
0100        CURRNT: DS      2       ; START OF CURRENT LINE
            ;
0102        BRKPT:  DS      2       ; START OF CURRENT COMMAND
            ;
0104        LOPVAR: DS      2       ; POINTS TO CURRENT LOOP VARIAB
                                    ; 0 IF NO RUNNING LOOP
            ;
0106        LSTPF:  DS      1       ; FLAG FOR INTEGER/FPT LOOP
                                    ; AND IMPLICIT/EXPLICIT STEP
            ;
0107        LSTEP:  DS      4       ; STEP VALUE IF EXPLICIT
            ;
010B        LCOUNT: DS      4       ; LOOP ITERATION COUNT
            ;
010F        LOPPT:  DS      2       ; POINTER TO START LOOP
            ;
0111        LOPLN:  DS      2       ; POINTER TO START LOOP LINE
            ;
0010        FRAME   EQU     $-LOPVAR+1 ; ALLOW FOR FLAGS WHEN PUSHI
            ;
0113        STKGOS: DS      2       ; STACK LEVEL AT LAST GOSUB
                                    ; 0 IF NO ACTIVE CALL
            SYSTOP:
            ;
            STRFL:                  ; TRACE/STEP FLAGS TOGETHER
            ;
0115        TRAFL:  DS      1       ; TRACE FLAG
0116        STEPF:  DS      1       ; STEP FLAG
            ;
0117        RDIPF:  DS      1       ; FLAG SET WHILE RUNNING INPUT
0118        RUNF:   DS      1       ;   "   "   "     "   PROGRA
            ;
            ; PREVIOUS 2 BYTES MUST BE CONSECUTIVE
           +        PAGE
```

```
              +
               ;
               ; RUNTIME SCRATCH AREA
               ;
               GSNWK:                      ; SCRATCH AREA FOR GOSUB/NEXT (2 BYTES
               LISW1:                      ; START OF LISTED AREA
               ;
0119           COLWK:  DS      2           ; SCRATCH AREA FOR SCOLG, SCOLT (4 BYTE
               ;
011B           LISW2:  DS      2           ; END LISTED AREA
               ;
               ; SAVE AREA FOR RESTART ON ERROR.
               ;
011D           ERSSP:  DS      2           ; STACK POINTER
               ;
011F                   DS      3           ;*
               ;*
0122           ERSFL:  DS      1           ; SET IF ENCODING A STORED LINE
               ;
               ; DATA/READ VARIABLES
               ;
0123           DATAC:  DS      1           ; OFFSET OF NEXT CH TO ENCODE IN "DATA
               ;
0124           DATAP:  DS      2           ; POINTER TO CURRENT DATA LINE
               ;!DATAQ: DS      2           ; POINTER AFTER CURRENT D. LINE IF ANY
               ;
0126           CONFL:  DS      1           ; SET IF THERE IS A SUSPENDED PROGRAM
               ;
0127           STACK:  DS      2           ; CURRENT BASE STACK LEVEL
               ;
0015           SFRAME  EQU     SYSTOP-SYSBOT
               ;
               ; SCRATCH LOCN FOR EXPRESSION EVALUATION
               ;
0129           WORKE:  DS      4
               ;
               ; RANDOM NUMBER KERNEL
               ;
012D           RNUM:   DS      4
               ;
               ;!RNDLY: DS      1           ; RANDOM NUMBER DELAY COUNT
              +        PAGE
```



```
              +
               ;
               ; OUTPUT SWITCHING
               ;
0131           OTSW:   DS      1           ; 0 TO OUTPUT TO SCREEN+RS232
                                           ; 1 OUTPUT TO SCREEN
                                           ; 2 TO EDIT BUFFER
                                           ; 3 TO DISK
               ;
               ; INPUT SWITCHING
               ;
               ;!INSW:  DS      1
                                           ; 0 FROM KEYBOARD
                                           ; 1 FROM DISK
               ;
               ;
               ; ENCODING INPUT SOURCE SWITCHING
               ;
0132           EFEPT:  DS      2           ; POINTER
0134           EFECT:  DS      1           ; COUNT
               ;
0135           EFSW:   DS      1           ; SET 0:        INPUT FROM KB/SCREEN
                                           ;     1:          "     "   STRING
                                           ;     2:          "     "   EDIT BUF
               ;
               ; VARIABLES USED DURING EXPRESSION ENCODING
               ; (COULD OVERLAP WITH RUNTIME VARIABLES)
               ;
0136           TYPE:   DS      1           ; TYPE OF LATEST EXPRESSION OR ITEM
               ;
0137           RGTOP:  DS      1           ; LATEST PRIORITY OPERATOR
               ;
0138           OLDOP:  DS      1           ; OLD PRIORITY+OPERATOR
               ;
0139           HOPPT:  DS      2           ; PTR TO PLACE FOR OPERATOR
               ;
013B           RGTPT:  DS      2           ; PTR TO RGT OPERAND LATEST OPERATOR
               ;
               ; ORDER OF LAST 7 BYTES IS IMPORTANT
               ;
              +        PAGE
```

```
                +
                ;
                ; MASK TO SELECT CASSETTE 1 OR 2
                ;
013D            CASSL:  DS      1       ; #10 FOR CASSETTE 1, #20 FOR 2
                ;
                ; ENCODED INPUT BUFFER
                ;
013E            EBUF:   DS      128     ; USED ALSO BY UTILITY
                ;
                ; INTERRUPT HANDLER VARIABLES
                ;
005F            TICIM   EQU     05FH    ; CURRENT INTERRUPT MASK
                ;
01BE            TIMER:  DS      2       ; TIMER LOCATION
                ;
01C0            CTIMR:  DS      1       ; CURSOR CLOCK
                ;
000F            CTIMV   EQU     15      ; FLASH TIME IN 20 MS UNITS
                ;
01C1            KBXCT:  DS      1       ; EXTEND KB SCAN TIME COUNTER
                ;
0002            KBXCK   EQU     2       ; KB SCAN TIME (UNITS OF 16 MS)
                                        ; RAND ROUTINE NEEDS THIS EVEN
                ;
                ; INTERRUPT MASKS DEFINITIONS
                ;
FFFB            SNDIAD  EQU     TICC+0BH ; SOUND TIMER ADDR
0008            SNDIM   EQU     08H     ; SOUND INT MASK BIT
                ;
FFFC            KBIAD   EQU     TICC+0CH ; KB TIMER ADDR
0040            KBIM    EQU     40H     ; KEYBOARD "   "   "
                ;
0080            CLKIM   EQU     080H    ; CLOCK  "   "   "
                ;
0004            STKIM   EQU     04H     ; STACK  "   "   "
                +       PAGE
```



```
                +
                ;
                ; IO LOCATIONS
                ;
                ; !POROM:         DS      1       ; MEMORY OF
                ; !POR1M:         DS      1       ; LAST OUTPUTS TO
0040            POROM   EQU     40H     ; OUTPUT PORTS
                ;
                ; SOUND CONTROL BLOCK STORAGE
                ; "
000E            SCBL    EQU     14      ; LENGTH OF A SOUND CONTROL BL
0009            NCBL    EQU     9       ;    "    "   NOISE    "
                ;
)1C2            SCB0:   DS      3*SCBL+NCBL ; SOUND + NOISE CHANNELS
                ;
                ; ENVELOPE STORAGE
                ;
)040            ENVLL   EQU     64      ; NUMBER OF BYTES/ENVELOPE
                ;
)002            NUMENV  EQU     2       ; NUMBER OF ENVELOPES
                ;
)1F5            ENVST:  DS      NUMENV*ENVLL ; ENVELOPE STORAGE
                ;
)275            IMPTAB: DS      'Z'-'A'+1 ; IMPLICIT TYPE TABLE
                ;
)28F            IMPTYP: DS      1       ; DEFAULT NUMBER TYPE
                ;
)290            REQTYP: DS      1       ; REQUIRED NUMBER TYPE
                ;
                ; SPARE VARIABLE SPACE
                ;
(291                    DS      10
(291            DATAQ   EQU     0291H   ;*
(293            RNDLY   EQU     0293H   ;*
(294            POROM   EQU     0294H   ;*
(295            POR1M   EQU     0295H   ;*
(296            INSW    EQU     0296H   ;*
                +       PAGE
```

```
                +
                ;
                ; HEAP/TEXT BUFFER/SYMTAB POINTERS
                ;
29B             HEAP:    DS      2       ; START OF HEAP
                ;
29D             HSIZE:   DS      2       ; SIZE OF HEAP
100             HSIZD    EQU     100H    ; DEFAULT SIZE
                ;
29F             TXTBGN:  DS      2       ; START OF TEXT BUFFER
                ;
                TXTUSE:                  ; END TEXT AREA AND
2A1             STBBGN:  DS      2       ; START SYMBOL TABLE
                ;
2A3             STBUSE:  DS      2       ; END SYMBOL TABLE
                ;
2A5             SCRBOT:  DS      2       ; BOTTOM OF SCREEN RAM AREA
                ;
                +        PAGE
```



```
                +
                ;
                ; KEYBOARD VARIABLES + CONSTANTS
                ;
02A7            KBTPT:   DS      2       ; POINTER TO CODE TABLE
                ;
02A9            MAP1:    DS      8       ; LATEST SCAN OF KEYS
                ;
02B1            MAP2:    DS      8       ; PREVIOUS SCAN
                ;
02B9            KNSCAN:  DS      1       ; SET TO SCAN FOR BREAK ONLY
                ;
0004            KBLEN    EQU     4       ; LENGTH OF ROLLOVER BUFFER
                KEYL:
02BA            KLIND:   DS      KBLEN   ; CIRCULAR BUFFER FOR KEYS PRE
                ;
02BE            KLIIN:   DS      2       ; NEXT POSN FOR INPUT TO KLIND
02C0            KLIOU:   DS      2       ; NEXT POSN FOR OUTPUT FROM KL
                ;
02C2            RPCNT:   DS      1       ; COUNT FOR REPT
                ;
02C3            SHLK:    DS      1       ; SET IF "SHIFT INVERT"
                ;
                         IF SUSP
                ;
02C4            KBRFL:   DS      1       ; FLAG FOR "BREAK PRESSED"
                ;
                         ENDIF
                ;
02B0            SHLOC    EQU     MAP1+7  ; BYTE CONTAINING SHIFT
0040            SHMSK    EQU     040H    ; SHIFT KEY BIT
                ;
02AF            RPLOC    EQU     MAP1+6  ; BYTE CONTAINING REPT KEY
0020            RPMSK    EQU     020H    ; REPT KEY BIT
                ;
0002            RPLIM    EQU     2       ; TIMING FOR REPT
                ;
0040            BRSEL    EQU     040H    ; COLUMN SELECT MASK FOR BREAK
0040            BRMSK    EQU     040H    ; BREAK KEY BIT
                ;
0020            BRLIM    EQU     20H     ; TIMING FOR HARD BREAK
                ;
                +        PAGE
```

```
                +
                ;
                ;   DISC/CASSETTE SWITCHING VECTOR
                ;
                IOVEC:
                ;
 02C5           WOPEN:  DS      3
                ;
 02C8           WBLK:   DS      3
                ;
 02CB           WCLOSE: DS      3
                ;
 02CE           ROPEN:  DS      3
                ;
 02D1           RBLK:   DS      3
                ;
                RCLOSE:
 02D4           RCLO:   DS      3
                ;
 02D7           MBLK:   DS      3
                ;
 02DA           RESET:  DS      3
                ;
 02DD           DOUTC:  DS      3
                ;
 02E0           DINC:   DS      3
                ;
 02E3                   DS      3       ; SPARE
                ;
 02E6           TAPSL:  DS      2
                ;
 02E8           TAPSD:  DS      2
                ;
 02EA           TAPST:  DS      2
                ;
                VAREND:
                VARLAST:
                ;
 02EC           RAM     SET     $
                ;
               +        PAGE
```



```
                +
                ;
C6C0                    ORG     0C6C0H  ; START OF BASIC
                ;
                ; BANK SWITCHING RESTARTS
                ;
                ; THE FOLLOWING ROUTINES SWITCH THE PAGED
                ;  BANKS OF ROM.  THEY ARE ENTERED VIA RST INSTRUCTIONS
                ;
                MARST:
                ;
C6C0 E1                 POP     H
                ;
C6C1 F3                 DI
                ;
C6C2 224300             SHLD    RSWK2   ; SAVE HL
C6C5 F5                 PUSH    PSW
C6C6 E1                 POP     H
C6C7 224100             SHLD    RSWK1   ; PSW
                ;
C6CA 2640               MVI     H,040H  ; BANK SELECT BITS FOR MATH PA
C6CC 3AD400             LDA     MVECA   ; OFFSET OF START HW/SW VECTOR
                MRS10:
C6CF E3                 XTHL
C6D0 86                 ADD     M       ; ADD ENTRY NUMBER
C6D1 23                 INX     H
C6D2 E3                 XTHL
                ;
C6D3 6F                 MOV     L,A     ; COMPLETE ENTRYPOINT ADDRESS
C6D4 3A4000             LDA     POROM   ; BANK SELECT PORT STATUS
C6D7 F5                 PUSH    PSW     ; REMEMBER
C6D8 E63F               ANI     03FH    ; KEEP OTHER BITS
C6DA B4                 ORA     H       ; ADD NEW SELECT BITS
C6DB 324000             STA     POROM   ; UPDATE MEMORY
C6DE 3206FD             STA     PORO    ; AND PORT
                ;
C6E1 26E0               MVI     H,VECA SHR 8
C6E3 CDF2C6             CALL    MRDCL
                ;
C6E6 E3                 XTHL
C6E7 F5                 PUSH    PSW
C6E8 7C                 MOV     A,H
C6E9 324000             STA     POROM   ; REINSTATE MEMORY
C6EC 3206FD             STA     PORO    ; + PORT
C6EF F1                 POP     PSW
C6F0 E1                 POP     H
C6F1 C9                 RET             ; BACK TO CALLER
                ;
```

DAI 8080 ASSEMBLY SERVICE,D2.2 PAGE 23
BASIC V1.0 DISK EDIT 7 2-MARCH-80

```
                  MRDCL:
C6F2 E5                  PUSH    H
C6F3 2A4100              LHLD    RSWK1
C6F6 E5                  PUSH    H
C6F7 F1                  POP     PSW
C6F8 2A4300              LHLD    RSWK2
C6FB FB                  EI
C6FC C9                  RET

                         PAGE
```


THIS PROGRAM NAMED SUM IS CALLING A MACHINE LANGUAGE SUBROUTINE LOADED AS AN ARRAY ''A'' NAMED ''SUM A'' THE SUBROUTINE ,LOCATED AT #3FC , PERFORMS INTEGER CALCULATION WITH 64 DIGITS RESOLUTION. YOU MUST LOAD THE PROGRAM, STOP THE RECORDER IF YOU DO NOT USE THE REMOTE CONTROL, RUN THE PROGRAM WHAT IS NOW LOADING THE ROUTINE AS AN ARRAY AND ASK YOU THE OPERATION TO PERFORM I.E. 12345+432 <RETURN> AND GIVES THE RESULT. IF YOU PRESS THE BREAK KEY TO CONTINUE YOU HAVE NOW TO RUN 35 ,OR FIRST TYPE 1 <RETURN> TO 24 <RETURN> WHAT WILL ERASE THIS TEXT AND LOADA ROUTINE AND YOU CAN NOW MAKE A NORMAL RUN. IF YOU WANT TO SAVE THE PROGRAM AND THE ROUTINE YOU MUST SAVE''PROGRAM NAME'' STOP RECORDER, SAVEA A''ROUTINE NAME''

YOU WILL NOTICE IF YOU LIST THE PROGRAM THAT 3 FIRST LINES ARE CLEAR 2000, DIM A(20,20), LOADA A''SUM A'' AFTER YOU HAVE LOADED THE ARRAY YOU CANNOT EDIT NOR CLEAR NOR DIM ARRAYS ALREADY DIMENSIONED.

PRESS ANY KEY CONTINUE THE PROGRAM LOADING ROUTINE

```
10    CLEAR 2000
20    DIM A(20,0,20,0)
30    LOADA A "SUM A"
35    PRINT "WHAT IS YOUR SUM    ";
40    INPUT A$
45    PRINT
50    CALLM #3FC,A$
60    PRINT "HERE IS THE ANSWER!",A$
70    GOTO 35
```

```
03F0 00 00 00 00 00 00 00 00 00 00 00 00 F5 C5 D5 7E
0400 23 66 6F E5 4E 23 CD 2F 05 11 99 06 CD F1 04 CA
0410 99 04 78 32 5E 07 7E 36 20 32 60 07 23 0D CA 89
0420 04 11 DA 06 CD F1 04 78 32 5F 07 21 1B 07 11 99
0430 06 01 DA 06 3A 60 07 FE 2A CA 6B 04 FE 2F CA 7F
0440 04 FE 2B CA 52 04 FE 2D C2 89 04 3A 57 07 2F 32
0450 5F 07 3A 5E 07 A7 CC F1 05 C4 DA 05 C5 D1 3A 5F
0460 07 A7 CC F1 05 C4 DA 05 C3 92 04 CD 06 06 CA 89
0470 04 3A 5E 07 47 3A 5F 07 A8 32 5D 07 C3 92 04 CD
0480 22 06 CA 89 04 C3 71 04 E1 E1 D1 C1 F1 23 36 3F
0490 00 C9 CD 61 07 3A 5D 07 E1 E5 23 06 00 A7 CA A5
04A0 04 36 2D 23 04 11 5B 07 1B 1A A7 CA A8 04 E5 21
04B0 E6 F8 19 4D E1 1A F6 30 77 23 1B 0D FA EB 04 C2
04C0 B5 04 F1 5E 16 00 4A E5 19 7E FE 20 C2 D5 04 2B
04D0 C2 1D C2 C9 04 79 D6 03 FA E6 04 0D 0D E1 E5 73
04E0 23 19 36 00 23 71 E1 D1 C1 F1 C9 2B 36 30 C3 C2
04F0 04 06 00 7E E6 30 FE 30 CA 15 05 2B 23 0D C8 7E
0500 36 20 FE 20 CA FC 04 23 FE 2B CA 15 05 FE 2D C2
0510 00 04 3E FF 47 7E FE 20 CA 29 05 E6 30 FE 30 C0
0520 CD BF 05 7E 36 20 E6 0F 12 23 0D C2 15 05 C9 E5
0530 21 99 06 1E 40 36 FF 23 1D C2 35 05 21 DA 06 1E
0540 40 36 FF 23 1D C2 41 05 21 1B 07 1E 40 36 00 23
0550 1D C2 4D 05 AF 32 5D 07 32 5E 07 32 5F 07 E1 C9
0560 F5 D5 C5 E5 11 40 00 19 EB E1 CD 71 05 C1 D1 E1
0570 C9 D5 1B 1A A7 C2 93 05 7B BD C2 72 05 D1 AF 32
0580 5D 07 C9 F2 9A 05 3A 5D 07 2F 32 5D 07 D5 13 1B
0590 1A 2F 3C 12 7B BD C2 8F 05 D1 13 13 E5 06 00 7E
05A0 7C 06 00 F2 B6 05 05 C6 0A FA A6 05 77 23 7B BD
05B0 C2 9F 05 E1 D1 C9 04 D6 0A F2 B6 05 C3 A6 05 C5
05C0 05 CD C9 05 D1 C1 FE 90 C9 1A F5 AF 12 F1 13 47
05D0 1A F5 78 12 F1 A7 F2 CE 05 C9 F5 D5 E5 CD E4 05
05E0 F1 D1 F1 C9 1A FE FF C8 2F 3C 86 77 13 23 C3 E4
05F0 05 F5 D5 E5 CD FB 05 E1 D1 F1 C9 1A FE FF C8 86
0600 77 13 23 C3 FB 05 0A 3D 02 FA 12 06 CD F1 05 C3
0610 06 06 CD 60 05 03 0A 2F D6 01 D8 CD BF 05 C8 C3
0620 06 06 AF 32 5C 07 55 21 D9 06 2B 7E FE FF C2 36
0630 06 36 00 C3 2A 06 E1 36 01 CD 83 06 FA 52 06 23
0640 D5 C5 D1 CD BF 05 D1 C8 3A 5C 07 3C 32 5C 07 C3
0650 37 06 35 CD 83 06 2B D5 C5 D1 CD 70 06 D1 3A 5C
0660 07 3D 32 5C 07 F8 34 CD 83 06 FA 52 06 C3 66 06
0670 C5 D5 CD 78 06 D1 C1 C9 13 1A 1B 12 13 FE FF C8
0680 C3 78 06 E5 C5 D5 D5 E1 C5 D1 CD DA 05 CD 60 05
0690 D1 C1 E1 3A 5D 07 A7 C9 80 00 00 00 00 00 00 00
06A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0700 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0710 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0720 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0730 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0740 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0750 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0760 00 21 1B 07 C3 60 05 00 00 00 00 00 00 00 00 00
```

REAL TIME CLOCK


```
5     CLEAR 300
6     POKE #29C,3:POKE #29E,0:POKE #3EC,#80:POKE #3ED,#28
10    FOR T%=0 TO 11:READ D%
20    FOR T1%=0 TO 15:READ D1%
25    IF D1%>=#100 THEN D1%=(PEEK(#2A6) IAND #FE IOR #E)+D1%-#100
26    POKE D%,D1%:D%=D%+1:NEXT:NEXT
30    POKE #71,#3:POKE #70,#0
100   DATA #300,#C5,#D5,#E5,#F5,#21,#89,#03,#06,#0A,#0E,#06,#16,#00,#1E,#32,#
110   DATA #310,#7B,#BE,#C2,#57,#03,#72,#23,#34,#78,#BE,#C2,#5E,#03,#72,#23,#
120   DATA #320,#79,#BE,#C2,#5E,#03,#72,#23,#34,#78,#BE,#C2,#5E,#03,#72,#23,#
130   DATA #330,#79,#BE,#C2,#5E,#03,#72,#23,#34,#78,#BE,#C2,#5E,#03,#72,#23,#
140   DATA #340,#2B,#23,#3E,#02,#BE,#C2,#5E,#03,#2B,#3E,#04,#BE,#C2,#5E,#03,#
150   DATA #350,#00,#23,#36,#00,#C3,#5E,#03,#F1,#E1,#D1,#C1,#C3,#A9,#D9,#3A,#
160   DATA #360,#101,#FE,#7A,#C2,#57,#03,#21,#BA,#03,#7E,#C6,#30,#32,#F1,#100
170   DATA #370,#7E,#C6,#30,#32,#F3,#100,#23,#7E,#C6,#30,#32,#F7,#100,#23,#7E
180   DATA #380,#30,#32,#F9,#100,#23,#7E,#C6,#30,#32,#FD,#100,#23,#7E,#C6,#30
190   DATA #390,#FF,#100,#3E,#FF,#32,#5C,#100,#32,#EE,#100,#32,#F0,#100,#32,#
#100
200   DATA #3A0,#32,#F4,#100,#32,#F6,#100,#32,#F8,#100,#32,#FA,#100,#32,#FC,#
,#32
210   DATA #3B0,#FF,#100,#32,#00,#101,#00,#C3,#5F,#03,#1A,#00,#00,#00,#00,#00
0
500   INPUT "INPUT THE TIME  < HH,MM,SS > ";T$:PRINT :A%=#3BF
510   FOR D%=0 TO LEN(T$)-1:T1$=MID$(T$,D%,1)
520   IF ASC(T1$)>47 AND ASC(T1$)<58 THEN POKE A%,VAL(T1$):A%=A%-1:IF A%=#3B
TH STOP
550   NEXT:STOP
```

[illegible] UTILITY V2.3

0300 3FF

```
0300 C5 D5 E5 F5 21 B9 03 06 0A 0E 06 16 00 1E 32 34
0310 78 BE C2 57 03 72 23 34 78 BE C2 5E 03 72 23 34
0320 78 BE C2 5E 03 72 23 34 78 BE C2 5E 03 72 23 34
0330 78 BE C2 5E 03 72 23 34 78 BE C2 5E 03 72 23 34
0340 78 23 3E 02 BE C2 5E 03 2B 3E 04 BE C2 5E 03 36
0350 00 23 36 00 C3 5E 03 F1 E1 D1 C1 C3 A9 D9 3A EF
0360 7F FE 7A C2 57 03 21 9A 03 7E C6 30 32 F1 7E 23
0370 7E C6 30 32 F3 7E 23 7E C6 30 32 F7 7E 23 7E C6
0380 30 32 F9 7E 23 7E C6 30 32 FD 7E 23 7E C6 30 32
0390 FF 7E 3E FF 32 EC 7E 32 EE 7E 32 F0 7E 32 F2 7E
03A0 32 F4 7E 32 F6 7E 32 F8 7E 32 FA 7E 32 FC 7E 32
03B0 FE 7E 32 00 7F 00 C3 5F 03 29 09 01 00 02 06 00
03C0 45 35 2C 23 46 35 2C 23 32 31 2C 23 42 39 2C 23
03D0 30 33 2C 23 30 36 2C 23 30 41 2C 23 30 45 2C 23
03E0 30 36 2C 23 31 36 2C 23 30 30 2C 23 00 07 30 36
03F0 30 31 35 32 35 00 01 35 80 01 32 80 19 18 00 00
```


*
R O T A T I N G   P Y R A M I D

================================

```
2     PRINT "ROTATING PYRAMIDE ,1,2,3 AND 4 ARE USED"
3     PRINT "WITH REPT KEY FOR ROTATION":WAIT TIME 400
5     MODE 6:MODE 6:SF=3.5:REM MODE +SCALING FACTOR
6     COLORG 0 15 0 15
7     GOSUB 2000:REM INITIALISE DATA
90    REM
92    GOSUB 800:REM DRAW NEW SHAPE
95    COLORG 0 15*(1-Q) 15*Q 15
96    GOSUB 900:REM ERASE OLD SHAPE
97    Q=1.0-Q
99    KS=ABS(KS)
100   A=GETC:IF A<ASC("0") THEN 100
120   FOR P=1.0 TO NP
130   XX(P)=X(P):YY(P)=Y(P)
140   NEXT
141   REM
150   ON A-ASC("0") GOTO 500,510,600,610,700,710
160   GOTO 100
161   REM
162   REM
500   KS=-KS
510   FOR P=1.0 TO NP
520   X=X(P):Y=Y(P)
530   X(P)=X*KC+Y*KS
540   Y(P)=Y*KC-X*KS
550   NEXT
560   GOTO 90
590   REM
591   REM
600   KS=-KS
610   FOR P=1.0 TO NP
620   Y=Y(P):Z=Z(P)
630   Y(P)=Y*KC+Z*KS
640   Z(P)=Z*KC-Y*KS
650   NEXT
660   GOTO 90
661   REM
662   REM
700   KS=-KS
710   FOR P=1.0 TO NP
720   Z=Z(P):X=X(P)
730   Z(P)=Z*KC+X*KS
740   X(P)=X*KC-Z*KS
```

```
750   NEXT
760   GOTO 90
800   REM
801   REM DRAW NEW PICTURE
802   REM
810   FOR L=1.0 TO NL
820   PA=LA(L)
830   PB=LB(L)
840   DRAW X(PA)+XC,Y(PA)+YC X(PB)+XC,Y(PB)+YC 17+Q*2
850   NEXT
860   RETURN
900   REM
901   REM ERASE OLD PICTURE
902   REM
910   FOR L=1.0 TO NL
920   PA=LA(L)
930   PB=LB(L)
940   DRAW XX(PA)+XC,YY(PA)+YC XX(PB)+XC,YY(PB)+YC 18-2*Q
950   NEXT
960   RETURN
990   REM
991   REM DATA SETUP ROUTINE
992   REM
2000  PHI=PI/20.0
2010  KS=SIN(PHI)
2020  KC=COS(PHI)
2030  XC=XMAX/2.0
2040  YC=YMAX/2.0
2050  Q=1.0
2100  READ NP,NL
2110  DIM X(NP),Y(NP),Z(NP)
2120  DIM XX(NP),YY(NP)
2130  DIM LA(NL),LB(NL)
2131  REM
2200  FOR P=1.0 TO NP
2210  READ X(P),Y(P),Z(P)
2211  X(P)=X(P)*SF
2212  Y(P)=Y(P)*SF
2213  Z(P)=Z(P)*SF
2220  NEXT
2221  REM
2230  FOR L=1.0 TO NL
2240  READ LA(L),LB(L)
2250  NEXT
2251  REM
2260  GOSUB 800
2270  RETURN
2300  REM
2301  REM DATA
2302  REM
2800  REM NUMBER OF POINTS AND NUMBER OF LINES
2900  DATA 5,8
2901  REM
2903  DATA 0,0,20
2904  DATA 20,20,-20
2905  DATA 20,-20,-2
2906  DATA -20,20,-2
2907  DATA -20,-20,-
2909  REM
2910  DATA 1,2
2911  DATA 1,3
2912  DATA 1,4
```



```
2913  DATA 1,5
2914  DATA 2,3
2915  DATA 2,4
2916  DATA 3,5
2917  DATA 4,5
2999  DATA 8,12
4000  DATA 1,2
4001  REM DATA FOR SOMETHING ELSE!
4002  REM
4009  DATA 20,20,20
4010  DATA 20,20,-20
4020  DATA 20,-20,20
4030  DATA 20,-20,-20
4040  DATA -20,20,20
4050  DATA -20,20,-20
4060  DATA -20,-20,20
4070  DATA -20,-20,-20
4110  DATA 1,3
4120  DATA 1,5
4130  DATA 2,4
4140  DATA 2,6
4150  DATA 3,4
4160  DATA 3,7
4170  DATA 4,8
4180  DATA 5,6
4190  DATA 5,7
4210  DATA 7,8
9999  END
*
```

C R A P S

=========

```
1       C1=1.0
2       C2=0.0
3       C3=14.0
4       C0=13.0
10      COLORG C0 C1 C2 C3:COLORT C0 0 0 0
11      MODE 3A
12      H=GETC
100     REM DRAW 14,19 14,68 C1
110     REM DRAW 14,68 63,68 C1
120     REM DRAW 63,68 63,19 C1
130     REM DRAW 63,19 14,19 C1
140     FILL 15,20 62,67 C2
150     REM DRAW 94,19 94,68 C1
160     REM DRAW 94,68 143,68 C1
170     REM DRAW 143,68 143,19 C1
180     REM DRAW 143,19 94,19 C1
190     FILL 95,20 142,67 C2
200     GOSUB 1200
210     PFS=0.0:TOSS%=0
212     CURSOR 0,3:PRINT "                    TO SHOOT CRAPS PRESS ANY KEY          ":
213     CURSOR 0,2:PRINT "      point                                       tosses":
214     CURSOR 0,1:PRINT "                                                        ":
215     CURSOR 0,0:PRINT "                                                        ":
216     CURSOR 28,2:PRINT "$";:CURSOR 28,2
220     GOSUB 1300
251     IF SUM%=7.0 OR SUM%=11.0 THEN CURSOR 25,1:GOSUB 1500:GOTO 210
252     IF SUM%=2.0 OR SUM%=3.0 OR SUM%=12.0 THEN CURSOR 24,1:GOSUB 1600:GOTO 2
253     POINT%=SUM%
254     GOSUB 1400:GOSUB 1300
255     IF POINT%=SUM% THEN CURSOR 25,1:GOSUB 1500:GOTO 210
260     IF SUM%=7 THEN CURSOR 25,1:GOSUB 1600:GOTO 210
280     GOTO 254
700     D=1.0+INT(10.0*RND(1.0)):IF D>6.0 GOTO 700
800     A=V+19.0
801     A1=A+7.0
802     B=V+35.0
803     B1=B+7.0
804     C=V+51.0
805     C1=C+7.0
810     IF D=1.0 OR D=3.0 OR D=5.0 THEN FILL B,40 B1,47 C3
820     IF D=1 THEN RETURN
830     FILL A,56 A1,63 C3
835     FILL C,24 C1,31 C3
```



```
840     IF D<4 THEN RETURN
850     FILL A,24 A1,31 C3
855     FILL C,56 C1,63 C3
860     IF D<6 THEN RETURN
870     FILL A,40 A1,47 C3
875     FILL C,40 C1,47 C3
880     RETURN
1200    FILL 19,24 58,63 C2
1210    FILL 99,24 138,63 C2
1220    V=0.0:GOSUB 700
1230    SUM%=INT(D)
1240    V=80.0:GOSUB 700
1245    SUM%=SUM%+INT(D)
1250    RETURN
1300    WAIT TIME 10:H=GETC:IF H=0.0 GOTO 1300:GOSUB 1200:RETURN
1400    CURSOR 6,1:IF POINT%<>0 THEN PRINT POINT%," ";
1401    TOSS%=TOSS%+1:CURSOR 47,1:PRINT TOSS%:CURSOR 28,2:RETURN
1500    PRINT "you win";:JF=1.0:WAIT TIME 200:RETURN
1600    PRINT "you lose";:JF=1.0:WAIT TIME 200:RETURN
*
```

```
*
R A N D O M L I N E S 3

=======================


5     COLORG 7 15 0 0
10    MODE 6
100   S%=X% MOD (XMAX):T%=Y% MOD (YMAX)
105   FOR A%=0 TO 60:X%=RND(XMAX):Y%=RND(YMAX)
110   DRAW S%,T% X%,Y% 15:DRAW S%,T% X%,Y% 0:S%=X%:T%=Y%
120   NEXT:WAIT TIME 100:GOTO 10
*
B U G

=====


5     MODE 5
10    X%=5:FOR Q%=YMAX-6 TO 0 STEP -1:X%=X%+1:GOSUB 100:NEXT
20    GOTO 5
100   DOT X%,Q% 15
110   DOT X%-1,Q%+1 13
120   DOT X%-2,Q%+2 11
130   DOT X%-3,Q%+3 8
140   DOT X%-4,Q%+4 6
150   DOT X%-5,Q%+5 3
160   DOT X%-6,Y%+6 1
170   RETURN


*
S O U N D S

===========


10    ENVELOPE 0 16:FOR A=0.0 TO 2.0:SOUND A 0 15 0 FREQ(33.0):NE
20    FOR A=5.0 TO 541.0 STEP A:GOSUB 100:NEXT
30    FOR Z=440.0 TO 33.0 STEP -(Z/100.0)
40    FOR G=0.0 TO 2.0:SOUND G 0 15 2 FREQ(Z+G)
50    NEXT G:WAIT TIME 5:NEXT Z:GOTO 10
100   Q=A MOD 3.0:R=(Q+1.0) MOD 3.0:S=(Q+2.0) MOD 3.0
110   SOUND Q 0 15 2 FREQ(A+32.0)
120   SOUND R 0 15 2 FREQ(A*A+32.0)
130   SOUND S 0 15 2 FREQ(A*A*A+32.0)
140   RETURN
```



```
*
C O L O R  G R A P H I C S

===========================


10    MODE 2:GOSUB 20:MODE 4:GOSUB 20:MODE 6:GOSUB 20:GOTO 10
20    FOR A%=0 TO YMAX:DRAW 0,0 XMAX,A% 20+(A% MOD 3):NEXT
30    FOR A%=0 TO XMAX-1:DRAW 0,0 A%,YMAX 20+(A% MOD 3):NEXT
40    FOR S%=0 TO 20:COLORG RND(15) RND(15) RND(15) RND(15)
50    WAIT TIME 20:NEXT S%:RETURN


*
G R A P H I C S  2

===================


10    MODE 2:GOSUB 20:MODE 4:GOSUB 20:MODE 6:GOSUB 20:GOTO 10
20    FOR A%=0 TO YMAX STEP 3:W%=W%+1:DRAW 0,0 XMAX,A% 20+(W% MOD 3):NEXT
30    FOR A%=0 TO XMAX-1 STEP 3:W%=W%+1:DRAW 0,0 A%,YMAX 20+(W% MOD 3):NE
40    FOR A%=1 TO XMAX STEP 3:W%=W%+1:DRAW A%,0 XMAX,YMAX 20+(W% MOD 3):N
50    FOR A%=1 TO YMAX STEP 3:W%=W%+1:DRAW 0,A% XMAX,YMAX 20+(W% MOD 3):N
60    FOR S%=0 TO 20:COLORG RND(15) RND(15) RND(15) RND(15)
70    WAIT TIME 20:NEXT S%:RETURN


*
R A N D O M  L I N E S

=======================


5     COLORG 7 15 0 0
10    MODE 4
100   S%=X% MOD (XMAX):T%=Y% MOD (YMAX)
105   FOR A%=0 TO 2:X%=RND(XMAX):Y%=RND(YMAX)
110   DRAW S%,T% X%,Y% 15:DRAW S%,T% X%,Y% 0:S%=X%:T%=Y%:NEXT:GOTO 10
```

```
5     ENVELOPE 0 15,2;10,2;15,2;10,2;0
6     ENVELOPE 1 15,5;12,5;10,100;0
10    REM music compose program
15    ENVELOPE 0 6
16    CLEAR 8000
17    DIM N$(50.0):DIM F%(50.0):DIM T(255.0):DIM E(255.0)
18    DIM V(255.0):DIM M(255.0):DIM D(255.0):DIM S(255.0)
20    DATA C0,65,C0+,69,D0,73,D0+,78,E0,82,F0,87,F0+,92,G0
21    DATA 98,G0+,104,A0,110,A0+,116,B0,123
30    DATA C,131,C+,138,D,147,D+,155,E,165,F,175,F+,185,G
31    DATA 196,G+,208,A,220,A+,233,B,247
40    DATA C1,262,C1+,277,D1,294,D1+,311,E1,330,F1,349,F1+
41    DATA 370,G1,392,G1+,415,A1,440,A1+,466,B1,494
50    DATA C2,523,C2+,554,D2,587,D2+,622,E2,659,F2,698,F2+
51    DATA 740,G2,784,G2+,831,A2,880,A2+,932,B2,988
60    FOR X=1.0 TO 48.0:READ N$(X):READ F%(X):NEXT
70    N$(0.0)="O":F%(0.0)=60000
75    N$(49.0)="C3":F%(49.0)=1046
90    PRINT CHR$(12)
100   REM compose
110   FOR X=1.0 TO 255.0
120   READ S(X):IF S(X)=999.0 THEN GOTO 190
125   READ E(X),NOTE$,V(X),D(X),M(X)
130   FOR V=0.0 TO 48.0
140   IF NOTE$=N$(V) THEN T(X)=F%(V):GOTO 180
150   NEXT V
180   NEXT
190   CURSOR 10,10
191   PRINT "from the motion picture ' THE STING '"
192   CURSOR 20,8:PRINT "THE ENTERTAINER "
194   CURSOR 30,6:PRINT "by SCOTT JOPLIN"
200   FOR P=1.0 TO X-1.0
210   SOUND S(P) E(P) V(P) M(P) FREQ(T(P))
211   WAIT TIME D(P)*5.0
220   NEXT
221   PRINT CHR$(12):SOUND OFF :WAIT TIME 10
225   CURSOR 10,10
226   PRINT "AFTER A BOTTLE OF WHISKY ......."
230   FOR P=1.0 TO X-1.0
240   SOUND S(P) E(P) V(P) M(P) FREQ(T(P)+RND(15.0))
241   WAIT TIME D(P)*5.0:NEXT
250   SOUND OFF :PRINT CHR$(12):POKE #7921,#56
251   CURSOR 2,10:PRINT "THANK YOU !"
300   DATA 0,1,D2,15,2,0,0,1,E2,15,2,0,0,1,C2,15,2,0
301   DATA 0,1,A1,15,4,0,0,1,B1,15,2,0,0,1,G1,15,4,0
302   DATA 2,1,D1,10,2,2,2,1,E1,10,2,0
303   DATA 2,1,C1,10,2,0,2,1,A,10,4,0,2,1,B,10,2,0
304   DATA 2,1,G,10,4,0
305   DATA 1,1,D,15,2,0,1,1,E,15,2,0,1,1,C,15,2,0
306   DATA 1,1,A0,15,4,0,1,1,B0,15,2,0,1,1,A0,15,2,0
307   DATA 1,1,G0+,15,2,0,1,1,G0,15,8,0
308   DATA 0,0,G,15,0,0,2,0,B,15,0,0,1,0,G1,15,4,0
309   DATA 0,0,O, 0,0,0,1,0,O,0;0,0,2,0,O,0,0,0
310   DATA 0, 0,D,10,2,0,0,0,D+,10,2,2,0,0,E,10,2,0
311   DATA 0,0,C1,10,5,0,0,0,E,10,2,0,0,0,C1,10,5,0
312   DATA 0,0,E,10,2,0,0,0,C1,10,8,0
313   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
314   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
315   DATA 0,0,D2+,12,0,0,2,0,F1+,12,2,0
```



```
316   DATA 0,0,E2,15,0,0,2,0,G1,15,2,0
317   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
318   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
319   DATA 0,0,E2,12,0,0,2,0,G1,12,4,0
320   DATA 0,0,B1,12,0,0,2,0,D1,12,2,0
321   DATA 0,0,D2,12,0,0,2,0,F1,12,4,0
322   DATA 0,0, C2,12,0,0,2,0,E1,12,8,0
323   DATA 2,0,O,0,0,0
324   DATA 0,0,D,12,2,0,0,0,D+,12,2,0
325   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
326   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
327   DATA 0,0,E,12,2,0,0,0,C1,12,10,0
328   DATA 0,0,A1,12,2,0,0,0,G1,12,2,0
329   DATA 0,0,F1+,12,0,0,2,0,C1,12,2,0
330   DATA 0,0,A1,12,2,0
331   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
332   DATA 0,0,E2,12,0,0,2,0,F1+,12,0,0,1,0,D0,12,3,0
333   DATA 0,0,D2,12,2,0,0,0,C2,12,2,0,0,0,A1,12,2,0
334   DATA 0,0,D2,12,0,0,2,0,F1,12,0,0,1,0,G0,12,8,0
335   DATA 0,0,O,0,0,0,1,0,O,0,0,0,2,0,O,0,0,0
336   DATA 0,0,D,12,2,0,0,0,D+,12,2,0
337   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
338   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
339   DATA 0,0,E,12,2,0,0,0,C1,12,8,0
340   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
341   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
342   DATA 0,0,D2+,12,0,0,2,0,F1+,12,2,0
343   DATA 0,0,E2,12,0,0,2,0,G1,12,2,0
344   DATA 0,0,C2,12,0,0,2,0,E2,12,2,0
345   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
346   DATA 0,0,E2,12,0,0,2,0,G1,12,3,0
347   DATA 0,0,B1,12,0,0,2,0,D1,12,2,0
348   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
349   DATA 0,0,C2,12,0,0,2,0,E1,12,4,0
350   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
351   DATA 00,0,D2,12,0,0,2,0,F1,12,2,0
352   DATA 1,1,C,15,0,0,0,0,E2,12,0,0,2,0,G1,12,2,0
353   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
354   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
355   DATA 1,1,A0+,15,0,0,0,0,E2,12,0,0,2,0,G1,12,3,0
356   DATA 0,0,C2,12,0,0,2,0,G1,12,2,0
357   DATA 0,0,D2,12,0,0,2,0,G1,12,2,0
358   DATA 0,0,C2,12,0,0,2,0,G1,12,2,0
359   DATA 1,1,A0,15,0,0,0,0,E2,12,0,0,2,0,A1,12,2,0
360   DATA 0,0,C2,12,0,0,2,0,C2,12,2,0
361   DATA 0,0,D2,12,0,0,2,0,A1,12,2,0
362   DATA 1,1,G0+,15,0,0,0,0,E2,12,0,0,2,0,G1+,12,3,0
363   DATA 0,0,C2,12,0,0,2,0,A1,12,2,0
364   DATA 0,0,D2,12,0,0,2,0,A1,12,2,0
365   DATA 0,0,C2,12,0,0,2,0,A1,12,2,0
366   DATA 1,1,G0,15,0,0,0,0,E2,12,0,0,2,0,G1,12,2,0
367   DATA 0,0,C2,12,0,0,2,0,E1,12,0,0
368   DATA 0,0,D2,1,0,0,2,0,F1,12,2,0
369   DATA 1,1,G0,15,0,0,0,0,E2,12,0,0,2,0,G1,12,3,0
370   DATA 0,0,B1,12,0,0,2,0,D1,12,2,0
371   DATA 0,0,D2,12,0,0,2,0,F1,12,4,0
372   DATA 1,1,C0,15,0,0,0,0,C2,12,0,0,2,0,E1,12,4,0
1000  DATA 999
```

N E   A R M   B A N D I T

========================

```
3    MODE 5A
2    COLORG 12 12 12 12
4    COLORT 12 0 0 0
9    CURSOR 0,3:PRINT "        pralines          PRESS ANY KEY        praline";
0    CURSOR 0,2:PRINT "   red   red   red   = 10      WIN          x     x     -
0    CURSOR 0,1:PRINT "    x     x     x    =  3                   -     x     x
0    CURSOR 28,1:PRINT "$";:CURSOR 28,1
00   Q%=64:GOSUB 1000
10   Q%=160:GOSUB 1000
20   Q%=256:GOSUB 1000
40   CURSOR 25,1:PRINT "         ";
41   CURSOR 28,1:PRINT "$";:CURSOR 28,1
42   A=GETC:IF A=0.0 GOTO 142
43   FOR Z=0.0 TO 15.0
44   Z1%=1+Z/6
45   ON Z1% GOTO 150,160,170
50   Q%=64:GOSUB 900
55   NOE=K
60   Q%=160:GOSUB 900
65   TWO=K
70   Q%=256:GOSUB 900
72   TRE=K
75   NEXT Z
78   GOSUB 1500
90   CURSOR 25,1:PRINT "pralines";:CURSOR 27,0:PRINT WINS%;" ";
82   WAIT TIME 100:GOTO 140
00   K=INT(RND(16.0))
10   IF K=8.0 GOTO 900
30   FILL Q%-8,90 Q%+7,130 K
70   RETURN
000  FILL Q%-32,42 Q%+31,170 0
001  FILL Q%-24,74 Q%+23,138 8
002  RETURN
500  IF NOE=3 AND TWO=3 AND TRE=3 THEN WINS%=10:RETURN
510  IF NOE=TWO AND NOE=TRE THEN WINS%=3:RETURN
520  IF NOE=TWO THEN WINS%=1:RETURN
530  IF TWO=TRE THEN WINS%=1:RETURN
540  WINS%=0:RETURN
```

D E S S I N S   A V E C   C U R S E U R

====================================

```
1      PRINT CHR$(12)
2      GOSUB 400
5      MODE 3
10     A=GETC
12     IF A=32.0 THEN 200
13     IF A=8.0 THEN 220
14     IF A=9.0 THEN 320
15     IF A<16.0 OR A>19.0 THEN 321
100    Y=Y+1.0:IF Y>YMAX THEN Y=YMAX
105    RETURN
110    Y=Y-1.0:IF Y<0.0 THEN Y=0.0
115    RETURN
120    X=X-1.0:IF X<0.0 THEN X=0.0
125    RETURN
130    X=X+1.0:IF X>XMAX THEN X=XMAX
135    RETURN
200    MODE 0:MODE 3:Y=0.0:X=0.0
210    GOTO 5
220    A=GETC:DOT X,Y 15
221    IF A=32.0 GOTO 200
222    IF A=9.0 GOTO 320
223    IF A<16.0 OR A>19.0 THEN 220
224    DOT X,Y 0:A=A-15.0:ON A GOSUB 100,110,120,130
225    GOTO 220
320    A=GETC:DOT X,Y 0
321    IF A=8.0 GOTO 220
322    IF A=32.0 GOTO 200
323    IF A<16.0 OR A>19.0 THEN 320
329    DOT X,Y 15:A=A-15.0:ON A GOSUB 100,110,120,130
330    GOTO 320
400    PRINT :PRINT
412    PRINT "LES DESSINS  S'OBTIENNENT  EN PRESSANT";
413    PRINT " UNE DES FLECHES":PRINT "          ";
430    PRINT "DANS LA DIRECTION QUI VOUS CONVIENT.":PRINT
432    PRINT " POUR EFFACER UN MORCEAU DE DESSIN ";
440    PRINT " REPLACEZ LE CURSEUR":PRINT "    ";
441    PRINT " A CET ENDROIT APRES AVOIR PRESSE";
442    PRINT " SUR CHAR DEL.":PRINT :PRINT "      ";
444    PRINT "POUR REPASSER EN MODE DESSIN";
445    PRINT " PRESSEZ SUR TAB":PRINT
470    PRINT "L'EFFACAGE  DE L'ECRAN  S'OBTIENT ";
480    PRINT " EN  PRESSANT LA BARRE"
481    PRINT "                D'ESPACEMENT"
490    PRINT :PRINT
491    INPUT "PRESSEZ LU ET RETURN APRES AVOIR FINI";Z$
492    IF LEFT$(Z$,1)="L" THEN 499
493    PRINT :GOTO 491
499    PRINT CHR$(12)
500    RETURN
```

GRAFTEXT SUBDEMO

```
1     CLEAR 1400
2     REM :DATA FOR GOSUB40040: X / Y / C / VFLAG / A$ / F
3     REM !!!! DELETE LINE 40 >>>>>> 70 !!!!!!!!!!!!!!!!!!!
5     COLORG 8 1 3 5
10    MODE 5
20    COLORG 8 0 14 1
30    GOSUB 40012:FOR X=0.0 TO XMAX:DOT X,225+20*SIN(X/20.0) 15:NEXT
31    FOR X=200.0 TO 230.0 STEP 3.0:DRAW X,10 X,45 0:NEXT
32    FOR Y=125.0 TO 150.0 STEP 2.0:FILL 260,Y XMAX,Y+1 Q:Q=Q+1.0:NEXT
33    X=10.0:Y=215.0:C=1.0:A$="DAI":VFLAG=0.0:F=2.0:GOSUB 40040
34    X=80.0:Y=215.0:C=6.0:A$="TEXT":GOSUB 40040
35    X=150.0:Y=215.0:C=5.0:A$="IN":GOSUB 40040
36    X=200.0:Y=215.0:C=0.0:F=2.0:A$="GRAFICS":GOSUB 40040
39    X=130.0:Y=190.0:C=2.0:F=1.0:A$="TEL. 02 / 3751114":GOSUB 40040
40    X=10.0:Y=200.0:C=0.0
41    A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ!#?$%&'()*=:-+;<>./1234567890"
50    GOSUB 40040
55    X=10.0:Y=170.0:C=3.0:F=2.0:GOSUB 40040
56    X=XMAX-10.0:Y=50.0:C=13.0:VFLAG=1.0:F=1.0:GOSUB 40040
60    VFLAG=0.0:X=10.0:Y=90.0:C=12.0:F=4.0:A$=LEFT$(A$,26):GOSUB 40040
65    GOTO 65
40012 DIM CAR$(90.0)
40021 FOR Z=32.0 TO 90.0:READ A$
40022 IF A$="STOP" THEN RETURN
40023 READ CAR$(Z):NEXT:RETURN
40040 X1=X:Y1=Y:IF F=0.0 THEN F=1.0
40041 FOR M=0.0 TO LEN(A$)-1.0
40042 T$=MID$(A$,M,1)
40050 GR$=CAR$(ASC(T$))
40060 FOR N=0.0 TO LEN(GR$)-1.0 STEP 4.0
40065 IF VFLAG=1.0 GOTO 40120
40070 IF MID$(GR$,N,1)="/" THEN X=X+(8.0*F):GOTO 40100
40080 ZZ=VAL(MID$(GR$,N,1)):YY=VAL(MID$(GR$,N+1,1))
40082 JC5%=X+ZZ*F:JC6%=Y+VAL(MID$(GR$,N+1,1))*F
40083 JC7%=X+VAL(MID$(GR$,N+2,1))*F:JC8%=Y+VAL(MID$(GR$,N+3,1))*F
40084 DRAW JC5%,JC6% JC7%,JC8% C
40085 IF F<1.5 THEN GOTO 40090
40086 JC9%=X+1+VAL(MID$(GR$,N+2,1))*F
40087 JC10%=Y+1+VAL(MID$(GR$,N+3,1))*F
40088 DRAW X+1+ZZ*F,Y+1+YY*F JC9%,JC10% C
40090 NEXT N
40100 IF X+8.0*F>=XMAX THEN X=X1:Y=Y-10.0*F
40102 NEXT M
40103 RETURN
40120 IF MID$(GR$,N,1)="/" THEN Y=Y-9.0*F:GOTO 40180
40130 JC1%=X+VAL(MID$(GR$,N+1,1))*F:JC2%=Y-VAL(MID$(GR$,N,1))*F
40131 JC3%=X+VAL(MID$(GR$,N+3,1))*F:JC4%=Y-VAL(MID$(GR$,N+2,1))*F
40132 DRAW JC1%,JC2% JC3%,JC4% C
40140 NEXT N
40180 IF Y-9.0*F<=0.0 THEN Y=Y1:X=X-9.0*F
40190 NEXT M
40200 RETURN
50000 DATA BLANCO,/,UITROEP!,31313337/,QOUTES,25274547/,#
50001 DATA 13531555212741 47/,$,12424253244415262656 3137/
50010 DATA %,1727162612564151 4252/,&,12132131533115511627 3536/,'
50011 DATA 3537/,(,131513311537/
50020 DATA ),315353555537/,*,125616523137/,+,32361454/,COMMA
50021 DATA 21323233/
```



```
50030 DATA -,1454/,.,31423241/,/,1256/,0,1216214152562747 1256/
50040 DATA 1,214131372637/,2,1151123344445556472716 27/,3
50041 DATA 1221214152533456175744 53/,4,414713531447/
50050 DATA 5,12212141525415451517 1757/,6,21411215144452531537 3757/
50051 DATA 7,212223561757/,8,2141244427471213151652535556/
50060 DATA 9,11313153535624541516 2747/,:,33333535/,;,213232333535/
50061 DATA <,14471441/
50070 DATA =,13531555/,>,21545427/,?,16272747343331313456/,APE,/
50080 DATA A,111551551353153737 55/,B,1117174714441141525 35556/,C
50081 DATA 121627474756214141 52/,D,111711415256 1747/
50090 DATA E,111711511444 1757/,F,111714441757/,G,1216275721515153534 3
50091 DATA H,111714545157/
50100 DATA I,214131372747/,J,122121415257/,K,111713572451/,L,11171151
50110 DATA M,11171735353435575751/,N,111751571652/,O,1216274756522141
50111 DATA 11171444174 75556/
50120 DATA Q,12162747565321313351/,R,11171747565514442451/,S
50121 DATA 122121415253244415162747 4756/,T,17573137/
50130 DATA U,111721415157/,V,131753571331 3153/,W,111751571133335133 34
50131 DATA X,111217165152575612 561652/
50140 DATA Y,161756571634 34563134/,Z,175712561151/
51000 DATA STOP
*
```

TRAFFICTEST


```
1      COLORG 8 1 3 5:MODE 5
2      ENVELOPE 1 15,10:0,10:
10     CLEAR 2000
30     GOSUB 40012
35     X=50.0:Y=230.0:C=14.0:F=1.5
36     A$="DAI TRAFFIC TEST":GOSUB 40040
110    DRAW 50,220 235,220 0
112    DRAW 0,170 280,170 0
115    P=170.0
120    READ A
125    IF A=999.0 THEN GOTO 140
130    READ B,C,D:DRAW A+50,B C+50,D 0:GOTO 120
140    A$="STOP FOR THE RED LIGHT":X=130.0:Y=80.0
141    C=3.0:F=1.0:GOSUB 40040
150    A$="NO REACTION ON GREEN !!":X=130.0:Y=60.0
151    C=5.0:F=1.0:GOSUB 40040
160    WAIT TIME 200:FILL 130,0 XMAX,100 8
200    REM TEST
210    C=INT(RND(2.0)):CO=3.0:IF C=1.0 THEN CO=5.0
215    SOUND 2 1 10 0 FREQ(800.0):WAIT TIME 20:SOUND OFF
220    WAIT TIME RND(50.0)
230    IF CO=3.0 THEN FILL 57,112 73,128 CO
235    IF CO=5 THEN FILL 57,87 73,103 5
237    IF CO=5 THEN GOTO 700
240    S=S+1.0:IF GETC=0.0 GOTO 240
250    FOR X=0.0 TO 250.0-S*2.0 STEP 3.0
251    FILL 300,X 310,X+1 1:SOUND 1 0 5 0 FREQ(31.0+X)
260    NEXT
265    SOUND OFF
270    MG=MG+10.0:NG=125.0+70.0-S/2.5
271    IF MG>280.0 THEN A$=" THE END":F=2.0:X=140.0:GOSUB 40040
272    IF MG>280.0 THEN WAIT TIME 1000:GOTO 1
275    IF NG<125.0 THEN NG=125.0
280    DRAW O,P MG,NG 15
290    O=MG:P=NG
295    S=S*1.5
300    IF S>=100.0 THEN A$=" WAKE UP !!          "
305    IF S>150.0 THEN A$=" YOU ARE SLOW !      "
310    IF S<100.0 THEN A$=" ATTENTION PLEASE !"
320    IF S<90.0 THEN A$=" NOT GOOD!           "
330    IF S<80.0 THEN A$=" MMMM...             "
340    IF S<70.0 THEN A$=" GOOD                "
350    IF S<60.0 THEN A$=" VERY GOOD!          "
360    IF S<50.0 THEN A$=" EXCELLENT !         "
370    IF S<40.0 THEN A$=" SUPERB !            "
380    IF S<30.0 THEN A$=" MARVELLOUS !        "
390    IF S<20.0 THEN A$=" GENIUS !            "
400    X=150.0:Y=50.0:C=3.0:F=1.0:GOSUB 40040
490    WAIT TIME 50
491    FILL 57,112 73,128 8:FILL 57,87 73,103 8
495    FILL 300,100 XMAX,YMAX 8
496    FILL 100,0 XMAX,100 8
```



```
506    S=0.0
510    GOTO 200
700    FOR X=0.0 TO 200.0:IF GETC<>0.0 THEN GOTO 710
705    NEXT:GOTO 490
710    FOR X=0.0 TO 10.0:SOUND 1 0 10 0 FREQ(1000.0)
711    SOUND 1 0 12 2 FREQ(500.0):WAIT TIME 10:NEXT
715    MG=MG+10.0:IF NG<125.0 THEN NG=125.0
716    DRAW O,P MG,NG 5:O=MG:P=NG
720    SOUND OFF :X=150.0:Y=80.0:C=5.0:F=1.5
721    A$="GREEN !":GOSUB 40040:GOTO 490
1000   GOTO 1000
40012 DIM CAR$(90.0)
40021 FOR Z=32.0 TO 90.0:READ A$
40022 IF A$="STOP" THEN RETURN
40023 READ CAR$(Z):NEXT:RETURN
40040 X1=X:IF F=0.0 THEN F=1.0
40041 FOR M=0.0 TO LEN(A$)-1.0
40042 T$=MID$(A$,M,1)
40050 GR$=CAR$(ASC(T$))
40060 FOR N=0.0 TO LEN(GR$)-1.0 STEP 4.0
40065 IF VFLAG=1.0 GOTO 40120
40070 IF MID$(GR$,N,1)="/" THEN X=X+(8.0*F):GOTO 40100
40080 JC1%=X+VAL(MID$(GR$,N,1))*F:JC2%=Y+VAL(MID$(GR$,N+1,1))*F
40081 JC3%=X+VAL(MID$(GR$,N+2,1))*F:JC4%=Y+VAL(MID$(GR$,N+3,1))*F
40082 DRAW JC1%,JC2% JC3%,JC4% C
40090 NEXT N
40100 IF X+8.0*F>=XMAX THEN X=X1:Y=Y-10.0*F
40102 NEXT M
40103 RETURN
40120 IF MID$(GR$,N,1)="/" THEN Y=Y-9.0*F:GOTO 40180
40130 JC5%=X+VAL(MID$(GR$,N+1,1))*F:JC6%=Y-VAL(MID$(GR$,N,1))*F
40131 JC7%=X+VAL(MID$(GR$,N+3,1))*F:JC8%=Y-VAL(MID$(GR$,N+2,1))*F
40132 DRAW JC5%,JC6% JC7%,JC8% C
40140 NEXT N
40180 IF Y-9.0*F<=0.0 THEN Y=Y1:X=X-9.0*F
40190 NEXT M
40200 RETURN
50000 DATA BLANCO,/,UITROEP!,31313337/,QOUTES,25274547/,#
50001 DATA 1353155521274147/,$,124242532444152626563137/
50010 DATA %,17271626125641514252/,&,121321315331155116273536/,'
50011 DATA 3537/,(,131513311537/
50020 DATA ),315353555537/,*,125616523137/,+,32361454/,COMMA,21323233
50030 DATA -,1454/,.,31423241/,/,1256/,0,12162141525627471256/
50040 DATA 1,214131372637/,2,115112334444555647271627/,3
50041 DATA 122121415253345617574453/,4,414713531447/
50050 DATA 5,122121415254154515171757/,6,214112151444525315373757/,7
50051 DATA 212223561757/,8,2141244427471213151652535556/
50060 DATA 9,113131535356245415162747/,:,33333535/,;,213232333535/,<
50061 DATA 14471441/
50070 DATA =,13531555/,>,21545427/,?,16272747343331313456/,APE,/
50080 DATA A,11155155135315373755/,B,111717471444114152535556/,C
50081 DATA 121627474 5621414152/,D,1117114152561747/
50090 DATA E,1117115114441757/,F,111714441757/,G,12162757215151535343
50091 DATA 111714545157/
50100 DATA I,214131372747/,J,122121415257/,K,111713572451/,L,1117115
50110 DATA M,11171735353435575751/,N,111751571652/,O,121627475652214
50111 DATA 1117144417475556/
50120 DATA Q,12162747565321313351/,R,11171747565514442451/,S
50121 DATA 1221214152532444151627474756/,T 17573137/
50130 DATA U,111721415157/,V,131753571333 53/,W,11175157113333513334
50131 DATA 111217165152575612561652/
50140 DATA Y,16175657163434563134/,Z,175712561151/
```

```
51140 DATA 10,0,10,80,20,0,20,80,25,80,30,85,30,85,30,135,30
51141 DATA 135,25,140,25,140,5,140,5 ,140,0,135,0,135,0,85
51150 DATA 0,85,5,80,999
*
```


==

```
1     GOTO 20
7     GOTO 64000
8     GOTO 64000
9     GOTO 64000
10    GOTO 64000
20    COLORT 8 0 0 8
21    POKE #131,1
22    PRINT CHR$(12)
23    CURSOR 1,20:PRINT "1  CHANGE BACKGROUND COLOUR"
24    CURSOR 31,20:PRINT "6  ANIMATION  / COLORT "
25    CURSOR 1,18:PRINT "2  FLASHING BACKGROUND"
26    CURSOR 31,18:PRINT "7  ..............  "
27    CURSOR 1,16:PRINT "3  SCREEN LINE ADDRESS"
28    CURSOR 31,16:PRINT "8  ..............  "
29    CURSOR 1,14:PRINT "4  SCREEN CURSOR ADDRESS"
30    CURSOR 31,14:PRINT "9  ..............  "
31    CURSOR 1,12:PRINT "5  ANIMATION, COLOURS 1619"
32    CURSOR 30,12:PRINT "10  ..............  "
40    CURSOR 30,2:INPUT "WICH PROGRAM ";P$:PRINT
41    IF P$="1" OR P$="2" OR P$="3" OR P$="4" THEN 46
42    IF P$="5" OR P$="6" THEN 46
43    IF P$="7" OR P$="8" OR P$="9" OR P$="10" THEN 64000
44    CURSOR 1,4:PRINT "WRONG INPUT ONLY THE NUMBER OF THE PROGRAM
45    CURSOR 30,2:PRINT "WICH PROGRAM                    ":GOTO 40
46    P=VAL(P$)
47    ON P GOTO 100,1000,2000,3000,4000,10000,7,8,9,10
100   PRINT CHR$(12):PRINT :PRINT :PRINT
108   LIST 110-170
110   E%=#FF
115   COLORT 0 9 9 0
120   B%=#7FEF
125   FOR A%=0 TO 23
130   D%=B%-3
135   FOR C%=0 TO 65
140   POKE D%,E%
145   D%=D%-2:NEXT
146   RJ%=GETC:IF RJ%=32 GOTO 20
155   B%=B%-#86:NEXT
165   E%= INOT E% IAND #FF
170   GOTO 120
1000  PRINT CHR$(12):A5%=0
1010  FOR A%=0 TO 10
1020  POKE #79E4+2*A%,#FF
1025  POKE #79E4+2*A%+#86,#FF
1030  NEXT
1035  CURSOR 23,12:PRINT "WARNING"
1040  FOR B%=20 TO 1 STEP -1
1043  GOSUB 1200
1045  COLORT 0 9 A5% 15-A5%
1046  GOSUB 1100
1050  WAIT TIME B%
1055  COLORT 0 9 15-A5% A5%
1056  GOSUB 1100
1060  WAIT TIME B%
1065  NEXT
1070  GOTO 1040
1100  RJ%=GETC:IF RJ%<>32 THEN RETURN
```

```
1130  PRINT :INPUT "LIST PROGRAM < Y/N > ";RJ$
1140  IF RJ$="Y" THEN PRINT CHR$(12):GOSUB 64500:GOTO 20
1141  IF RJ$="N" THEN PRINT CHR$(12):PRINT :GOTO 20
1145  CURSOR 0,10:PRINT SPC(30):CURSOR 0,11
1150  RETURN
1200  A5%=A5%+1:IF A5%>15 THEN A5%=0
1210  RETURN
2000  GOSUB 2100
2020  FOR A%=0 TO 23
2035  PRINT 23.0-A%:SPC(9-CURX);"# ";HEX$(#7FEA-(#86*A%));
2036  PRINT SPC(22-CURX);"# ";HEX$(#7FED-(#86*A%));SPC(37-CURX);
2040  PRINT "# ";HEX$(#7F6A-(#86*A%));
2041  PRINT SPC(52-CURX);"# "+HEX$(#7F6D-(#86*A%))
2045  IF A%=11 THEN GOSUB 2150:GOSUB 2100
2050  NEXT:PRINT :GOSUB 2150:GOTO 20
2100  PRINT CHR$(12):PRINT
2105  PRINT "         # LOCATION                      # LOCATION"
2110  PRINT "LINE     COLOR CODE    # LOCATION";
2111  PRINT "     COLOR CODE    # LOCATION"
2120  PRINT "NUMBER   BEGIN LINE    BEGIN LINE";
2121  PRINT "     END LINE      END LINE"
2125  PRINT
2130  RETURN
2150  RJ%=GETC:IF RJ%<>32 GOTO 2150
2160  RETURN
3000  PRINT CHR$(12):PRINT :PRINT "CHARACTERS FROM  <-2 TO 61 > "
3002  PRINT "LINES FROM        < 0 TO 23 > ":PRINT
3003  PRINT "INPUT CURSOR EXAMPLE 31,12 FOR CENTER OF SCREEN":PRINT
3004  INPUT "INPUT CURSOR ";B1%,A1%:PRINT :PRINT
3005  IF A1%<0.0 OR B1%>61.0 OR A1%>23.0 THEN PRINT "WRONG INPUT":PRINT :GOTC
3009  B1%=B1%+3
3010  PRINT "POKE # ";HEX$((#7FEA-(#86*(23-A1%)))-((B1%*2)));" TO CHANGE COLC
3020  PRINT "POKE # ";HEX$((#7FED-(#86*(23-A1%)))-((B1%*2)));" TO CHANGE [illegible]AR
3030  PRINT :PRINT
3035  PRINT "FOR OTHERS PRESS RETURN ,FOR OTHER PROGRAMS SPACE BAR"
3040  RJ%=GETC:IF RJ%=32 GOTO 20
3045  IF RJ%=0 GOTO 3040
3050  GOTO 3004
4000  MODE 4
4110  FOR B=0.0 TO 2.0*PI STEP 0.2
4120  A=B-0.2:B%=16:GOSUB 4220
4130  A=B:B%=17:GOSUB 4220
4140  COLORG 0 10 0 10
4150  A=B-0.1:B%=18:GOSUB 4220
4160  A=B+0.1:B%=19:GOSUB 4220
4170  COLORG 0 0 10 10
4190  NEXT
4190  A=B-0.2:B%=16:GOSUB 4220
4200  A=B-0.1:B%=18:GOSUB 4220
4210  GOTO 4110
4220  X%=XMAX/2+30*SIN(A)
4230  Y%=YMAX/2+30*COS(A)
4240  DRAW XMAX/2,YMAX/2 X%,Y% B%
4245  RJ%=GETC:IF RJ%=32.0 THEN MODE 0:GOTO 20
4250  RETURN
10000 MODE 0:COLORT 8 0 0 8
10010 PRINT CHR$(12.0)
10020 A%=#7A23-2:B%=#79A8+2
10030 FOR C%=A% TO B% STEP -2
10040 POKE C%,#FF
```



```
10041 REM POKE C-2,#FF
10042 WAIT TIME 1:POKE C%+2,#0
10050 NEXT:POKE C%,#0
10060 FOR C%=B% TO A% STEP 2
10070 POKE C%,#FF:POKE C%-2,#0
10080 NEXT:POKE C%,#0
10090 JCC%=GETC:IF JCC%>0 GOTO 1
10100 GOTO 10030
64000 P%=P
64005 CURSOR 1,4:PRINT "
64006 PRINT "                              "
64010 CURSOR 1,4:PRINT "NO PROGRAM IN";P%
64020 GOTO 45
64500 PRINT :LIST 1000-1070:GOSUB 2150:RETURN
*
```

B I O R Y T H M

================

```
90    CLEAR 1000
95    PRINT CHR$(12)
100   DIM X$(31.0):DIM M$(12.0)
110   M$(1.0)="JAN"
111   M$(2.0)="FEB"
112   M$(3.0)="MAR"
113   M$(4.0)="APR"
114   M$(5.0)="MAI"
115   M$(6.0)="JUN"
116   M$(7.0)="JUL"
117   M$(8.0)="AUG"
118   M$(9.0)="SEP"
119   M$(11.0)="NOV"
120   M$(12.0)="DEC"
121   M$(10.0)="OCT"
200   P9=6.28318
210   P1=23.0:P2=28.0:P3=33.0
220   D1=P9/P1:D2=P9/P2:D3=P9/P3
230   DATA 31,28,31,30,31,30,31,31,30,31,30,31
300   INPUT "YOUR NAME PLEASE ";N$
311   PRINT
312   PRINT "BIORYTHM OF YEAR OR MONTH ";
313   INPUT X$
320   IF X$<>"YEAR" AND X$<>"MONTH" THEN GOTO 311
330   N1=0.0
340   GOSUB 8000
360   IF B1>2.0 THEN GOTO 400
370   IF B1=2.0 THEN IF B2=29.0 THEN GOTO 400
380   R=(B3-1900.0)/4.0
381   IF INT(R)<>R THEN GOTO 400
390   N1=1.0
400   GOSUB 8500
420   FOR J=1.0 TO B1
430   READ X
440   NEXT J
450   N1=N1+X-B2
460   IF B1=12.0 THEN GOTO 510
470   FOR J=B1+1.0 TO 12.0
480   READ X
490   N1=N1+X
500   NEXT J
510   IF C3-B3<2.0 THEN GOTO 560
520   FOR J=B3-1899.0 TO C3-1901.0
530   IF INT(J/4.0)=J/4.0 THEN N1=N1+1.0
540   N1=N1+365.0
550   NEXT J
560   RESTORE
570   IF C1=1.0 THEN GOTO 620
580   FOR J=1.0 TO C1-1.0
590   READ X
600   N1=N1+X
610   NEXT J
620   T=(C3-1900.0)/4.0
621   IF INT(T)<>T THEN GOTO 640
630   IF C1>2.0 THEN N1=N1+1.0
640   I1=N1:I2=N1:I3=N1
```



```
650   READ X
655   PRINT CHR$(12)
660   PRINT "  BIORYTHMIC CHART  ";N$
665   PRINT :PRINT
667   B2%=B2:B1%=B1:B3%=B3
670   PRINT "DATE OF BIRTH";B2%;" ";B1%;" ";B3%
680   PRINT :PRINT :PRINT
690   PRINT "I=INTELLIGENCE"
700   PRINT "P=PHYSICAL"
710   PRINT "E=EMOTIONNAL"
720   L=0.0
730   GOSUB 2000
740   D=0.0
745   L=L+1.0
750   FOR I=1.0 TO 31.0
760   X$(I)=" "
770   NEXT I
780   X$(16.0)=":"
800   Y1=INT(15.0*SIN((L+I1)*D1)+16.5)
810   Y2=INT(15.0*SIN((L+I2)*D2)+16.5)
820   Y3=INT(15.0*SIN((L+I3)*D3)+16.5)
830   X$(Y1)="P"
840   X$(Y2)="E"
850   X$(Y3)="I"
860   IF Y1=Y2 THEN X$(Y1)="*"
870   IF Y2=Y3 THEN X$(Y3)="*"
880   IF Y1=Y3 THEN X$(Y1)="*"
890   D=D+1.0
900   IF D<X+1.0 THEN GOTO 1020
910   S1=S1+1.0
920   IF S1=12.0 THEN GOTO 1500
930   C1=C1+1.0
940   IF C1>12.0 THEN GOTO 980
950   READ X
955   IF X9=1.0 THEN GOTO 1500
960   GOSUB 3000
970   GOTO 1020
980   RESTORE
990   C1=1.0
1000  C3=C3+1.0
1010  GOTO 950
1020  D%=D
1021  IF D<10.0 THEN 1023
1022  PRINT M$(C1);" ";D%;"    ";:GOTO 1025
1023  PRINT M$(C1);"  ";D%;"    ";
1025  Y$=" "
1030  FOR J=1.0 TO 31.0
1050  Y$=Y$+X$(J)
1055  NEXT J
```

```
1056  PRINT Y$
1060  GOTO 745
1500  STOP
2000  IF X$="MONTH" THEN X9=1.0
2020  PRINT :PRINT " BIORYTHMIC CHART OF ";N$;:C3%=C3
2022  PRINT " FOR ";M$(C1);"  ";C3%
2030  PRINT
2040  PRINT "                    ";"(-)";
2045  PRINT "                    ";"(+)"
2050  PRINT
2060  D=1.0
2070  RETURN
3000  IF X$="MONTH" THEN X9=1.0
3002  PRINT
3004  D=1.0
3010  RETURN
8000  PRINT :PRINT "MONTH,DAY,YEAR OF BIRTH"
8002  PRINT "EXAMPLE   BIRTH ON 3D MAY 1942"
8003  PRINT "PRESS 5  RETURN 3  RETURN 1942"
8015  INPUT B1,B2,B3
8020  RETURN
8500  PRINT
8501  PRINT " GIVE MONTH OND YEAR FOR THE BIORYTHM"
8502  PRINT "EX FOR AND STARTING ON JANUARY    1980"
8503  PRINT "PRESS 1 RETURN 1980 RETURN"
8508  INPUT C1,C3
8510  IF B3>=C3 THEN GOTO 90
8520  RETURN
*
```

SIMON


```
1     MODE 3A:BST=0.0:CNT=0.0
2     CURSOR 0,3:PRINT "        LAST PLAY";
3     CURSOR 40,3:PRINT "BEST RESULT";
4     GOSUB 5000
10    REM CLEAR 1000
15    ENVELOPE 0 3,10;3,10;3,10;0
20    DIM A(4.0):DIM B(4.0)
30    A(1.0)=40.0:B(1.0)=40.0:A(2.0)=70.0
35    B(2.0)=70.0:A(3.0)=100.0:B(3.0)=40.0
37    A(4.0)=70.0:B(4.0)=10.0
40    DIM TUNE(100.0)
70    DIM NOTE(4.0)
80    NOTE(4.0)=262.0:NOTE(1.0)=330.0:NOTE(3.0)=392.0:NOTE(2.0)=523
100   DIM COLOR(4.0)
110   COLOR(1.0)=1.0:COLOR(2.0)=5.0:COLOR(3.0)=7.0:COLOR(4.0)=11.0
450   CNT=0.0
480   CNT=CNT+1.0
490   TUNE(CNT)=INT(RND(4.0))+1.0
500   WAIT TIME 30
520   FOR I=1.0 TO CNT
530   PLAY=TUNE(I)
540   GOSUB 2000
560   NEXT I
590   I=0.0
600   I=I+1.0
610   IF I<=CNT THEN 635
620   GOTO 480
635   GOSUB 6000
640   GOSUB 2000
645   IF BST<CNT THEN BST=CNT
650   IF PLAY=TUNE(I) THEN 600
670   GOSUB 5000
760   CURSOR 22,2:PRINT "PLAY BROKEN";:WAIT TIME 75
761   CURSOR 22,2:PRINT "                    ";:CURSOR 44,2
770   IF BST>CNT THEN GOSUB 5010
771   GOTO 10
2000  SOUND 0 0 10 0 FREQ(NOTE(PLAY))
2020  SOUND 2 0 10 2 FREQ(NOTE(PLAY)*4.0)
2040  FILL A(PLAY),B(PLAY) A(PLAY)+20.0,B(PLAY)+20.0 COLOR(PLAY)
3000  WAIT TIME 20
3050  SOUND OFF
4040  FILL A(PLAY),B(PLAY) A(PLAY)+20.0,B(PLAY)+20.0 0
4100  RETURN
5000  CURSOR 10,2:CNT%=CNT:PRINT CNT%;:PRINT "   ";
5010  CURSOR 44,2:BST%=BST:PRINT BST%;:PRINT "
5015  CURSOR 44,2
5020. RETURN
6000  WAIT TIME 5:G=GETC:IF G=0.0 GOTO 6000
6050  IF G=18.0 THEN PLAY=1.0
6060  IF G=16.0 THEN PLAY=2.0
6070  IF G=19.0 THEN PLAY=3.0
6080  IF G=17.0 THEN PLAY=4.0
6120  RETURN
```

*LOAD

PADDLE SOUND

```
1     REM MAKE SOUND WITH BOTH PADDLES
5     ENVELOPE 0 16
10    P=PDL(0):Q=PDL(2):R=PDL(3)
30    IF P>3.0 OR Q>31.0 THEN SOUND 1 0 R*3/52 0 FREQ(P*12.0+Q)
40    S=PDL(1):T=PDL(4):U=PDL(5)
50    IF S>3.0 OR T>31 THEN SOUND 2 0 U*3/52 0 FREQ(S*12.0+T)
90    GOTO 10
```

*

RANDOM POS TEST

```
1     MODE 0
2     COLORG 7 0 15 4
4     INPUT "TYPE H OR S , FOR HARDWARE OR SOFTWARE":RNT$
6     W%=1
7     MODE 4
10    DIM A%(XMAX)
15    IF RNT$="S" THEN K=RND(XMAX+1.0):GOTO 21
16    IF RNT$="H" THEN K=RND(0.0)*(XMAX+1.0):GOTO 21
20    GOTO 4
21    R=R+K
22    S%=S%+1
30    A%(K)=A%(K)+1.0
40    O%=A%(K)
50    P%=O%/W%
60    IF P%*W%<>O% THEN 20
69    IF P%>=YMAX+1 THEN DOT XMAX,0 14:GOTO 69
70    DOT K,P% 15
75    DOT T%,0 7
90    T%=(R/S%-((XMAX+1)*0.495))*100
91    IF T%<0 THEN T%=0
92    IF T%>XMAX THEN T%=XMAX
93    DOT T%,0 0
999   GOTO 15
```


LANDSCAPE V2

```
5     ENVELOPE 0 5,10:2,5:4,15:0
6     ENVELOPE 1 10,5:15,2:5,3:0
10    MODE 5:FLAG9%=0
20    FILL 0,0 XMAX,50 5
30    FILL 0,50 XMAX,YMAX 12
50    DRAW 0,0 150,50 0
60    DRAW 150,50 XMAX,0 0
70    FOR X=0.0 TO 2.0*PI STEP 0.1
80    DRAW 250,150 250+30*COS(X),150+30*SIN(X) 14
90    NEXT
95    GOSUB 1000
165   NOISE 1 15
166   WAIT TIME 3
170   FILL A,50 A+10,60 0
180   FILL A,50 A+1,60 12
185   NOISE 1 15
190   FILL A+10,50 A+11,60 0
195   IF A>50.0 GOTO 210
200   A=A+1.0:GOTO 165
210   FOR X=0.0 TO PI STEP 5E-2
220   DOT 150+50*COS(X),50+50*SIN(X) 0
225   SOUND 1 0 10 0 FREQ(X*100.0+31.0)
230   NEXT
240   A=150.0:B=150.0:C=50.0
250   FILL A,50 B,C 11
260   A=A-1.0:B=B+1.0:C=C+1.0
270   IF A<120.0 GOTO 300
280   GOTO 250
300   SOUND 1 0 15 2 FREQ(2000.0)
310   WAIT TIME 5
320   SOUND 1 0 10 2 FREQ(31.0)
325   NOISE 1 15
330   WAIT TIME 1
340   SOUND 1 0 15 2 FREQ(330.0)
350   SOUND 0 0 15 2 FREQ(440.0)
355   SOUND 2 0 15 2 FREQ(523.0)
360   WAIT TIME 100
370   SOUND 0 0 15 2 FREQ(370.0)
380   WAIT TIME 100
390   SOUND 0 0 15 2 FREQ(415.0)
400   SOUND 2 0 15 2 FREQ(494.0)
450   WAIT TIME 50
500   SOUND 1 0 15 2 FREQ(1318.0)
515   WAIT TIME 100
516   SOUND OFF
520   SOUND 1 0 10 0 FREQ(247.0)
530   WAIT TIME 13
```

```
540   SOUND 1 0 10 0 FREQ(277.0)
550   WAIT TIME 20
560   SOUND 1 0 10 0 FREQ(247.0)
570   WAIT TIME 13
590   SOUND 1 0 10 0 FREQ(208.0)
595   SOUND 1 0 5 0 FREQ(165.0)
600   WAIT TIME 20:SOUND OFF
610   FOR X=0.0 TO 200.0
620   DOT RND(XMAX),(50+RND(YMAX-50.0)) 15
621   NOISE 0 10
625   SOUND 1 0 1 0 FREQ(RND(1000.0)+31.0):WAIT TIME 1:SOUND OFF
626   NOISE OFF
630   NEXT
650   FLAG9%=1
1000  FOR X=0.0 TO 100.0
1100  DRAW 50+A,100 55+A,95 0
1110  DRAW 55+A,95 60+A,100 0
1120  DRAW 50+A,100 55+A,95 12
1130  DRAW 55+A,95 60+A,100 12
1140  DRAW 50+A,95 60+A,95 0
1150  DRAW 50+A,95 60+A,95 12:A=RND(50.0)
1155  SOUND 1 0 3 3 FREQ(3000.0+RND(1000.0))
1156  WAIT TIME 1:SOUND OFF
1160  NEXT X
1170  IF FLAG9%=1 GOTO 1000
1200  RETURN
*
```

POLYGONS

```
1     CLEAR 5000
5     INPUT "How many sides ";N
6     PRINT :INPUT "Radius  (between 4 and 120) ";R
10    MODE 5
50    DIM B(N),C(N)
90    P1=2.0*PI/N
100   FOR I=1.0 TO N
110   B(I)=R+10.0+R*COS((I-1.0)*P1)
120   C(I)=R+10.0+R*SIN((I-1.0)*P1)
130   NEXT I
140   FOR J=1.0 TO N
150   FOR I=1.0 TO N
160   DRAW B(J),C(J) B(I),C(I) 15
170   NEXT I:NEXT J
180   WAIT TIME 100:GOTO 5
*
```


MUSIC V2

```
5     DIM F(20.0)
6     ENVELOPE 0 15,3:7,5:3,10:0
10    FOR N=1.0 TO 17.0:READ F(N):NEXT
15    FOR JCC%=1 TO 27
20    READ N,L
30    A=F(N):GOSUB 100:WAIT TIME L
35    NEXT
41    RESTORE:GOTO 10
100   SOUND 0 0 15 0 FREQ(A)
200   SOUND 1 0 15 0 FREQ(A*2.0)
300   SOUND 2 0 10 0 FREQ(A*4.0)
301   RETURN
1000  DATA 262,277,294,311,330,349,370,392,415,440,466
1005  DATA 494,523,554,587,622,659
1010  DATA 1,5,5,5,8,5,13,10,12,5,13,5,15,5,17,10,13,5
1020  DATA 8,5,5,5,1,10,17,10,13,10,8,10,5,10,1,10,1,1
1030  DATA 4,1,10,1,14,1,1,2,3,4,5,6,7,8,9,10,5,13,8
*
```

VIENNA V2

```
2     ENVELOPE 0 1,5:2,5:3,5:0
3     ENVELOPE 1 5,3:3,7:1,3:1
5     DIM F(20.0)
10    FOR N=1.0 TO 17.0:READ F(N):NEXT
15    DATA 262,277,294,311,330,349,370,392
16    DATA 415,440,466,494,523,554,587,622,659
17    FOR JCC%=1 TO 18
20    READ O,E,V,M,N,L
40    SOUND O E V M FREQ(F(N)):WAIT TIME L
45    NEXT
50    RESTORE:GOTO 10
100   DATA 0,0, 5,0, 7,0,1,0, 5,0, 4,50
110   DATA 0,0, 7,2, 8,0,1,0, 7,2, 5,20
120   DATA 0,0,10,2,17,0,1,0,10,2,13,80
130   DATA 0,0, 5,0,12,0,1,0, 5,0, 9,20
140   DATA 0,0, 7,0,13,0,1,0, 7,0,10,10
150   DATA 0,0,10,0,13,0,1,0 ,7,0,10,80
160   DATA 0,0,10,0,12,0,1,0,10,0, 9,20
170   DATA 0,0,12,0,13,0,1,0,12,0,10,10
180   DATA 0,0,15,0, 8,0,1,1,15,2, 5,30
```

* * * * * * M U S I C   T U T O R * * * * * *

====================

THIS PROGRAM GENERATES MUSIC AND DISPLAYS THE NOTES.
IF YOU ANSWER YES BY TYPING Y TO THE FIRST QUESTION,
THE ONLY KEYS YOU CAN PRESS ARE THE A TO F (DO TO SI)
AND IF YOU ANSWER NO BY TYPING N ALL ALPHABETIC KEYS
ARE GIVING A NOTE.   YOU CAN ALSO DISPLAY THE NOTES
LARGE OR SMALL SCALE BY TYPING L OR S TO THE QUESTION
BUT YOU NEED A 48K RAM FOR THE SMALL SCALE.

THE NUMERIC KEYS HAVE THE FOLLOWING FUNCTIONS:

1= NORMAL NOTES
2= TREMOLO
3= GLISSANDO
4= GLISSANDO+TREMOLO
5= SHORT NOTES
7= START RECORDING UP TO 2000 NOTES
8= ENDS RECORDING AND REPLAYS EACH TIME YOU PRESS IT
9= SCROLLS PAGE
10=CLEARS PAGE
SHIFT+ALPHA KEY=INVERT NOTES
TAB KEY RESTART THE PROGRAM

```
1      CLEAR 10000:LIMIT%=10:DIM ARRAY%(LIMIT%,200.0)
2      PAGE%=0:POINTER%=0:RECORD%=0:PLAYBACK%=0:TUTOR%=0:ACCENT%=0
3      PRINT CHR$(12):PRINT :PRINT :PRINT "TUTOR MODE  YES OR NO  < Y / N >"
4      ANS%=GETC:IF ANS%=0 GOTO 4
5      IF ANS%=ASC("Y") THEN TUTOR%=1:GOTO 7
6      IF ANS%<>ASC("N") GOTO 1
7      PRINT :PRINT "SIZE - LARGE OR SMALL. < L / S >"
8      ANS%=GETC:IF ANS%=0 GOTO 8
9      IF ANS%=ASC("L") THEN MODE 3:GOTO 15
10     IF ANS%=ASC("S") THEN MODE 5:GOTO 15
11     PRINT "ANSWER ONLY WITH ''S'' OR ''L''":GOTO 7
15     ENVELOPE 0 15,100;8,75;3,50;0:ENVELOPE 1 15,3;10,2;0:STYLE%=0
17     RESTORE:DIM NOTE(21.0,2.0),COMP%(21.0,1.0),SPOT%(21.0)
18     FOR I%=1 TO 13:FOR J%=0 TO 1:READ COMP%(I%,J%):NEXT J%
19     NOTE(I%,0.0)=FREQ(267.0*(2.0^(I%/12.0)))
21     NOTE(I%,1.0)=2.0*NOTE(I%,0.0):NOTE(I%,2.0)=NOTE(I%,0.0)/2.0:NEXT I%
22     FOR I%=14 TO 21:FOR J%=0 TO 1:READ COMP%(I%,J%):NEXT J%:FOR J%=0 TO 2
23     READ CHORD%:NOTE(I%,J%)=NOTE(CHORD%,0.0):NEXT J%:NEXT I%
24     FOR I%=1 TO 21:READ SPOT%(I%):NEXT I%
25     GOSUB 1500
28     FOR TIMER%=1 TO 100-99*ACCENT%
30     GOSUB 10000:IF KEY%=0.0 THEN NEXT TIMER%:SOUND OFF :GOTO 28
31     IF KEY%=53.0 THEN ACCENT%=0:GOTO 30
32     IF KEY%=54 THEN ACCENT%=1:GOTO 30
33     IF KEY%=48 THEN GOSUB 2000:GOTO 30
34     IF (KEY%=57) OR (WHERE=(-1)) THEN OFFSET=OFFSET-75.0:GOSUB 2010:GOTO 3
35     IF KEY%=9.0 THEN SOUND OFF :MODE 0:GOTO 3
36     IF (KEY%>48.0) AND (KEY%<53.0) THEN STYLE%=KEY%-49:GOTO 30
37     OCTAVE%=1:IF (KEY%>96) OR (KEY%=60) THEN OCTAVE%=2:GOSUB 3000
```



```
38     FOR J%=1+13*TUTOR%*(1-ACCENT%) TO 21
39     IF KEY%<>COMP%(J%,TUTOR%) THEN NEXT J%:GOSUB 3500:GOTO 28
40     FOR I%=0 TO 2
41     SOUND I% ACCENT% 15-10*SGN(I%) STYLE% NOTE(J%,I%)/OCTAVE%:NEXT I%
42     IF (SPOT%(J%)=100.0) OR (WHERE=(-1.0)) OR (OFFSET<0.0) GOTO 100
48     GOSUB 4000
50     FILL AA,BB CC,DD EE
55     DRAW FF,GG HH,II JJ
60     WHERE=WHERE+10.0:IF WHERE>XMAX-10.0 THEN WHERE=-1.0
100    GOTO 28
1000   DATA 90,67,83,67,88,68,68,67,67,69,86,70,71,67,66,71,72,67,78,65
1010   DATA 74,67,77,66,44,99,87,67,1,5,8,69,68,3,8,1,82,69,5,1,8,84,70
1015   DATA 6,10,13,89,,71,8,1,5,85,65,10,1,6,73,66,12,3,8,79,99,13,5,8
1020   DATA -10,100,-5,100,0,5,100,10,100,15,100,20,25,-10,-5,0,5,10,15,20,
1500   OFFSET=YMAX-62.0:GOTO 2020
2000   FILL 0,0 XMAX,YMAX 0:GOTO 1500
2010   IF OFFSET<0 GOTO 1500
2020   WHERE=5.0
2030   FILL 0,OFFSET-12 XMAX,OFFSET+62 0
2040   FOR Z%=OFFSET TO OFFSET+40 STEP 10
2050   DRAW 0,Z% XMAX,Z% 12:NEXT Z%:RETURN
3000   KEY%=KEY%-32:IF KEY%=28 THEN KEY%=44
3010   RETURN
3500   TIMER%=TIMER%+1:NEXT TIMER%:SOUND OFF
3510   RETURN
4000   AA=WHERE-2.0:BB=OFFSET+(OCTAVE%-1.0)*35.0+SPOT%(J%)-2.0
4010   CC=WHERE+2.0:DD=OFFSET+(OCTAVE%-1.0)*35.0+SPOT%(J%)+2.0
4020   EE=SPOT%(J%)/5.0+8.0
4030   FF=WHERE+6.0-4.0*OCTAVE%:GG=OFFSET+SPOT%(J%)+(OCTAVE%-1.0)*35.0
4040   HH=WHERE+6.0-4.0*OCTAVE%:II=OFFSET+SPOT%(J%)+20.0:JJ=SPOT%(J%)/5.0+
4050   RETURN
5000   IF KEY%=56 THEN RECORD%=0:ARRAY%(PAGE%,POINTER%)=128
5010   RETURN
6000   IF POINTER%=200 THEN POINTER%=0:PAGE%=PAGE%+1:GOSUB 7000
6010   RETURN
7000   IF PAGE%>LIMIT% THEN PAGE%=LIMIT%:RECORD%=0:PLAYBACK%=0
7010   RETURN
10000 KEY%=GETC:IF KEY%=55 THEN GOTO 30000
10002 IF (KEY%=56) AND (RECORD%=0) THEN PLAYBACK%=1:POINTER%=0:PAGE%=0
10005 IF RECORD%=1 THEN ARRAY%(PAGE%,POINTER%)=KEY%:GOSUB 5000
10010 IF PLAYBACK%=1 THEN KEY%=ARRAY%(PAGE%,POINTER%)
10015 IF (RECORD%=1.0) OR (PLAYBACK%=1.0) THEN POINTER%=POINTER%+1:GOSUB
10020 IF KEY%=128 THEN PLAYBACK%=0
10030 RETURN
30000 RECORD%=1:PLAYBACK%=0:POINTER%=0:PAGE%=0
30010 KEY%=GETC:IF KEY%=0 GOTO 30010
30020 GOTO 10002
*
```

```
5     CLEAR 5000
10    MODE 6
15    DIM A(250.0),B(250.0)
20    COLORG 8 0 15 3
30    FOR X=0.0 TO 2.0*PI STEP 3E-2
40    A(N)=XMAX/2.0+100.0*COS(X):B(N)=VMAX/2.0+100.0*SIN(X*2.0)
45    N=N+1.0
50    NEXT
90    COLORG 8 0 15 3
100   FOR X=0.0 TO 209.0
110   DRAW 150,125 A(X),B(X) 0
115   DRAW 0.0 A(X),B(X) 3
116   DRAW A(X),B(X) XMAX,0 15
120   NEXT
300   FOR X=0.0 TO 50.0
320   COLORG 0 A 0 0
330   WAIT TIME 15
335   COLORG 0 0 A 0
337   WAIT TIME 15
338   COLORG 0 0 0 A
339   WAIT TIME 15
340   A=A+1.0:IF A=16.0 THEN A=1.0
345   NEXT X
400   FOR X=0.0 TO 50.0
410   COLORG RND(15.0) RND(15.0) RND(15.0) RND(15.0)
420   WAIT TIME 20
430   NEXT X
450   GOTO 90
```


========================

```
1     MODE 0:PRINT CHR$(12):PRINT :PRINT
2     PRINT "...............TOWER OF HANOI..................."
3     PRINT :PRINT
4     PRINT "AN EXAMPLE OF ANIMATED GRAPHIC CAPABILITIES OF THE"
5     PRINT :PRINT "              D A I PERSONAL COMPUTER"
6     PRINT :PRINT :PRINT :PRINT "DO YOU WANT INSTRUCTIONS"
7     PRINT :PRINT "ANSWER YES OR NO  ":INPUT A$
8     IF A$="YES" GOTO 10:IF A$="NO" GOTO 20
9     PRINT CHR$(12):PRINT :PRINT "ANSWER ONLY YES OR NO":GOTO 2
10    PRINT CHR$(12):PRINT :PRINT
11    PRINT "          TOWER OF HANOI":PRINT :PRINT :PRINT
12    PRINT "YOU HAVE TO MOVE ALL HORIZONTAL BARS FROM COLUMN 1 TO"
13    PRINT "COLUMN 3 WITHOUT PLACING A LARGER BAR ABOVE A SMALLER"
14    PRINT "BAR.     FOR MOVING THE BAR YOU PRESS ON 1 , 2  OR  3"
15    PRINT "GIVING  THE NUMBER  OF THE COLUMN  FROM WHERE THE BAR"
16    PRINT "HAS  TO LEAVE  FOLLOWED  BY THE NUMBER  OF THE COLUMN"
17    PRINT "WHERE THE BAR HAS TO GO":PRINT :PRINT :PRINT
18    PRINT "PRESS ANY KEY TO START THE GAME"
19    T=GETC:IF T=0.0 GOTO 18
20    CLEAR 2000
21    DIM Z(100.0)
22    PRINT CHR$(12)
23    COLORT 7 0 0 0
24    COLORG 7 4 5 1
25    MODE 2A
30    JC1%=0:V9=48.0:N=9.0:C1=4.0:C2=5.0:C3=1.0:C0=7.0
33    DRAW 0,0 70,0 C1
36    FOR I=1.0 TO 3.0
38    DRAW I*24-12,0 I*24-12,V9 C2
40    Z(1.0)=0.0:Z(I*10.0)=10.0:NEXT
50    M=1.0:C=C3
60    FOR I=1.0 TO N
70    Z(1.0)=I:Z(10.0+I)=10.0-I
80    GOSUB 900:NEXT
90    GOTO 110
100   PRINT "INVALID MOVE"
110   JC1%=JC1%+1:PRINT "YOUR MOVE  FROM <1,2 OR 3>  ";
111   P=GETC:WAIT TIME 5:IF P=0.0 GOTO 111
112   M1=P-48.0:M1%=M1:PRINT M1%;:PRINT "  TO  ";
113   P=GETC:WAIT TIME 5:IF P=0.0 GOTO 113
114   M2=P-48.0:M2%=M2:PRINT M2%;:PRINT "   ";:PRINT JC1%;:PRINT " MOVE
120   IF M1<>INT(M1) OR M1<1.0 OR M1>3.0 GOTO 100
130   IF M2<>INT(M2) OR M2<1.0 OR M2>3.0 GOTO 100
140   IF M1=M2 OR Z(M1)=0.0 GOTO 100
150   P1=Z(M1)+10.0*M1
160   P2=Z(M2)+10.0*M2
170   IF Z(P1)>Z(P2) GOTO 100
200   M=M1:C=C0:GOSUB 900
210   Z(M2)=Z(M2)+1.0:Z(P2+1.0)=Z(P1)
220   Z(M1)=Z(M1)-1.0
230   M=M2:C=C3:GOSUB 900
240   G=G+1.0
250   IF Z(3.0)<N GOTO 110
300   PRINT "THAT TOOK YOU ",JC1%,"MOVES"
310   STOP
900   X=M*24.0-12.0
910   Y=5.0*Z(M)
920   X1=Z(Z(M)+10.0*M)+2.0
930   DRAW X-X1,Y X-1,Y C
940   DRAW X+1,Y X+X1,Y C
950   RETURN
```

GRAPHIC OF SINUS

==============================

```
1      COLORT 0 15 0 0:PRINT CHR$(12.0):PRINT :PRINT
2      PRINT "THIS PROGRAM DRAW A SINUS WAVE ON THE SCREEN"
3      PRINT :PRINT :PRINT "IF YOUR MACHINE IS AN  8K RAM  YOU MUST CHANGE 6A
4      PRINT "INTO 2A IN LINE 12 AND INTO 4A FOR A  12 K MACHINE"
5      PRINT "THIS IS ACHIEVED BY TYPING EDIT 30 AND PLACING THE"
6      PRINT "CURSOR ON THE ''6'' OF ''6A''WITH THE CURSOR ARROW"
7      PRINT "KEY AND PRESS CHAR DEL KEY AND ''2'' OR ''4''  KEY.":PRINT
8      PRINT :PRINT 'PRESS ANY KEY TO CONTINUE"
9      P=GETC:IF P=0.0 GOTO 9
12     MODE 5A:PRINT CHR$(12):PRINT " FUNCTION  =  A *SINUS B *(X - C)+ D"
13     PRINT "A=? ";
14     P=GETC:IF P=0.0 GOTO 14
15     WAIT TIME 5:A1=P-48.0:A1%=A1:PRINT A1%,"B= ?";
16     P=GETC:IF P=0.0 GOTO 16
17     WAIT TIME 5:A2=P-48.0:A2%=A2:PRINT A2%,"C= ?";
18     P=GETC:IF P=0.0 GOTO 18
19     WAIT TIME 5:A3=P-48.0:A3%=A3:PRINT A3%,"D= ?";
20     P=GETC:IF P=0.0 GOTO 20
21     WAIT TIME 5:A4=P-48.0:A4%=A4:PRINT A4%,
25     WAIT TIME 20:PRINT CHR$(12)
30     COLORG 0 15 5 10
35     PRINT "GRAFIC OF THE FUNCTION :"
40     PRINT A1;"SIN";A2;"(X-";A3;")+";A4
50     D=XMAX/4.0/PI
60     FOR N=0.0 TO XMAX STEP D
65     DRAW N,0 N,YMAX 5
70     NEXT N
75     A4=YMAX/2.0-A4*D
80     FOR M=0.0 TO A4 STEP D
85     DRAW 0,A4-M XMAX,A4-M 5
90     NEXT M
95     FOR M=0.0 TO YMAX-A4 STEP D
100    DRAW 0,A4+M XMAX,A4+M 5
105    NEXT M
115    DRAW 0,A4 XMAX,A4 10
130    FOR X=0.0 TO XMAX
140    DOT X,SIN(A2*(4.0*PI*X/XMAX-A3))*D*A1+YMAX/2.0 15
150    NEXT X
200    PRINT "PRESS ANY KEY TO CONTINUE"
220    W=GETC:WAIT TIME 10:IF W=0.0 GOTO 220:GOTO 12
250    PRINT :PRINT :PRINT :PRINT :PRINT "G R A P H I C   O F   S I N U S":PRI
260    PRINT "==============================":PRINT :PRINT :PRINT
270    LIST
```


ARITHMETIQUE

======================

```
5      COLORT 12 0 0 0
10     A%=0:B%=0:C%=0:ANS%=0:R%=0:W%=0:POPER%=0:MODE 0
11     GOSUB 3000:GOSUB 3100:GOSUB 3300
20     CURSOR 12,21:PRINT "A R I T H M A T I C   T E A C H E R ";
22     CURSOR 15,19:PRINT "for add press...............1";
24     CURSOR 15,18:PRINT "for subtract press..........2";
26     CURSOR 15,17:PRINT "for take-away-add press.....3";
28     CURSOR 15,16:PRINT "for multiply press..........4";
30     CURSOR 15,15:PRINT "for divide press............5";
32     CURSOR 15,14:PRINT "for multiply-divide press...6";
34     CURSOR 20,12:PRINT "SELECT YOUR CHOICE";
36     CURSOR 28,10:PRINT "?";:CURSOR 28,10
50     CR%=GETC
51     CR%=GETC:IF CR%=0 THEN 51
52     IF CR%=49 THEN 100:IF CR%=50 THEN 200:IF CR%=51 THEN 400
54     IF CR%=52 THEN 600:IF CR%=53 THEN 700:IF CR%=54 THEN 800
56     GOTO 50
100    A%=0:B%=0:MODE 0:GOSUB 3300:REM CLEAR TOP OF SCREEN
101    CURSOR 28,21:PRINT "ADD"
102    POPER%=0:E%=0:MODE 0
103    GOSUB 3304
104    XP%=19:YP%=19:CURSOR XP%,YP%:X%=A%:GOSUB 1000
105    XP%=27:CURSOR XP%,YP%:X%=B%:GOSUB 1000
106    XP%=35:CURSOR XP%,YP%:X%=ANS%:GOSUB 1000
107    GOSUB 2500:REM CALCULATE RANDOM NUMBERS
108    C%=A%+B%:XP%=20:YP%=13:CURSOR XP%,YP%+1
110    PRINT A%;"   +   ";B%;"   =  ?";
112    XP%=XP%-1:CURSOR XP%,YP%:X%=A%:GOSUB 1000
114    XP%=XP%+8:CURSOR XP%,YP%:X%=B%:GOSUB 1000
118    CP%=36:GOSUB 2040:GOSUB 2050:REM PRINT R% & W%
120    GOSUB 3000:REM DRAW BASIC FACE
122    IF E%=1 THEN E%=0:GOTO 128
124    GOSUB 3100:REM DRAW REWARD FACE
126    GOTO 130
128    GOSUB 3200:REM DRAW PUNISH FACE
130    CURSOR CP%,14:ANS%=0:DIG%=0
132    GOSUB 1500
134    IF POPER%=1 THEN 10:IF POPER%=2 THEN 102
136    ANS%=CR%-48+ANS%
138    IF ANS%>C% THEN W%=W%+1:GOSUB 2050:GOSUB 3200:E%=1:GOTO 3500
140    IF ANS%<C% AND DIG%>=2.0 THEN W%=W%+1:GOSUB 2050:GOSUB 3200:E%=1:GOTO 3
142    IF ANS%<C% AND DIG%=0.0 THEN PRINT ANS%;:ANS%=ANS%*10:DIG%=DIG%+1:GOTO
143    IF ANS%=C% THEN R%=R%+1:GOSUB 2040:GOTO 146
144    DIG%=DIG%+1:PRINT ANS%;:GOTO 132
146    DIG%=0:CURSOR XP%+9,14:PRINT ANS%;
148    REM X%=ANS%:XP%=XP%+8:CURSOR XP%,YP%:GOSUB 1000
150    WAIT TIME 50:CURSOR 20,14
152    IF E%=1 GOTO 108
154    GOTO 102
200    PRINT "SUBTRACT"
202    GOTO 202
400    A%=0:B%=0:C%=0:MODE 0:GOSUB 3300:REM CLEAR TOP OF SCREEN
401    CURSOR 21,17:PRINT "TAKE-AWAY-ADD";
402    E%=0.0:MODE 0
407    XP%=16:YP%=19:X%=A%:CURSOR XP%,YP%:GOSUB 1000
408    XP%=26:X%=C%:CURSOR XP%,YP%:GOSUB 1000
409    XP%=33:X%=B%:CURSOR XP%,YP%:GOSUB 1000
410    GOSUB 2500:REM CALCULATE RANDOM NUMBERS
```

```
415   C%=A%-B%:XP%=17:YP%=13:CURSOR XP%,YP%+1
420   PRINT A%;"    ?    ?    = ";B%;
425   XP%=XP%-1:CURSOR XP%,YP%:X%=A%:GOSUB 1000
430   XP%=XP%+17:CURSOR XP%,YP%:X%=B%:GOSUB 1000
435   CP%=23:GOSUB 2040:REM PRINT R%
440   GOSUB 2050:REM AND W%
445   GOSUB 3000:REM DRAW BASIC FACE
450   IF E%=1 THEN GOTO 465
455   GOSUB 3100:REM DRAW REWARD FACE
460   GOTO 470
465   E%=0:GOSUB 3200:REM DRAW PUNISH FACE
470   CP%=CP%:CURSOR CP%,14
475   GOSUB 1500
480   IF POPER%=1.0 THEN GOTO 10
485   IF C%=0.0 AND CR%=79.0 THEN PRINT "-";:R%=R%+1:GOSUB 2040:GOTO 525
490   IF C%=0 AND CR%=81 THEN PRINT "+";:R%=R%+1:GOSUB 2040:GOTO 525
495   IF C%>0 AND CR%=79 THEN PRINT "-";:R%=R%+1:GOSUB 2040:GOTO 525
500   IF C%<0.0 AND CR%=81.0 THEN PRINT "+";:R%=R%+1:GOSUB 2040:GOTO 525
505   IF POPER%=2.0 THEN GOTO 400
510   W%=W%+1:E%=1:GOSUB 3200:REM PUNISH FACE
515   CURSOR CP%,14:GOSUB 2050
520   GOTO 475
525   CP%=CP%+5:CURSOR CP%,14
530   GOSUB 1500
535   IF POPER%=1 OR POPER%=2 THEN GOTO 475
540   D%=CR%-48
541   IF D%=ABS(C%) THEN N$=CHR$(CR%):PRINT N$;:R%=R%+1:GOSUB 2040:GOTO 56
545   W%=W%+1:GOSUB 3200:REM PUNISH FACE
550   E%=1:GOSUB 2050
555   GOTO 530
560   IF E%=1 THEN MODE 0:GOTO 415
565   C%=VAL(N$):XP%=XP%-7:YP%=YP%:X%=C%:CURSOR XP%,YP%:REM GOSUB 1000
566   WAIT TIME 50
570   CURSOR XP%+7,YP%+1:GOTO 402
600   PRINT "MULTIPLY"
602   GOTO 602
700   PRINT "DIVIDE"
702   GOTO 702
800   PRINT "MULTIPLY-DIVIDE"
802   GOTO 802
1000  REM SUBROUTINE TO PLACE DOMINO DOTS
1001  REM EXPECTS TO HAVE DEFINED BEFORE CALL
1002  REM THE X AND Y CURSOR POSITION OF THE FIRST DOT
1003  REM SPECIFIED BY (XP%) AND (YP%)
1004  REM THE NUMBER OF DOTS TO BE PRINTED
1005  REM SPECIFIED BY (X%)
1009  M%=0
1010  IF X%=0 THEN RETURN
1015  IF X%<0 THEN X%=X%+5:GOTO 1030
1020  IF X%>=5 THEN V%=5:M%=M%+1:GOSUB 1040:CURSOR XP%,YP%-M%:X%=X%-5:GOTO 1
1030  V%=X%:GOSUB 1040:RETURN
1040  FOR P%=1 TO V%:PRINT ".";:NEXT:RETURN
1500  REM ROUTINE TO GET A CHARACTER AND TEST
1501  REM FOR OTHER FUNCTIONS AS TAB AND REPT
1503  REM SETS VARIABLE POPER% TO EQUAL 1
1504  REM WHEN DESIRABLE TO RESELECT A NEW PROGRAM
1510  CR%=GETC
1511  CR%=GETC:IF CR%=0 THEN 1511
1512  IF CR%=19 THEN POPER%=2:R%=0:W%=0:GOSUB 2040:GOSUB 2050:RETURN
1515  IF CR%=16 THEN POPER%=1:RETURN
```



```
1520  RETURN
2000  REM ROUTINES THAT PRINT VALUES OF R% & W%
2001  REM IT RETURNS CURSOR TO POSITION OF CP%
2040  CURSOR 1,3:PRINT R%;:CURSOR CP%,14:RETURN
2050  CURSOR 48,3:PRINT W%;:CURSOR CP%,14:RETURN
2500  REM CALCULATES TWO RANDOM NUMBERS
2501  REM THEY ARE (A%) AND (B%)
2510  A%=10*RND(1.0):A%=INT(A%)
2520  B%=10.0*RND(1.0):B%=INT(B%)
2530  RETURN
3000  FR%=0:GOSUB 3005:FR%=47:GOSUB 3005
3005  CURSOR FR%+1,12:PRINT "#######";
3010  FOR F%=7 TO 11
3020  CURSOR FR%,F%:PRINT "# ~     ~ #";:NEXT
3030  CURSOR FR%+1,6:PRINT "#       #";
3040  CURSOR FR%+2,5:PRINT "#####";
3050  CURSOR FR%+2,10:PRINT "o    o";
3060  CURSOR FR%+2,9:PRINT "  *  ";
3061  IF FR%=47.0 THEN CURSOR 49,12:PRINT "^    ^"
3062  CURSOR 16,3:PRINT "PRESS ";CHR$(9);" KEY TO RESET SCORE"
3063  CURSOR 18,1:PRINT "PRESS ";CHR$(94);" KEY TO RESELECT"
3100  FR%=0:GOSUB 3250:FR%=47:GOSUB 3253:RETURN
3200  FR%=0:GOSUB 3253:FR%=47:GOSUB 3250:RETURN
3250  CURSOR FR%+2,8:PRINT "'    '";
3251  CURSOR FR%+2,7:PRINT " ''' ";
3252  RETURN
3253  CURSOR FR%+2,8:PRINT " ''' ";
3254  CURSOR FR%+2,7:PRINT "'    '";
3255  RETURN
3300  CURSOR 0,20:PRINT "                                        ";
3301  PRINT "                                        ";
3302  CURSOR 0,21:PRINT "                                        ";
3303  PRINT "                                        ";
3304  CURSOR 0,22:PRINT "                                        ";
3305  PRINT "                                        ";
3306  CURSOR 0,23:PRINT "                                        ";
3307  PRINT "                                        ";
3308  RETURN
3500  CURSOR 20,14:MODE 0:GOTO 108
*
```

A G E N D A

===========

```
2      CLEAR 15000
5      DIM NAME$(50.0),SURNAME$(50.0),ADRESS$(50.0)
10     PRINT CHR$(12):FOR X1=0.0 TO 59.0
20     PRINT CHR$(1);
30     NEXT X1
40     CURSOR 0,0
50     FOR X2=0.0 TO 59.0
60     PRINT CHR$(1);
70     NEXT X2
90     CURSOR 0,20
100    PRINT "*              This is a  demonstration program
110    PRINT "*              for people who do not know about
120    PRINT "*                         COMPUTER.
130    PRINT "*************************************************************"
140    GOSUB 10000
160    PRINT CHR$(12)
170    FOR X=0.0 TO 59.0
180    PRINT CHR$(2);
190    NEXT X
195    CURSOR 0,18
200    PRINT "#############################################################"
210    PRINT "#                                                           #"
220    PRINT "#  We shall make a list of i.e. 50 persons with             #"
240    PRINT "#                                                           #"
250    PRINT "#        1) NAME                                            #"
260    PRINT "#        2) SURNAME                                         #"
270    PRINT "#        3) NUMBER                                          #"
280    PRINT "#        4) ADRESS                                          #"
290    PRINT "#                                                           #"
300    PRINT "#############################################################"
400    GOSUB 10000
405    PRINT CHR$(12)
410    PRINT "#############################################################"
420    PRINT "# NOTE :- If you type an error press on   !CHAR DEL!        #"
430    PRINT "#        - NEVER press on the reset button                  #"
440    PRINT "#        - Every command to the computer must be            #"
450    PRINT "#          followed by pressing RETURN.                     #"
455    PRINT "#        - When you have typed all the names you wanted     #"
457    PRINT "#          to enter just type HALT and the same if you      #"
459    PRINT "#          want to pass to an other part of the program     #"
460    PRINT "#############################################################"
470    GOSUB 10000
500    PRINT CHR$(12)
510    PRINT "============================================================="
520    PRINT "+                          M E N U                          +"
530    PRINT "+                          -------                          +"
540    PRINT "+       1) New data base            ->> NEW                 +"
550    PRINT "+       2) Look the data            ->> LOOK                +"
560    PRINT "+       3) Search ONE of the data ->> SEARCH                +"
570    PRINT "+       4)                          ->> HALT                +"
580    PRINT "+                                                           +"
590    PRINT "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
600    PRINT CHR$(13)
```



```
610    DIM OPTIE$(1.0):INPUT "Type now one of those options !":OPTIE$
630    IF OPTIE$="NEW" GOTO 1000
640    IF OPTIE$="LOOK" GOTO 2000
650    IF OPTIE$="SEARCH" GOTO 3000
660    IF OPTIE$="VUL" GOTO 4000
670    IF OPTIE$="HALT" GOTO 5000
680    PRINT
690    PRINT "Please answer only with NEW, LOOK, SEARCH or HALT."
700    GOTO 600
1000   REM ********************** NEW **********************
1010   I%=1
1020   GOSUB 20000
1030   CURSOR 54,20
1040   PRINT I%
1050   CURSOR 8,21
1060   INPUT NAME$(I%)
1070   IF NAME$(I%)="HALT" GOTO 500
1080   CURSOR 12,20
1090   INPUT SURNAME$(I%)
1100   CURSOR 14,19
1110   INPUT ADRESS$(I%)
1120   I%=I%+1
1130   IF I%<=20 GOTO 1020
1140   PRINT "Sorry , but you have filled the data base!!!"
1150   GOSUB 10000
1160   GOTO 500
2000   REM ********************** LOOK**********************
2010   I%=1
2020   IF NAME$(I%)="HALT" GOTO 500
2025   GOSUB 20000
2030   CURSOR 54,20
2040   PRINT I%
2050   CURSOR 8,21
2060   PRINT NAME$(I%)
2070   CURSOR 12,20
2080   PRINT SURNAME$(I%)
2090   CURSOR 14,19
2100   PRINT ADRESS$(I%)
2110   GOSUB 10000
2120   I%=I%+1
2130   IF I%<=20.0 GOTO 2020
2140   PRINT CHR$(12):PRINT "You have now looked to the 50 persons  !"
2150   GOSUB 10000
2160   GOTO 500
3000   REM ********************** SEARCH **********************
3005   PRINT CHR$(12)
3010   PRINT " YOU WANT TO SEARCH A PERSON."
3020   PRINT " Which characteristic do you know???"
3030   PRINT "     1)Name          ->>NAME"
3040   PRINT "     2)Surname       ->>SURN"
3050   PRINT "     3)Adress        ->>ADRE"
3060   PRINT "     4)Number        ->>NUMB"
3070   PRINT "     5)None ....     ->>NONE"
3080   PRINT CHR$(13)
3090   DIM KOMMANDO$(1.0):INPUT KOMMANDO$
3100   IF KOMMANDO$="NAME" GOTO 3200
3110   IF KOMMANDO$="SURN" GOTO 3300
3130   IF KOMMANDO$="NUMB" GOTO 3500
3140   IF KOMMANDO$="ADRE" GOTO 3400
3150   IF KOMMANDO$="NONE" GOTO 2010
3160   PRINT :PRINT "Answer  only with NAME,SURN,NUMB,ADRE or NONE!"
```

```
3100  GOTO 3030
3200  REM ---------------------- SEARCH NAME -------------------------
3201  PRINT CHR$(12)
3202  DIM D$(1.0):INPUT "Do you know the name YES or NO ":D$
3203  IF D$="NO" GOTO 3210
3204  IF D$="YES" GOTO 7000
3205  PRINT :PRINT " Answer only with YES or NO .":PRINT :GOTO 3202
3210  PRINT :PRINT " Here follow the list of the names : "
3220  I%=1
3225  IF NAME$(I%)<>"HALT" THEN 3230
3226  GOTO 3260
3230  PRINT I%;"  ";NAME$(I%)
3240  I%=I%+1
3250  IF I%<=20 GOTO 3225
3260  INPUT "Wich number do you want to see";I%
3270  GOTO 3540
3300  REM --------------------- SEARCH SURNAME--------------------
3301  PRINT CHR$(12)
3302  DIM F$(1.0):INPUT " do you know the surname  type YES or NO";F$
3303  IF F$="NO" GOTO 3320
3304  IF F$="YES" GOTO 7100
3305  PRINT :PRINT " Answer please only wit YES or NO !!!":PRINT :GOTO 3302
3320  PRINT " Here follows the list of the surnames : "
3330  I%=1
3340  IF NAME$(I%)<>"HALT" THEN 3360
3345  GOTO 3385
3360  PRINT I%;"  ";SURNAME$(I%)
3370  I%=I%+1
3380  IF I%<=20 GOTO 3340
3385  INPUT "Wich number do you want to see  ";I%
3390  GOTO 3540
3400  REM ------------------ SEARCH ADRESS---------------------
3401  PRINT CHR$(12)
3402  DIM G$(1.0):INPUT " Do you know the adress , type YES or NO";G$
3403  IF G$="NO" GOTO 3420
3404  IF G$="YES" GOTO 7200
3405  PRINT :PRINT " Answer only with YES or NO  ":PRINT :GOTO 3402
3420  PRINT " Hereunder the list of all the adresses : "
3430  I%=1
3440  IF NAME$(I%)<>"HALT" THEN 3460
3445  GOTO 3490
3460  PRINT I%;"  ";ADRESS$(I%)
3470  I%=I%+1
3480  IF I%<=20 GOTO 3440
3490  INPUT " Wich number do you want to see ";I%
3495  GOTO 3540
3500  REM -----------------SEAR NUMBER--------------------------
3510  PRINT CHR$(12)
3520  INPUT " Wich number do you want to see";I%
3540  GOSUB 20000
3545  GOSUB 30000
3570  GOSUB 10000
3580  GOTO 500
4000  REM ********************** FILL ************************
5000  REM ********************** HALT **************************
7000  REM -------------------- NAME KNOWN-------------------
7010  I%=1:PRINT
7014  DIM GEKEND$(1.0):INPUT "Wich name do you want to see ":GEKEND$
7020  IF NAME$(I%)=GEKEND$ GOTO 7050
7030  I%=I%+1
7040  IF I%<=20 GOTO 7020
```



```
7045  GOTO 500
7050  GOSUB 20000
7060  GOSUB 30000
7070  GOSUB 10000
7080  GOTO 7030
7100  REM -------------------- SURNAME KNOWN--------------
7110  I%=1:PRINT
7114  DIM GEKEND$(1.0):INPUT " Wich surname do you want to see ";GEKEN
7120  IF SURNAME$(I%)=GEKEND$ GOTO 7150
7130  I%=I%+1
7140  IF I%<=20 GOTO 7120
7145  GOTO 500
7150  GOSUB 20000
7160  GOSUB 30000
7170  GOSUB 10000
7180  GOTO 7130
7200  REM -----------------    ADRESS KNOWN--------------
7210  I%=1:PRINT
7214  DIM GEKEND$(1.0):INPUT " Wich adress do you want to see ";GEKEND
7220  IF ADRESS$(I%)=GEKEND$ GOTO 7250
7230  I%=I%+1
7240  IF I%<=20 GOTO 7220
7245  GOTO 500
7250  GOSUB 20000
7260  GOSUB 30000
7270  GOSUB 10000
7280  GOTO 7230
9999  REM ********************** RETURNSUBR ***************
10000 CURSOR 5,3
10010 PRINT "                           ----------"
10020 CURSOR 5,2
10030 PRINT " ***    NOW PRESS ON  ! RETURN !     ***"
10040 CURSOR 5,1
10050 PRINT "                           ----------"
10060 DIM TERUG$(1.0):INPUT TERUG$
10070 RETURN
19999 REM ********************** LABELSUBR ****************
20000 PRINT CHR$(12)
20010 PRINT "**********************************************
20020 PRINT "* NAME :                                 ********
20030 PRINT "* SURNAME :                              *Nr.*
20040 PRINT "* ADRESS :                               ********
20050 PRINT "**********************************************
20060 RETURN
30000 REM ***************** PRINT SUBR **********************
30045 CURSOR 54,20:PRINT I%
30050 CURSOR 7,21:PRINT NAME$(I%)
30055 CURSOR 12,20:PRINT SURNAME$(I%)
30060 CURSOR 14,19:PRINT ADRESS$(I%)
30070 RETURN
*
```

```
                +
                ;
C003                       ORG     0C003H
                ;
C003            XMINIT:    DS      3          ; PACKAGE INIT
                ;
C006            XFINM:     DS      3          ; INCR FPT NUMBER IN MEM
C009            XFDCM:     DS      3          ; DECR FPT NUMBER IN MEM
                ;
C00C            XFCOMP:    DS      3          ; FLOATING POINT COMPARE
                ;
C00F            XIINM:     DS      3          ; INCR INT NUMBER IN MEM
C012            XIDCM:     DS      3          ; DECR INT NUMBER IN MEM
                ;
C015            XICOMP:    DS      3          ; INTEGER COMPARE
                ;
C018            XPUSH:     DS      3          ; SAVE FPAC ON STACK
C01B            XPOP:      DS      3          ; RETRIEVE FPAC FROM STACK
                ;
                ; IO FUNCTIONS
                ;
C01E            XFCB:      DS      3          ; INPUT A FPT NUMBER TO FPAC
C021            XFBC:      DS      3          ; CONVERT A FPT NUMBER FOR OUTPUT
C024            XICB:      DS      3          ; INPUT INTEGER NUMBER TO IAC
C027            XIBC:      DS      3          ; CONVERT INTEGER FOR OUTPUT
C02A            XHCB:      DS      3          ; INPUT HEX NUMBER TO IAC
C02D            XHBC:      DS      3          ; CONVERT IAC TO HEX FOR OUTPUT
C030            XPRTY:     DS      3          ; PRETTIES UP FPT OR INTEGER NUMBE
                ;
C033            DECBUF:    DS      2          ; LOCATION OF OUTPUT BUFFER
                ;
                +          PAGE
```



```
            +
            ;
            ; MEMORY + IO MAP
            ;
            ; DEFINES WHERE TO FIND THE HARDWARE
            ;
FB00        MTHAD   EQU     0FB00H  ; MATH CHIP (IF FITTED)
            ;
FC00        SNDAD   EQU     0FC00H  ; 8253 ADDRESS (IF FITTED)
            ;
FC00                SND0    EQU     SNDAD   ; CHAN 0
FC02                SND1    EQU     SNDAD+2 ; CHAN 1
FC04                SND2    EQU     SNDAD+4 ; CHAN 2
FC06                SNDC    EQU     SNDAD+6 ; CONTROL
FC00                PDLCH   EQU     SND0    ; PADDLE READING CHANNEL
            ;
                    ; 8253 MODE BYTES
            ;
0032                COM1    EQU     032H    ; CHAN 0, MODE 1, 2 BYTE OPERATION
            ;
0036                COM3    EQU     036H    ; CHAN 0, MODE 3, 2 BYTE
0076                C1M3    EQU     076H
00B6                C2M3    EQU     0B6H
            ;
0030                COM0    EQU     030H    ; CHAN 0, MODE 0, 2 BYTE OP
            ;
0000                COFIX   EQU     0       ; FIX COUNT ON CHANNEL 0
            ;
FD00        PORI    EQU     0FD00H  ; INPUT PORT
            ;
0004                PIPGE   EQU     04H     ; PAGE SIGNAL
            ;
0008                PIDTR   EQU     08H     ; SERIAL OP READY
            ;
0010                PIBU1   EQU     10H     ; BUTTON ON PADDLE 1
            ;
0020                PIBU2   EQU     20H     ; BUTTON ON PADDLE 2
            ;
0040                PIRPI   EQU     40H     ; RANDOM BITS
            ;
0080                PICAI   EQU     80H     ; CASSETTE INPUT DATA
            ;
FD01        PDLST   EQU     0FD01H  ; PADDLE SAMPLING START
            ;
FD04        PORO    EQU     0FD04H  ; VOLUME OUTPUTS CHANS 0,1
            ;
FD05        POR1    EQU     PORO+1  ; VOLUMES CHAN 2 AND NOISE
            ;
```

```
AI 8080 ASSEMBLY SERVICE,D2.2                       PAGE 11
ASIC V1.0 DISK EDIT 7 2-MARCH-80

FD06            PORO    EQU     OFD06H  ; OUTPUT PORT
                ;
0001                    POCAS   EQU     01H     ; CASSETTE OUTPUT BIT
0007                    PDLMSK  EQU     7       ; PADDLE SELECT BITS
                ;
0008                    POPNA   EQU     08H     ; PADDLE ENABLE BIT
                ;
0010                    POCM1   EQU     10H     ; CASSETTE MOTOR CONTROL
0020                    POCM2   EQU     20H     ;     "       "       "
                ;
                        ; TOP 2 BITS ARE BANK SWITCHING
                ;
FE00            GIC     EQU     OFE00H  ; RWBUS GIC ADDRESS
                ;
0080                    RWMOP   EQU     080H    ; RW OUTPUT MODE
                ;
0090                    RWMIP   EQU     090H    ; RW INPUT MODE
                ;
FFF0            TICC    EQU     OFFF0H  ; TICC ADDRESS
                ;
F900            STTOP   EQU     OF900H  ; TOP OF STACK RAM
                ;
F800            SRBOT   EQU     OF800H  ; BOTTOM OF STACK RAM
                ;
               +        PAGE
```



```
DAI 8080 ASSEMBLY SERVICE,D2.2                       PAGE 14
BASIC V1.0 DISK EDIT 7 2-MARCH-80

                +
                 ;
                 ; VARIABLES:-
                 ;
0100                     ORG     0100H
                 ;
                 ; USER STATE:
                 ;
                 ; FOLLOWING ARE SAVED BY SOFT BREAK
                 ;
                 SYSBOT:
                 ;
0100             CURRNT: DS      2       ; START OF CURRENT LINE
                 ;
0102             BRKPT:  DS      2       ; START OF CURRENT COMMAND
                 ;
0104             LOPVAR: DS      2       ; POINTS TO CURRENT LOOP VARIABLE
                                         ; 0 IF NO RUNNING LOOP
                 ;
0106             LSTPF:  DS      1       ; FLAG FOR INTEGER/FPT LOOP
                                         ; AND IMPLICIT/EXPLICIT STEP
                 ;
0107             LSTEP:  DS      4       ; STEP VALUE IF EXPLICIT
                 ;
010B             LCOUNT: DS      4       ; LOOP ITERATION COUNT
                 ;
010F             LOPPT:  DS      2       ; POINTER TO START LOOP
                 ;
0111             LOPLN:  DS      2       ; POINTER TO START LOOP LINE
                 ;
0010             FRAME   EQU     $-LOPVAR+1 ; ALLOW FOR FLAGS WHEN PUSHING I
                 ;
0113             STKGOS: DS      2       ; STACK LEVEL AT LAST GOSUB
                                         ; 0 IF NO ACTIVE CALL
                 SYSTOP:
                 ;
                 STRFL:                  ; TRACE/STEP FLAGS TOGETHER
                 ;
0115             TRAFL:  DS      1       ; TRACE FLAG
0116             STEPF:  DS      1       ; STEP FLAG
                 ;
0117             RDIPF:  DS      1       ; FLAG SET WHILE RUNNING INPUT
0118             RUNF:   DS      1       ;   "   "    "        "    PROGRAM
                 ;
                 ; PREVIOUS 2 BYTES MUST BE CONSECUTIVE
                +        PAGE
```

```
          +

          ;
          ; RUNTIME SCRATCH AREA
          ;
          GSNWK:                    ; SCRATCH AREA FOR GOSUB/NEXT (2 BYTES)
          LISW1:                    ; START OF LISTED AREA
          ;
)119      COLWK:  DS     2          ; SCRATCH AREA FOR SCOLG, SCOLT (4 BYTES
          ;
)11B      LISW2:  DS     2          ; END LISTED AREA
          ;
          ; SAVE AREA FOR RESTART ON ERROR.
          ;
)11D      ERSSP:  DS     2          ; STACK POINTER
          ;
)11F              DS     3          ;*
          ;*
)122      ERSFL:  DS     1          ; SET IF ENCODING A STORED LINE
          ;
          ; DATA/READ VARIABLES
          ;
)123      DATAC:  DS     1          ; OFFSET OF NEXT CH TO ENCODE IN "DA..."
          ;
)124      DATAP:  DS     2          ; POINTER TO CURRENT DATA LINE
          ;!DATAQ: DS     2          ; POINTER AFTER CURRENT D. LINE IF AN
          ;
)126      CONFL:  DS     1          ; SET IF THERE IS A SUSPENDED PROGRAM
          ;
)127      STACK:  DS     2          ; CURRENT BASE STACK LEVEL
          ;
)015      SFRAME  EQU    SYSTOP-SYSBOT
          ;
          ; SCRATCH LOCN FOR EXPRESSION EVALUATION
          ;
)129      WORKE:  DS     4
          ;
          ; RANDOM NUMBER KERNEL
          ;
)12D      RNUM:   DS     4
          ;
          ;!RNDLY: DS     1          ; RANDOM NUMBER DELAY COUNT
         +        PAGE
```



```
          +

          ;
          ; OUTPUT SWITCHING
          ;
0131      OTSW:   DS     1          ; 0 TO OUTPUT TO SCREEN+RS232
                                    ; 1 OUTPUT TO SCREEN
                                    ; 2 TO EDIT BUFFER
                                    ; 3 TO DISK
          ;
          ; INPUT SWITCHING
          ;
          ;!INSW:  DS     1
                                    ; 0 FROM KEYBOARD
                                    ; 1 FROM DISK
          ;
          ;
          ; ENCODING INPUT SOURCE SWITCHING
          ;
0132      EFEPT:  DS     2          ; POINTER
0134      EFECT:  DS     1          ; COUNT
          ;
0135      EFSW:   DS     1          ; SET 0:      INPUT FROM KB/SCREEN
                                    ;     1:         "     "   STRING
                                    ;     2:         "     "   EDIT BUFFER
          ;
          ; VARIABLES USED DURING EXPRESSION ENCODING
          ; (COULD OVERLAP WITH RUNTIME VARIABLES)
          ;
0136      TYPE:   DS     1          ; TYPE OF LATEST EXPRESSION OR ITEM
          ;
0137      RGTOP:  DS     1          ; LATEST PRIORITY OPERATOR
          ;
0138      OLDOP:  DS     1          ; OLD PRIORITY+OPERATOR
          ;
0139      HOPPT:  DS     2          ; PTR TO PLACE FOR OPERATOR
          ;
013B      RGTPT:  DS     2          ; PTR TO RGT OPERAND LATEST OPERATOR
          ;
          ; ORDER OF LAST 7 BYTES IS IMPORTANT
          ;
          +       PAGE
```

```
               +
               ;
               ; MASK TO SELECT CASSETTE 1 OR 2
               ;
013D           CASSL:  DS      1       ; #10 FOR CASSETTE 1,#20 FOR 2
               ;
               ; ENCODED INPUT BUFFER
               ;
013E           EBUF:   DS      128     ; USED ALSO BY UTILITY
               ;
               ; INTERRUPT HANDLER VARIABLES
               ;
005F           TICIM   EQU     05FH    ; CURRENT INTERRUPT MASK
               ;
01BE           TIMER:  DS      2       ; TIMER LOCATION
               ;
01C0           CTIMR:  DS      1       ; CURSOR CLOCK
               ;
000F           CTIMV   EQU     15      ; FLASH TIME IN 20 MS UNITS
               ;
01C1           KBXCT:  DS      1       ; EXTEND KB SCAN TIME COUNTER
               ;
0002           KBXCK   EQU     2       ; KB SCAN TIME (UNITS OF 16 MS)
                                       ; RAND ROUTINE NEEDS THIS EVEN
               ;
               ; INTERRUPT MASKS DEFINITIONS
               ;
FFFB           SNDIAD  EQU     TICC+0BH ; SOUND TIMER ADDR
0008           SNDIM   EQU     08H     ; SOUND INT MASK BIT
               ;
FFFC           KBIAD   EQU     TICC+0CH ; KB TIMER ADDR
0040           KBIM    EQU     40H     ; KEYBOARD "   "   "
               ;
0080           CLKIM   EQU     080H    ; CLOCK "   "   "
               ;
0004           STKIM   EQU     04H     ; STACK "   "   "
               +       PAGE
```



```
               +
               ;
               ; IO LOCATIONS
               ;
               ;!POROM:         DS      1       ; MEMORY OF
               ;!POR1M:         DS      1       ; LAST OUTPUTS TO
0040           POROM   EQU     40H     ; OUTPUT PORTS
               ;
               ; SOUND CONTROL BLOCK STORAGE
               ;"
000E           SCBL    EQU     14      ; LENGTH OF A SOUND CONTROL BLOCK
0009           NCBL    EQU     9       ;   "    "  NOISE    "      "
               ;
)1C2           SCBO:   DS      3*SCBL+NCBL ; SOUND + NOISE CHANNELS
               ;
               ; ENVELOPE STORAGE
               ;
)040           ENVLL   EQU     64      ; NUMBER OF BYTES/ENVELOPE
               ;
)002           NUMENV  EQU     2       ; NUMBER OF ENVELOPES
               ;
)1F5           ENVST:  DS      NUMENV*ENVLL ; ENVELOPE STORAGE
               ;
)275           IMPTAB: DS      'Z'-'A'+1 ; IMPLICIT TYPE TABLE
               ;
)28F           IMPTYP: DS      1       ; DEFAULT NUMBER TYPE
               ;
)290           REQTYP: DS      1       ; REQUIRED NUMBER TYPE
               ;
               ; SPARE VARIABLE SPACE
               ;
(291                   DS      10
(291           DATAQ   EQU     0291H   ;*
(293           RNDLY   EQU     0293H   ;*
(294           POROM   EQU     0294H   ;*
(295           POR1M   EQU     0295H   ;*
(296           INSW    EQU     0296H   ;*
               +       PAGE
```

```
                +
                ;
                ;  HEAP/TEXT BUFFER/SYMTAB POINTERS
                ;
29B             HEAP:    DS      2          ; START OF HEAP
                ;
29D             HSIZE:   DS      2          ; SIZE OF HEAP
100             HSIZD    EQU     100H       ; DEFAULT SIZE
                ;
29F             TXTBGN:  DS      2          ; START OF TEXT BUFFER
                ;
                TXTUSE:                     ; END TEXT AREA AND
2A1             STBBGN:  DS      2          ; START SYMBOL TABLE
                ;
2A3             STBUSE:  DS      2          ; END SYMBOL TABLE
                ;
2A5             SCRBOT:  DS      2          ; BOTTOM OF SCREEN RAM AREA
                ;
                +        PAGE
```



```
                +
                ;
                ;  KEYBOARD VARIABLES + CONSTANTS
                ;
02A7            KBTPT:   DS      2          ; POINTER TO CODE TABLE
                ;
02A9            MAP1:    DS      8          ; LATEST SCAN OF KEYS
                ;
02B1            MAP2:    DS      8          ; PREVIOUS SCAN
                ;
02B9            KNSCAN:  DS      1          ; SET TO SCAN FOR BREAK ONLY
                ;
0004            KBLEN    EQU     4          ; LENGTH OF ROLLOVER BUFFER
                KEYL:
02BA            KLIND:   DS      KBLEN      ; CIRCULAR BUFFER FOR KEYS PRESSED
                ;
02BE            KLIIN:   DS      2          ; NEXT POSN FOR INPUT TO KLIND
02C0            KLIOU:   DS      2          ; NEXT POSN FOR OUTPUT FROM KLIND
                ;
02C2            RPCNT:   DS      1          ; COUNT FOR REPT
                ;
02C3            SHLK:    DS      1          ; SET IF "SHIFT INVERT"
                ;
                         IF SUSP
                ;
02C4            KBRFL:   DS      1          ; FLAG FOR "BREAK PRESSED"
                ;
                         ENDIF
                ;
02B0            SHLOC    EQU     MAP1+7     ; BYTE CONTAINING SHIFT
0040            SHMSK    EQU     040H       ; SHIFT KEY BIT
                ;
02AF            RPLOC    EQU     MAP1+6     ; BYTE CONTAINING REPT KEY
0020            RPMSK    EQU     020H       ; REPT KEY BIT
                ;
0002            RPLIM    EQU     2          ; TIMING FOR REPT
                ;
0040            BRSEL    EQU     040H       ; COLUMN SELECT MASK FOR BREAK
0040            BRMSK    EQU     040H       ; BREAK KEY BIT
                ;
0020            BRLIM    EQU     20H        ; TIMING FOR HARD BREAK
                ;
                +        PAGE
```

```
                +
                ;
                ;   DISC/CASSETTE SWITCHING VECTOR
                ;
                IOVEC:
                ;
02C5            WOPEN:  DS      3
                ;
02C8            WBLK:   DS      3
                ;
02CB            WCLOSE: DS      3
                ;
02CE            ROPEN:  DS      3
                ;
02D1            RBLK:   DS      3
                ;
                RCLOSE:
02D4            RCLO:   DS      3
                ;
02D7            MBLK:   DS      3
                ;
02DA            RESET:  DS      3
                ;
02DD            DOUTC:  DS      3
                ;
02E0            DINC:   DS      3
                ;
02E3                    DS      3       ; SPARE
                ;
02E6            TAPSL:  DS      2
                ;
02E8            TAPSD:  DS      2
                ;
02EA            TAPST:  DS      2
                ;
                VAREND:
                VARLAST:
                ;
02EC            RAM     SET     $
                ;
                +       PAGE
```



```
                +
                ;
C6C0                    ORG     0C6C0H  ; START OF BASIC
                ;
                ; BANK SWITCHING RESTARTS
                ;
                ; THE FOLLOWING ROUTINES SWITCH THE PAGED
                ;  BANKS OF ROM.  THEY ARE ENTERED VIA RST INSTRUCTIONS
                ;
                MARST:
                ;
C6C0 E1                 POP     H
                ;
C6C1 F3                 DI
                ;
C6C2 224300             SHLD    RSWK2   ; SAVE HL
C6C5 F5                 PUSH    PSW
C6C6 E1                 POP     H
C6C7 224100             SHLD    RSWK1   ; PSW
                ;
C6CA 2640               MVI     H,040H  ; BANK SELECT BITS FOR MATH PACK
C6CC 3AD400             LDA     MVECA   ; OFFSET OF START HW/SW VECTOR
                MRS10:
C6CF E3                 XTHL
C6D0 86                 ADD     M       ; ADD ENTRY NUMBER
C6D1 23                 INX     H
C6D2 E3                 XTHL
                ;
C6D3 6F                 MOV     L,A     ; COMPLETE ENTRYPOINT ADDRESS
C6D4 3A4000             LDA     POROM   ; BANK SELECT PORT STATUS
C6D7 F5                 PUSH    PSW     ; REMEMBER
C6D8 E63F               ANI     03FH    ; KEEP OTHER BITS
C6DA B4                 ORA     H       ; ADD NEW SELECT BITS
C6DB 324000             STA     POROM   ; UPDATE MEMORY
C6DE 3206FD             STA     PORO    ; AND PORT
                ;
C6E1 26E0               MVI     H,VECA SHR 8
C6E3 CDF2C6             CALL    MRDCL
                ;
C6E6 E3                 XTHL
C6E7 F5                 PUSH    PSW
C6E8 7C                 MOV     A,H
C6E9 324000             STA     POROM   ; REINSTATE MEMORY
C6EC 3206FD             STA     PORO    ; + PORT
C6EF F1                 POP     PSW
C6F0 E1                 POP     H
C6F1 C9                 RET             ; BACK TO CALLER
                ;
```

```
DAI 8080 ASSEMBLY SERVICE, D2. 2                              PAGE 23
BASIC V1. 0 DISK EDIT 7 2-MARCH-80

                    MRDCL:
  C6F2 E5                         PUSH     H
  C6F3 2A4100                     LHLD     RSWK1
  C6F6 E5                         PUSH     H
  C6F7 F1                         POP      PSW
  C6F8 2A4300                     LHLD     RSWK2
  C6FB FB                         EI
  C6FC C9                         RET

                                  PAGE
```


THIS PROGRAM NAMED SUM IS CALLING A MACHINE LANGUAGE SUBROUTINE LOADED AS AN ARRAY ''A'' NAMED ''SUM A'' THE SUBROUTINE ,LOCATED AT #3FC , PERFORMS INTEGER CALCULATION WITH 64 DIGITS RESOLUTION. YOU MUST LOAD THE PROGRAM, STOP THE RECORDER IF YOU DO NOT USE THE REMOTE CONTROL, RUN THE PROGRAM WHAT IS NOW LOADING THE ROUTINE AS AN ARRAY AND ASK YOU THE OPERATION TO PERFORM I.E. 12345+432 <RETURN> AND GIVES THE RESULT. IF YOU PRESS THE BREAK KEY TO CONTINUE YOU HAVE NOW TO RUN 35 ,OR FIRST TYPE 1 <RETURN> TO 24 <RETURN> WHAT WILL ERASE THIS TEXT AND LOADA ROUTINE AND YOU CAN NOW MAKE A NORMAL RUN. IF YOU WANT TO SAVE THE PROGRAM AND THE ROUTINE YOU MUST SAVE''PROGRAM NAME'' STOP RECORDER, SAVEA A''ROUTINE NAME''

YOU WILL NOTICE IF YOU LIST THE PROGRAM THAT 3 FIRST LINES ARE CLEAR 2000, DIM A(20,20), LOADA A''SUM A'' AFTER YOU HAVE LOADED THE ARRAY YOU CANNOT EDIT NOR CLEAR NOR DIM ARRAYS ALREADY DIMENSIONED.

PRESS ANY KEY CONTINUE THE PROGRAM LOADING ROUTINE

```
10    CLEAR 2000
20    DIM A(20.0,20.0)
22    LOADA A "SUM A"
35    PRINT "WHAT IS YOUR SUM    ";
40    INPUT A$
45    PRINT
50    CALLM #3FC,A$
60    PRINT "HERE IS THE ANSWER!",A$
70    GOTO 35
```

```
03F0 00 00 00 00 00 00 00 00 00 00 00 00 F5 C5 D5 7E
0400 23 66 6F E5 4E 23 CD 2F 05 11 99 06 CD F1 04 CA
0410 29 04 78 32 5E 07 7E 36 20 32 60 07 23 0D CA 89
0420 04 11 DA 06 CD F1 04 78 32 5F 07 21 1B 07 11 99
0430 06 01 DA 06 3A 60 07 FE 2A CA 68 04 FE 2F CA 7F
0440 04 FE 2B CA 52 04 FE 2D C2 89 04 3A 57 07 2F 32
0450 5F 07 3A 5E 07 A7 CC F1 05 C4 DA 05 C5 D1 3A 5F
0460 07 A7 CC F1 05 C4 DA 05 C3 92 04 CD 06 06 CA 89
0470 04 3A 5E 07 47 3A 5F 07 A8 32 5D 07 C3 92 04 CD
0480 C2 06 CA 89 04 C3 71 04 E1 E1 D1 C1 F1 23 36 3F
0490 [illegible] C9 CD 61 07 3A 5D 07 E1 E5 23 06 00 A7 CA A5
04A0 04 36 2D 23 04 11 5B 07 1B 1A A7 CA A8 04 E5 21
04B0 E6 F8 19 4D E1 1A F6 30 77 23 1B 0D FA EB 04 C2
04C0 05 04 F1 5E 16 00 4A E5 19 7E FE 20 C2 D5 04 2B
04D0 C2 1D CC C9 04 79 D6 03 FA E6 04 0D 0D E1 E5 73
04E0 23 19 36 00 23 71 E1 D1 C1 F1 C9 2B 36 30 C3 C2
04F0 04 06 00 7E E6 30 FE 30 CA 15 05 2B 23 0D C8 7E
0500 7E 20 FE 20 CA FC 04 23 FE 2B CA 15 05 FE 2D C2
0510 29 04 3E FF 47 7E FE 20 CA 29 05 E6 30 FE 30 C0
0520 CD BF 05 7E 36 20 E5 0F 12 23 0D C2 15 05 C9 E5
0530 21 99 06 1E 40 36 FF 23 1D C2 35 05 21 DA 06 1E
0540 40 36 FF 23 1D C2 41 05 21 1B 07 1E 40 36 00 23
0550 1D C2 4D 05 AF 32 5D 07 32 5E 07 32 5F 07 E1 C9
0560 F5 D5 C5 E5 11 40 00 19 EB E1 CD 71 05 C1 D1 E1
0570 C9 D5 1B 1A A7 C2 93 05 7B BD C2 72 05 D1 AF 32
0580 5D A7 C9 F2 9A 05 3A 5D 07 2F 32 5D 07 D5 13 1B
0590 1A 2F 3C 12 7B BD C2 8F 05 D1 13 13 E5 06 00 7E
05A0 FE 06 00 F2 B6 05 05 C6 0A FA A6 05 77 23 7B BD
05B0 C2 9F 05 E1 D1 C9 04 D6 0A F2 B6 05 C3 A6 05 C5
05C0 D5 CD C9 05 D1 C1 FE 80 C9 1A F5 AF 12 F1 13 47
05D0 1A F5 78 12 F1 A7 F2 CE 05 C9 F5 D5 E5 CD E4 05
05E0 E1 D1 F1 C9 1A FE FF C8 2F 3C 86 77 13 23 C3 E4
05F0 05 F5 D5 E5 CD FB 05 E1 D1 F1 C9 1A FE FF C8 86
0600 77 13 23 C3 FB 05 0A 3D 02 FA 12 06 CD F1 05 C3
0610 06 06 CD 60 05 03 0A 2F D6 01 D8 CD BF 05 C3 C3
0620 06 06 AF 32 5C 07 E5 21 D9 06 2B 7E FE FF C2 36
0630 06 36 00 C3 2A 06 E1 36 01 CD 83 06 FA 52 06 23
0640 D5 C5 D1 CD BF 05 D1 C8 3A 5C 07 3C 32 5C 07 C3
0650 37 06 35 CD 83 06 2B D5 C5 D1 CD 70 06 D1 3A 5C
0660 07 3D 32 5C 07 F8 34 CD 83 06 FA 52 06 C3 66 06
0670 C5 D5 CD 79 06 D1 C1 C9 13 1A 1B 12 13 FE FF C8
0680 C3 78 06 E5 C5 D5 D5 E1 C5 D1 CD DA 05 CD 60 05
0690 D1 C1 E1 3A 5D 07 A7 C9 80 00 00 00 00 00 00 00
06A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0700 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0710 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0720 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0730 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0740 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0750 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0760 00 01 1B 07 C3 60 05 00 00 00 00 00 00 00 00 00
```

REAL TIME CLOCK


```
   CLEAR 300
   POKE #29C,3:POKE #29E,0:POKE #3EC,#80:POKE #3ED,#28
   FOR T%=0 TO 11:READ D%
   FOR T1%=0 TO 15:READ D1%
   IF D1%>=#100 THEN D1%=(PEEK(#2A6) IAND #FE IOR #E)+D1%-#100
   POKE D%,D1%:D%=D%+1:NEXT:NEXT
50 POKE #71,#3:POKE #70,#0
   DATA #300,#C5,#D5,#E5,#F5,#21,#89,#03,#06,#0A,#0E,#06,#16,#00,#1E,#32,#34
   DATA #310,#7B,#BE,#C2,#57,#03,#72,#23,#34,#78,#BE,#C2,#5E,#03,#72,#23,#34
   DATA #320,#79,#BE,#C2,#5E,#03,#72,#23,#34,#78,#BE,#C2,#5E,#03,#72,#23,#34
   DATA #330,#79,#BE,#C2,#5E,#03,#72,#23,#34,#78,#BE,#C2,#5E,#03,#72,#23,#34
   DATA #340,#2B,#23,#3E,#02,#BE,#C2,#5E,#03,#2B,#3E,#04,#BE,#C2,#5E,#03,#36
   DATA #350,#00,#23,#36,#00,#C3,#3E,#03,#F1,#E1,#D1,#C1,#C3,#49,#D9,#3A,#EF
   DATA #360,#101,#FE,#7A,#C2,#57,#03,#21,#BA,#03,#7E,#06,#30,#32,#F1,#100,#
   DATA #370,#7E,#06,#30,#32,#F3,#100,#23,#7E,#06,#30,#32,#F7,#100,#23,#7E,#(
   DATA #380,#30,#32,#F9,#100,#23,#7E,#06,#30,#32,#FD,#103,#23,#7E,#06,#30,#
   DATA #390,#FF,#100,#3E,#FF,#32,#FC,#100,#32,#EE,#100,#32,#F0,#100,#32,#F2,
#100
   DATA #3A0,#32,#F4,#100,#32,#F6,#100,#32,#F8,#100,#32,#FA,#100,#32,#FC,#100
#32
   DATA #3B0,#FE,#100,#32,#00,#101,#00,#C3,#5F,#03,#1A,#00,#00,#00,#00,#00,#0
   INPUT "INPUT THE TIME  < HH,MM,SS > ";T$:PRINT :A%=#3BF
   FOR D%=0 TO LEN(T$)-1:T1$=MID$(T$,D%,1)
   IF ASC(T1$)>47 AND ASC(T1$)<58 THEN POKE A%,VAL(T1$):A%=A%-1:IF A%=#3B9 TH
N STOP
   [illegible]
```

[illegible] UTILITY V2.3

0300 3FF

```
0300 C5 D5 E5 F5 21 B9 03 06 0A 0E 06 16 00 1E 32 34
0310 78 BE C2 57 03 72 23 34 78 BE C2 5E 03 72 23 34
0320 78 BE C2 5E 03 72 23 34 78 BE C2 5E 03 72 23 34
0330 78 BE C2 5E 03 72 23 34 78 BE C2 5E 03 72 23 34
0340 2B 23 3E 02 BE C2 5E 03 2B 3E 04 BE C2 5E 03 36
0350 00 23 36 00 C3 5E 03 F1 E1 D1 C1 C3 A9 D9 3A EF
0360 7F FE 7A C2 57 03 21 9A 03 7E C6 30 32 F1 7E 23
0370 7E C6 30 32 F3 7E 23 7E C6 30 32 F7 7E 23 7E C6
0380 30 32 F9 7E 23 7E C6 30 32 FD 7E 23 7E C6 30 32
0390 FF 7E 3E FF 32 EC 7E 32 EE 7E 32 F0 7E 32 F2 7E
03A0 32 F4 7E 32 F6 7E 32 F8 7E 32 FA 7E 32 FC 7E 32
03B0 FE 7E 32 00 7F 00 C3 5F 03 2A 09 01 00 02 06 00
03C0 45 35 2C 23 46 35 2C 23 32 31 2C 23 42 39 2C 23
03D0 30 33 2C 23 30 36 2C 23 30 41 2C 23 30 45 2C 23
03E0 30 36 2C 23 31 36 2C 23 30 30 2C 23 00 07 30 36
03F0 20 31 35 32 35 00 01 35 80 01 32 80 19 18 00 00
```


*
R O T A T I N G   P Y R A M I D

================================

```
2     PRINT "ROTATING PYRAMIDE ,1,2,3 AND 4 ARE USED"
3     PRINT "WITH REPT KEY FOR ROTATION":WAIT TIME 400
5     MODE 6:MODE 6:SF=3.5:REM MODE +SCALING FACTOR
6     COLORG 0 15 0 15
7     GOSUB 2000:REM INITIALISE DATA
90    REM
92    GOSUB 800:REM DRAW NEW SHAPE
95    COLORG 0 15*(1-Q) 15*Q 15
96    GOSUB 900:REM ERASE OLD SHAPE
97    Q=1.0-Q
99    KS=ABS(KS)
100   A=GETC:IF A<ASC("0") THEN 100
120   FOR P=1.0 TO NP
130   XX(P)=X(P):YY(P)=Y(P)
140   NEXT
141   REM
150   ON A-ASC("0") GOTO 500,510,600,610,700,710
160   GOTO 100
161   REM
162   REM
500   KS=-KS
510   FOR P=1.0 TO NP
520   X=X(P):Y=Y(P)
530   X(P)=X*KC+Y*KS
540   Y(P)=Y*KC-X*KS
550   NEXT
560   GOTO 90
590   REM
591   REM
600   KS=-KS
610   FOR P=1.0 TO NP
620   Y=Y(P):Z=Z(P)
630   Y(P)=Y*KC+Z*KS
640   Z(P)=Z*KC-Y*KS
650   NEXT
660   GOTO 90
661   REM
662   REM
700   KS=-KS
710   FOR P=1.0 TO NP
720   Z=Z(P):X=X(P)
730   Z(P)=Z*KC+X*KS
740   X(P)=X*KC-Z*KS
```

```
750   NEXT
760   GOTO 90
800   REM
801   REM DRAW NEW PICTURE
802   REM
810   FOR L=1.0 TO NL
820   PA=LA(L)
830   PB=LB(L)
840   DRAW X(PA)+XC,Y(PA)+YC X(PB)+XC,Y(PB)+YC 17+Q*2
850   NEXT
860   RETURN
900   REM
901   REM ERASE OLD PICTURE
902   REM
910   FOR L=1.0 TO NL
920   PA=LA(L)
930   PB=LB(L)
940   DRAW XX(PA)+XC,YY(PA)+YC XX(PB)+XC,YY(PB)+YC 18-2*Q
950   NEXT
960   RETURN
990   REM
991   REM DATA SETUP ROUTINE
992   REM
2000  PHI=PI/20.0
2010  KS=SIN(PHI)
2020  KC=COS(PHI)
2030  XC=XMAX/2.0
2040  YC=YMAX/2.0
2050  Q=1.0
2100  READ NP,NL
2110  DIM X(NP),Y(NP),Z(NP)
2120  DIM XX(NP),YY(NP)
2130  DIM LA(NL),LB(NL)
2131  REM
2200  FOR P=1.0 TO NP
2210  READ X(P),Y(P),Z(P)
2211  X(P)=X(P)*SF
2212  Y(P)=Y(P)*SF
2213  Z(P)=Z(P)*SF
2220  NEXT
2221  REM
2230  FOR L=1.0 TO NL
2240  READ LA(L),LB(L)
2250  NEXT
2251  REM
2260  GOSUB 800
2270  RETURN
2300  REM
2301  REM DATA
2302  REM
2800  REM NUMBER OF POINTS AND NUMBER OF LINES
2900  DATA 5,8
2901  REM
2903  DATA 0,0,20
2904  DATA 20,20,-20
2905  DATA 20,-20,-20
2906  DATA -20,20,-20
2907  DATA -20,-20,-20
2909  REM
2910  DATA 1,2
2911  DATA 1,3
2912  DATA 1,4
```



```
2913  DATA 1,5
2914  DATA 2,3
2915  DATA 2,4
2916  DATA 3,5
2917  DATA 4,5
2999  DATA 8,12
4000  DATA 1,2
4001  REM DATA FOR SOMETHING ELSE!
4002  REM
4009  DATA 20,20,20
4010  DATA 20,20,-20
4020  DATA 20,-20,20
4030  DATA 20,-20,-20
4040  DATA -20,20,20
4050  DATA -20,20,-20
4060  DATA -20,-20,20
4070  DATA -20,-20,-20
4110  DATA 1,3
4120  DATA 1,5
4130  DATA 2,4
4140  DATA 2,6
4150  DATA 3,4
4160  DATA 3,7
4170  DATA 4,8
4180  DATA 5,6
4190  DATA 5,7
4210  DATA 7,8
9999  END
*
```

RAPS

========

```
     C1=1.0
     C2=0.0
     C3=14.0
     C0=13.0
0    COLORG C0 C1 C2 C3:COLORT C0 0 0 0
1    MODE 3A
2    H=GETC
00   REM DRAW 14,19 14,68 C1
10   REM DRAW 14,68 63,68 C1
20   REM DRAW 63,68 63,19 C1
30   REM DRAW 63,19 14,19 C1
40   FILL 15,20 62,67 C2
50   REM DRAW 94,19 94,68 C1
60   REM DRAW 94,68 143,68 C1
70   REM DRAW 143,68 143,19 C1
80   REM DRAW 143,19 94,19 C1
90   FILL 95,20 142,67 C2
200  GOSUB 1200
210  PFS=0.0:TOSS%=0
212  CURSOR 0,3:PRINT "                    TO SHOOT CRAPS PRESS ANY KEY
213  CURSOR 0,2:PRINT "     point                                   tosses :
214  CURSOR 0,1:PRINT "                                                    "
215  CURSOR 0,0:PRINT "                                                    ";
216  CURSOR 28,2:PRINT "$";:CURSOR 28,2
220  GOSUB 1300
251  IF SUM%=7.0 OR SUM%=11.0 THEN CURSOR 25,1:GOSUB 1500:GOTO 210
252  IF SUM%=2.0 OR SUM%=3.0 OR SUM%=12.0 THEN CURSOR 24,1:GOSUB 1600:GOTO
253  POINT%=SUM%
254  GOSUB 1400:GOSUB 1300
255  IF POINT%=SUM% THEN CURSOR 25,1:GOSUB 1500:GOTO 210
60   IF SUM%=7 THEN CURSOR 25,1:GOSUB 1600:GOTO 210
80   GOTO 254
00   D=1.0+INT(10.0*RND(1.0)):IF D>6.0 GOTO 700
00   A=V+19.0
01   A1=A+7.0
02   B=V+35.0
03   B1=B+7.0
04   C=V+51.0
05   C1=C+7.0
10   IF D=1.0 OR D=3.0 OR D=5.0 THEN FILL B,40 B1,47 C3
20   IF D=1 THEN RETURN
30   FILL A,56 A1,63 C3
35   FILL C,24 C1,31 C3
```



```
840   IF D<4 THEN RETURN
850   FILL A,24 A1,31 C3
855   FILL C,56 C1,63 C3
860   IF D<6 THEN RETURN
870   FILL A,40 A1,47 C3
875   FILL C,40 C1,47 C3
880   RETURN
1200  FILL 19,24 58,63 C2
1210  FILL 99,24 138,63 C2
1220  V=0.0:GOSUB 700
1230  SUM%=INT(D)
1240  V=80.0:GOSUB 700
1245  SUM%=SUM%+INT(D)
1250  RETURN
1300  WAIT TIME 10:H=GETC:IF H=0.0 GOTO 1300:GOSUB 1200:RETURN
1400  CURSOR 6,1:IF POINT%<>0 THEN PRINT POINT%," ";
1401  TOSS%=TOSS%+1:CURSOR 47,1:PRINT TOSS%:CURSOR 28,2:RETURN
1500  PRINT "you win";:JF=1.0:WAIT TIME 200:RETURN
1600  PRINT "you lose";:JF=1.0:WAIT TIME 200:RETURN
*
```

*
R A N D O M L I N E S 3

======================

```
5     COLORG 7 15 0 0
10    MODE 6
100   S%=X% MOD (XMAX):T%=Y% MOD (YMAX)
105   FOR A%=0 TO 60:X%=RND(XMAX):Y%=RND(YMAX)
110   DRAW S%,T% X%,Y% 15:DRAW S%,T% X%,Y% 0:S%=X%:T%=Y%
120   NEXT:WAIT TIME 100:GOTO 10
```

*
B U G

=====

```
5     MODE 5
10    X%=5:FOR Q%=YMAX-6 TO 0 STEP -1:X%=X%+1:GOSUB 100:NEXT
20    GOTO 5
100   DOT X%,Q% 15
110   DOT X%-1,Q%+1 13
120   DOT X%-2,Q%+2 11
130   DOT X%-3,Q%+3 8
140   DOT X%-4,Q%+4 6
150   DOT X%-5,Q%+5 3
160   DOT X%-6,Y%+6 1
170   RETURN
```

*
S O U N D S

===========

```
10    ENVELOPE 0 16:FOR A=0.0 TO 2.0:SOUND A 0 15 0 FREQ(33.0):NEX
20    FOR A=5.0 TO 541.0 STEP A:GOSUB 100:NEXT
30    FOR Z=440.0 TO 33.0 STEP -(Z/100.0)
40    FOR G=0.0 TO 2.0:SOUND G 0 15 2 FREQ(Z+G)
50    NEXT G:WAIT TIME 5:NEXT Z:GOTO 10
100   Q=A MOD 3.0:R=(Q+1.0) MOD 3.0:S=(Q+2.0) MOD 3.0
110   SOUND Q 0 15 2 FREQ(A+32.0)
120   SOUND R 0 15 2 FREQ(A*A+32.0)
130   SOUND S 0 15 2 FREQ(A*A*A+32.0)
140   RETURN
```


*
C O L O R   G R A P H I C S

=========================

```
10    MODE 2:GOSUB 20:MODE 4:GOSUB 20:MODE 6:GOSUB 20:GOTO 10
20    FOR A%=0 TO YMAX:DRAW 0,0 XMAX,A% 20+(A% MOD 3):NEXT
30    FOR A%=0 TO XMAX-1:DRAW 0,0 A%,YMAX 20+(A% MOD 3):NEXT
40    FOR S%=0 TO 20:COLORG RND(15) RND(15) RND(15) RND(15)
50    WAIT TIME 20:NEXT S%:RETURN
```

*
G R A P H I C S   2

==================

```
10    MODE 2:GOSUB 20:MODE 4:GOSUB 20:MODE 6:GOSUB 20:GOTO 10
20    FOR A%=0 TO YMAX STEP 3:W%=W%+1:DRAW 0,0 XMAX,A% 20+(W% MOD 3):NEXT
30    FOR A%=0 TO XMAX-1 STEP 3:W%=W%+1:DRAW 0,0 A%,YMAX 20+(W% MOD 3):NEXT
40    FOR A%=1 TO XMAX STEP 3:W%=W%+1:DRAW A%,0 XMAX,YMAX 20+(W% MOD 3):NEXT
50    FOR A%=1 TO YMAX STEP 3:W%=W%+1:DRAW 0,A% XMAX,YMAX 20+(W% MOD 3):NEXT
60    FOR S%=0 TO 20:COLORG RND(15) RND(15) RND(15) RND(15)
70    WAIT TIME 20:NEXT S%:RETURN
```

*
R A N D O M   L I N E S

=====================

```
5     COLORG 7 15 0 0
10    MODE 4
100   S%=X% MOD (XMAX):T%=Y% MOD (YMAX)
105   FOR A%=0 TO 2:X%=RND(XMAX):Y%=RND(YMAX)
110   DRAW S%,T% X%,Y% 15:DRAW S%,T% X%,Y% 0:S%=X%:T%=Y%:NEXT:GOTO 10
```

```
5     ENVELOPE 0 15,2:10,2:15,2:10,2:0
6     ENVELOPE 1 15,5:12,5:10,100:0
10    REM music compose program
15    ENVELOPE 0 6
16    CLEAR 8000
17    DIM N$(50.0):DIM F%(50.0):DIM T(255.0):DIM E(255.0)
18    DIM V(255.0):DIM M(255.0):DIM D(255.0):DIM S(255.0)
20    DATA C0,65,C0+,69,D0,73,D0+,78,E0,82,F0,87,F0+,92,G0
21    DATA 98,G0+,104,A0,110,A0+,116,B0,123
30    DATA C,131,C+,138,D,147,D+,155,E,165,F,175,F+,185,G
31    DATA 196,G+,208,A,220,A+,233,B,247
40    DATA C1,262,C1+,277,D1,294,D1+,311,E1,330,F1,349,F1+
41    DATA 370,G1,392,G1+,415,A1,440,A1+,466,B1,494
50    DATA C2,523,C2+,554,D2,587,D2+,622,E2,659,F2,698,F2+
51    DATA 740,G2,784,G2+,831,A2,880,A2+,932,B2,988
60    FOR X=1.0 TO 48.0:READ N$(X):READ F%(X):NEXT
70    N$(0.0)="O":F%(0.0)=60000
75    N$(49.0)="C3":F%(49.0)=1046
90    PRINT CHR$(12)
100   REM compose
110   FOR X=1.0 TO 255.0
120   READ S(X):IF S(X)=999.0 THEN GOTO 190
125   READ E(X),NOTE$,V(X),D(X),M(X)
130   FOR Y=0.0 TO 48.0
140   IF NOTE$=N$(Y) THEN T(X)=F%(Y):GOTO 180
150   NEXT Y
180   NEXT
190   CURSOR 10,10
191   PRINT "from the motion picture ' THE STING '"
192   CURSOR 20,8:PRINT "THE ENTERTAINER "
194   CURSOR 30,6:PRINT "by SCOTT JOPLIN"
200   FOR P=1.0 TO X-1.0
210   SOUND S(P) E(P) V(P) M(P) FREQ(T(P))
211   WAIT TIME D(P)*5.0
220   NEXT
221   PRINT CHR$(12):SOUND OFF :WAIT TIME 10
225   CURSOR 10,10
226   PRINT "AFTER A BOTTLE OF WHISKY ......."
230   FOR P=1.0 TO X-1.0
240   SOUND S(P) E(P) V(P) M(P) FREQ(T(P)+RND(15.0))
241   WAIT TIME D(P)*5.0:NEXT
250   SOUND OFF :PRINT CHR$(12):POKE #7921,#56
251   CURSOR 2,10:PRINT "THANK YOU !"
300   DATA 0,1,D2,15,2,0,0,1,E2,15,2,0,0,1,C2,15,2,0
301   DATA 0,1,A1,15,4,0,0,1,B1,15,2,0,0,1,G1,15,4,0
302   DATA 2,1,D1,10,2,2,2,1,E1,10,2,0
303   DATA 2,1,C1,10,2,0,2,1,A,10,4,0,2,1,B,10,2,0
304   DATA 2,1,G,10,4,0
305   DATA 1,1,D,15,2,0,1,1,E,15,2,0,1,1,C,15,2,0
306   DATA 1,1,A0,15,4,0,1,1,B0,15,2,0,1,1,A0,15,2,0
307   DATA 1,1,G0+,15,2,0,1,1,G0,15,8,0
308   DATA 0,0,G,15,0,0,2,0,B,15,0,0,1,0,G1,15,4,0
309   DATA 0,0,O,0,0,0,1,0,O,0,0,0,2,0,O,0,0,0
310   DATA 0,0,D,10,2,0,0,0,D+,10,2,2,0,0,E,10,2,0
311   DATA 0,0,C1,10,5,0,0,0,E,10,2,0,0,0,C1,10,5,0
312   DATA 0,0,E,10,2,0,0,0,C1,10,8,0
313   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
314   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
315   DATA 0,0,D2+,12,0,0,2,0,F1+,12,2,0
```



```
316   DATA 0,0,E2,15,0,0,2,0,G1,15,2,0
317   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
318   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
319   DATA 0,0,E2,12,0,0,2,0,G1,12,4,0
320   DATA 0,0,B1,12,0,0,2,0,D1,12,2,0
321   DATA 0,0,D2,12,0,0,2,0,F1,12,4,0
322   DATA 0,0,C2,12,0,0,2,0,E1,12,8,0
323   DATA 2,0,O,0,0,0
324   DATA 0,0,D,12,2,0,0,0,D+,12,2,0
325   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
326   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
327   DATA 0,0,E,12,2,0,0,0,C1,12,10,0
328   DATA 0,0,A1,12,2,0,0,0,G1,12,2,0
329   DATA 0,0,F1+,12,0,0,2,0,C1,12,2,0
330   DATA 0,0,A1,12,2,0
331   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
332   DATA 0,0,E2,12,0,0,2,0,F1+,12,0,0,1,0,D0,12,3,0
333   DATA 0,0,D2,12,2,0,0,0,C2,12,2,0,0,0,A1,12,2,0
334   DATA 0,0,D2,12,0,0,2,0,F1,12,0,0,1,0,G0,12,8,0
335   DATA 0,0,O,0,0,0,1,0,O,0,0,0,2,0,O,0,0,0
336   DATA 0,0,D,12,2,0,0,0,D+,12,2,0
337   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
338   DATA 0,0,E,12,2,0,0,0,C1,12,5,0
339   DATA 0,0,E,12,2,0,0,0,C1,12,8,0
340   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
341   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
342   DATA 0,0,D2+,12,0,0,2,0,F1+,12,2,0
343   DATA 0,0,E2,12,0,0,2,0,G1,12,2,0
344   DATA 0,0,C2,12,0,0,2,0,E2,12,2,0
345   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
346   DATA 0,0,E2,12,0,0,2,0,G1,12,3,0
347   DATA 0,0,B1,12,0,0,2,0,D1,12,2,0
348   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
349   DATA 0,0,C2,12,0,0,2,0,E1,12,4,0
350   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
351   DATA 00,0,D2,12,0,0,2,0,F1,12,2,0
352   DATA 1,1,C,15,0,0,0,0,E2,12,0,0,2,0,G1,12,2,0
353   DATA 0,0,C2,12,0,0,2,0,E1,12,2,0
354   DATA 0,0,D2,12,0,0,2,0,F1,12,2,0
355   DATA 1,1,A0+,15,0,0,0,0,E2,12,0,0,2,0,G1,12,3,0
356   DATA 0,0,C2,12,0,0,2,0,G1,12,2,0
357   DATA 0,0,D2,12,0,0,2,0,G1,12,2,0
358   DATA 0,0,C2,12,0,0,2,0,G1,12,2,0
359   DATA 1,1,A0,15,0,0,0,0,E2,12,0,0,2,0,A1,12,2,0
360   DATA 0,0,C2,12,0,0,2,0,C2,12,2,0
361   DATA 0,0,D2,12,0,0,2,0,A1,12,2,0
362   DATA 1,1,G0+,15,0,0,0,0,E2,12,0,0,2,0,G1+,12,3,0
363   DATA 0,0,C2,12,0,0,2,0,A1,12,2,0
364   DATA 0,0,D2,12,0,0,2,0,A1,12,2,0
365   DATA 0,0,C2,12,0,0,2,0,A1,12,2,0
366   DATA 1,1,G0,15,0,0,0,0,E2,12,0,0,2,0,G1,12,2,0
367   DATA 0,0,C2,12,0,0,2,0,E1,12,0,0
368   DATA 0,0,D2,1,0,0,2,0,F1,12,2,0
369   DATA 1,1,G0,15,0,0,0,0,E2,12,0,0,2,0,G1,12,3,0
370   DATA 0,0,B1,12,0,0,2,0,D1,12,2,0
371   DATA 0,0,D2,12,0,0,2,0,F1,12,4,0
372   DATA 1,1,C0,15,0,0,0,0,C2,12,0,0,2,0,E1,12,4,0
1000  DATA 999
```

N E   A R M   B A N D I T

======================

```
3    MODE 5A
2    COLORG 12 12 12 12
4    COLORT 12 0 0 0
3    CURSOR 0,3:PRINT "       pralines          PRESS ANY KEY       pralines":
3    CURSOR 0,2:PRINT "   red   red   red   = 10      WIN          x     x     -
0    CURSOR 0,1:PRINT "    x     x     x    =  3                   -     x     x     =
9    CURSOR 28,1:PRINT "$";:CURSOR 28,1
00   Q%=64:GOSUB 1000
10   Q%=160:GOSUB 1000
20   Q%=256:GOSUB 1000
40   CURSOR 25,1:PRINT "          ";
41   CURSOR 28,1:PRINT "$";:CURSOR 28,1
42   A=GETC:IF A=0.0 GOTO 142
43   FOR Z=0.0 TO 15.0
44   Z1%=1+Z/6
45   ON Z1% GOTO 150,160,170
50   Q%=64:GOSUB 900
55   NOE=K
60   Q%=160:GOSUB 900
65   TWO=K
70   Q%=256:GOSUB 900
72   TRE=K
75   NEXT Z
78   GOSUB 1500
80   CURSOR 25,1:PRINT "pralines";:CURSOR 27,0:PRINT WINS%;" ";
82   WAIT TIME 100:GOTO 140
00   K=INT(RND(16.0))
10   IF K=8.0 GOTO 900
30   FILL Q%-8,90 Q%+7,130 K
70   RETURN
000  FILL Q%-32,42 Q%+31,170 0
001  FILL Q%-24,74 Q%+23,138 8
002  RETURN
500  IF NOE=3 AND TWO=3 AND TRE=3 THEN WINS%=10:RETURN
510  IF NOE=TWO AND NOE=TRE THEN WINS%=3:RETURN
520  IF NOE=TWO THEN WINS%=1:RETURN
530  IF TWO=TRE THEN WINS%=1:RETURN
540  WINS%=0:RETURN
```

D E S S I N S   A V E C   C U R S E U R

==================================

```
1      PRINT CHR$(12)
2      GOSUB 400
5      MODE 3
10     A=GETC
12     IF A=32.0 THEN 200
13     IF A=8.0 THEN 220
14     IF A=9.0 THEN 320
15     IF A<16.0 OR A>19.0 THEN 321
100    V=V+1.0:IF V>VMAX THEN V=VMAX
105    RETURN
110    V=V-1.0:IF V<0.0 THEN V=0.0
115    RETURN
120    X=X-1.0:IF X<0.0 THEN X=0.0
125    RETURN
130    X=X+1.0:IF X>XMAX THEN X=XMAX
135    RETURN
200    MODE 0:MODE 3:V=0.0:X=0.0
210    GOTO 5
220    A=GETC:DOT X,V 15
221    IF A=32.0 GOTO 200
222    IF A=9.0 GOTO 320
223    IF A<16.0 OR A>19.0 THEN 220
224    DOT X,V 0:A=A-15.0:ON A GOSUB 100,110,120,130
225    GOTO 220
320    A=GETC:DOT X,V 0
321    IF A=8.0 GOTO 220
322    IF A=32.0 GOTO 200
323    IF A<16.0 OR A>19.0 THEN 320
329    DOT X,V 15:A=A-15.0:ON A GOSUB 100,110,120,130
330    GOTO 320
400    PRINT :PRINT
412    PRINT "LES DESSINS  S'OBTIENNENT  EN PRESSANT";
413    PRINT " UNE DES FLECHES":PRINT "          ";
430    PRINT "DANS LA DIRECTION QUI VOUS CONVIENT.":PRINT
432    PRINT " POUR EFFACER UN MORCEAU DE DESSIN ";
440    PRINT " REPLACEZ LE CURSEUR":PRINT "    ";
441    PRINT " A CET ENDROIT APRES AVOIR PRESSE";
442    PRINT " SUR CHAR DEL.":PRINT :PRINT "     ";
444    PRINT "POUR REPASSER EN MODE DESSIN";
445    PRINT " PRESSEZ SUR TAB":PRINT
470    PRINT "L'EFFACAGE  DE L'ECRAN  S'OBTIENT ";
480    PRINT " EN  PRESSANT LA BARRE"
481    PRINT "                  D'ESPACEMENT"
490    PRINT :PRINT
491    INPUT "PRESSEZ LU ET RETURN APRES AVOIR FINI":Z$
492    IF LEFT$(Z$,1)="L" THEN 499
493    PRINT :GOTO 491
499    PRINT CHR$(12)
500    RETURN
```

GRAFTEXT SUBDEMO

```
1      CLEAR 1400
2      REM :DATA FOR GOSUB40040: X / Y / C / VFLAG / A$ / F
3      REM !!!! DELETE LINE 40 >>>>>> 70 !!!!!!!!!!!!!!!!!!!!
5      COLORG 8 1 3 5
10     MODE 5
20     COLORG 8 0 14 1
30     GOSUB 40012:FOR X=0.0 TO XMAX:DOT X,225+20*SIN(X/20.0) 15:NEXT
31     FOR X=200.0 TO 230.0 STEP 3.0:DRAW X,10 X,45 0:NEXT
32     FOR Y=125.0 TO 150.0 STEP 2.0:FILL 260,Y XMAX,Y+1 Q:Q=Q+1.0:NEXT
33     X=10.0:Y=215.0:C=1.0:A$="DAI":VFLAG=0.0:F=2.0:GOSUB 40040
34     X=80.0:Y=215.0:C=6.0:A$="TEXT":GOSUB 40040
35     X=150.0:Y=215.0:C=5.0:A$="IN":GOSUB 40040
36     X=200.0:Y=215.0:C=0.0:F=2.0:A$="GRAFICS":GOSUB 40040
39     X=130.0:Y=190.0:C=2.0:F=1.0:A$="TEL. 02 / 3751114":GOSUB 40040
40     X=10.0:Y=200.0:C=0.0
41     A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ!#?$%&'()*=:-+;<>./1234567890"
50     GOSUB 40040
55     X=10.0:Y=170.0:C=3.0:F=2.0:GOSUB 40040
56     X=XMAX-10.0:Y=50.0:C=13.0:VFLAG=1.0:F=1.0:GOSUB 40040
60     VFLAG=0.0:X=10.0:Y=90.0:C=12.0:F=4.0:A$=LEFT$(A$,26):GOSUB 40040
65     GOTO 65
40012 DIM CAR$(90.0)
40021 FOR Z=32.0 TO 90.0:READ A$
40022 IF A$="STOP" THEN RETURN
40023 READ CAR$(Z):NEXT:RETURN
40040 X1=X:Y1=Y:IF F=0.0 THEN F=1.0
40041 FOR M=0.0 TO LEN(A$)-1.0
40042 T$=MID$(A$,M,1)
40050 GR$=CAR$(ASC(T$))
40060 FOR N=0.0 TO LEN(GR$)-1.0 STEP 4.0
40065 IF VFLAG=1.0 GOTO 40120
40070 IF MID$(GR$,N,1)="/" THEN X=X+(8.0*F):GOTO 40100
40080 ZZ=VAL(MID$(GR$,N,1)):YY=VAL(MID$(GR$,N+1,1))
40082 JC5%=X+ZZ*F:JC6%=Y+VAL(MID$(GR$,N+1,1))*F
40083 JC7%=X+VAL(MID$(GR$,N+2,1))*F:JC8%=Y+VAL(MID$(GR$,N+3,1))*F
40084 DRAW JC5%,JC6% JC7%,JC8% C
40085 IF F<1.5 THEN GOTO 40090
40086 JC9%=X+1+VAL(MID$(GR$,N+2,1))*F
40087 JC10%=Y+1+VAL(MID$(GR$,N+3,1))*F
40088 DRAW X+1+ZZ*F,Y+1+YY*F JC9%,JC10% C
40090 NEXT N
40100 IF X+8.0*F>=XMAX THEN X=X1:Y=Y-10.0*F
40102 NEXT M
40103 RETURN
40120 IF MID$(GR$,N,1)="/" THEN Y=Y-9.0*F:GOTO 40180
40130 JC1%=X+VAL(MID$(GR$,N+1,1))*F:JC2%=Y-VAL(MID$(GR$,N,1))*F
40131 JC3%=X+VAL(MID$(GR$,N+3,1))*F:JC4%=Y-VAL(MID$(GR$,N+2,1))*F
40132 DRAW JC1%,JC2% JC3%,JC4% C
40140 NEXT N
40180 IF Y-9.0*F<=0.0 THEN Y=Y1:X=X-9.0*F
40190 NEXT M
40200 RETURN
50000 DATA BLANCO,/,UITROEP!,31313337/,QOUTES,25274547/,#
50001 DATA 1353155521274147/,$,124242532444152626563137/
50010 DATA %,17271626125641514252/,&,121321315331155116273536/,'
50011 DATA 3537/,(,131513311537/
50020 DATA ),315353555537/,*,125616523137/,+,32361454/,COMMA
50021 DATA 21323233/
```



```
50030 DATA -,1454/,.,31423241/,/,1256/,0,12162141525627471256/
50040 DATA 1,214131372637/,2,1151123344445556647271627/,3
50041 DATA 12212141525333456175744 53/,4,414713531447/
50050 DATA 5,12212141525415451517175 7/,6,21411215144452531537 3757/
50051 DATA 7,212223561757/,8,21412444274712131516525 35556/
50060 DATA 9,11313153535624541516274 7/,:,33333535/,;,21323233 3535/
50061 DATA <,14471441/
50070 DATA =,13531555/,>,21545427/,?,16272747343331313456/,APE,/
50080 DATA A,1115515513531537 3755/,B,111717471444114152535556/,C
50081 DATA 1216274747562141 4152/,D,1117114152561747/
50090 DATA E,1117115114441757/,F,111714441757/,G,12162757215151535343/
50091 DATA H,111714545157/
50100 DATA I,214131372747/,J,122121415257/,K,111713572451/,L,11171151/
50110 DATA M,11171735353435575751/,N,111751571652/,O,1216274756522141/,P
50111 DATA 1117144417475556/
50120 DATA Q,121627475653213133 51/,R,1117174756551444 2451/,S
50121 DATA 122121415253244415162747 4756/,T,17573137/
50130 DATA U,111721415157/,V,13175357133131 53/,W,1117515711333351 3334/
50131 DATA X,111217165152575612561652/
50140 DATA Y,161756571634345631 34/,Z,175712561151/
51000 DATA STOP
```

TRAFFICTEST


```
1     COLORG 8 1 3 5:MODE 5
2     ENVELOPE 1 15,10:0,10:
10    CLEAR 2000
30    GOSUB 40012
35    X=50.0:Y=230.0:C=14.0:F=1.5
36    A$="DAI TRAFFIC TEST":GOSUB 40040
110   DRAW 50,220 235,220 0
112   DRAW 0,170 280,170 0
115   P=170.0
120   READ A
125   IF A=999.0 THEN GOTO 140
130   READ B,C,D:DRAW A+50,B C+50,D 0:GOTO 120
140   A$="STOP FOR THE RED LIGHT":X=130.0:Y=80.0
141   C=3.0:F=1.0:GOSUB 40040
150   A$="NO REACTION ON GREEN !!":X=130.0:Y=60.0
151   C=5.0:F=1.0:GOSUB 40040
160   WAIT TIME 200:FILL 130,0 XMAX,100 8
200   REM TEST
210   C=INT(RND(2.0)):CO=3.0:IF C=1.0 THEN CO=5.0
215   SOUND 2 1 10 0 FREQ(800.0):WAIT TIME 20:SOUND OFF
220   WAIT TIME RND(50.0)
230   IF CO=3.0 THEN FILL 57,112 73,128 CO
235   IF CO=5 THEN FILL 57,87 73,103 5
237   IF CO=5 THEN GOTO 700
240   S=S+1.0:IF GETC=0.0 GOTO 240
250   FOR X=0.0 TO 250.0-S*2.0 STEP 3.0
251   FILL 300,X 310,X+1 1:SOUND 1 0 5 0 FREQ(31.0+X)
260   NEXT
265   SOUND OFF
270   MG=MG+10.0:NG=125.0+70.0-S/2.5
271   IF MG>280.0 THEN A$=" THE END":F=2.0:X=140.0:GOSUB 40040
272   IF MG>280.0 THEN WAIT TIME 1000:GOTO 1
275   IF NG<125.0 THEN NG=125.0
280   DRAW O,P MG,NG 15
290   O=MG:P=NG
295   S=S*1.5
300   IF S>=100.0 THEN A$=" WAKE UP !!          "
305   IF S>150.0 THEN A$=" YOU ARE SLOW !      "
310   IF S<100.0 THEN A$=" ATTENTION PLEASE !"
320   IF S<90.0 THEN A$=" NOT GOOD!           "
330   IF S<80.0 THEN A$=" MMMM...             "
340   IF S<70.0 THEN A$=" GOOD                "
350   IF S<60.0 THEN A$=" VERY GOOD!          "
360   IF S<50.0 THEN A$=" EXCELLENT !         "
370   IF S<40.0 THEN A$=" SUPERB !            "
380   IF S<30.0 THEN A$=" MARVELLOUS !        "
390   IF S<20.0 THEN A$=" GENIUS !            "
400   X=150.0:Y=50.0:C=3.0:F=1.0:GOSUB 40040
490   WAIT TIME 50
491   FILL 57,112 73,128 8:FILL 57,87 73,103 8
495   FILL 300,100 XMAX,YMAX 8
496   FILL 100,0 XMAX,100 8
```



```
506   S=0.0
510   GOTO 200
700   FOR X=0.0 TO 200.0:IF GETC<>0.0 THEN GOTO 710
705   NEXT:GOTO 490
710   FOR X=0.0 TO 10.0:SOUND 1 0 10 0 FREQ(1000.0)
711   SOUND 1 0 12 2 FREQ(500.0):WAIT TIME 10:NEXT
715   MG=MG+10.0:IF NG<125.0 THEN NG=125.0
716   DRAW O,P MG,NG 5:O=MG:P=NG
720   SOUND OFF :X=150.0:Y=80.0:C=5.0:F=1.5
721   A$="GREEN !":GOSUB 40040:GOTO 490
1000  GOTO 1000
40012 DIM CAR$(90.0)
40021 FOR Z=32.0 TO 90.0:READ A$
40022 IF A$="STOP" THEN RETURN
40023 READ CAR$(Z):NEXT:RETURN
40040 X1=X:IF F=0.0 THEN F=1.0
40041 FOR M=0.0 TO LEN(A$)-1.0
40042 T$=MID$(A$,M,1)
40050 GR$=CAR$(ASC(T$))
40060 FOR N=0.0 TO LEN(GR$)-1.0 STEP 4.0
40065 IF VFLAG=1.0 GOTO 40120
40070 IF MID$(GR$,N,1)="/" THEN X=X+(8.0*F):GOTO 40100
40080 JC1%=X+VAL(MID$(GR$,N,1))*F:JC2%=Y+VAL(MID$(GR$,N+1,1))*F
40081 JC3%=X+VAL(MID$(GR$,N+2,1))*F:JC4%=Y+VAL(MID$(GR$,N+3,1))*F
40082 DRAW JC1%,JC2% JC3%,JC4% C
40090 NEXT N
40100 IF X+8.0*F>=XMAX THEN X=X1:Y=Y-10.0*F
40102 NEXT M
40103 RETURN
40120 IF MID$(GR$,N,1)="/" THEN Y=Y-9.0*F:GOTO 40180
40130 JC5%=X+VAL(MID$(GR$,N+1,1))*F:JC6%=Y-VAL(MID$(GR$,N,1))*F
40131 JC7%=X+VAL(MID$(GR$,N+3,1))*F:JC8%=Y-VAL(MID$(GR$,N+2,1))*F
40132 DRAW JC5%,JC6% JC7%,JC8% C
40140 NEXT N
40180 IF Y-9.0*F<=0.0 THEN Y=Y1:X=X-9.0*F
40190 NEXT M
40200 RETURN
50000 DATA BLANCO,/,UITROEP!,31313337/,QOUTES,25274547/,#
50001 DATA 135315552127 4147/,$,12424253244415262656 3137/
50010 DATA %,17271626125641514252/,&,12132131533115511627 3536/,'
50011 DATA 3537/,(,131513311537/
50020 DATA ),315353555537/,*,125616523137/,+,32361454/,COMMA,21323233/
50030 DATA -,1454/,.,31423241/,/,1256/,0,12162141525627471256/
50040 DATA 1,2141313726 37/,2,115112334444555647271627/,3
50041 DATA 12212141525333456175744 53/,4,414713531447/
50050 DATA 5,12212141525415451517 1757/,6,2141121514444525315373757/,7
50051 DATA 212223561757/,8,21412444274712131516525 35556/
50060 DATA 9,11313153535624541516 2747/,:,33333535/,;,21323233 3535/,<
50061 DATA 14471441/
50070 DATA =,13531555/,>,21545427/,?,16272747343331313456/,APE,/
50080 DATA A,11155155135315373755/,B,1117174714441141525 35556/,C
50081 DATA 121627474 '5621414152/,D,1117114152561747/
50090 DATA E,1117115114441757/,F,111714441757/,G,121627572151515 35343/,H
50091 DATA 111714545157/
50100 DATA I,214131372747/,J,122121415257/,K,111713572451/,L,11171151/
50110 DATA M,1117173535343557 5751/,N,111751571652/,O,121627475652 2141/,P
50111 DATA 111714441747 5556/
50120 DATA Q,121627475653213133 51/,R,111717475655144424 51/,S
50121 DATA 1221214152532444151627474756/,T 17573137/
50130 DATA U,111721415157/,V,131753571331 3 53/,W,111751571133335 13334/,X
50131 DATA 11121716515257561256 1652/
50140 DATA Y,16175657163434563134/,Z,175712561151/
```

```
51140 DATA 10,0,10,80,20,0,20,80,25,80,30,85,30,85,30,135,30
51141 DATA 135,25,140,25,140,5,140,5 ,140,0,135,0,135,0,85
51150 DATA 0,85,5,80,999
*
```


===

```
1      GOTO 20
7      GOTO 64000
8      GOTO 64000
9      GOTO 64000
10     GOTO 64000
20     COLORT 8 0 0 8
21     POKE #131,1
22     PRINT CHR$(12)
23     CURSOR 1,20:PRINT "1  CHANGE BACKGROUND COLOUR"
24     CURSOR 31,20:PRINT "6  ANIMATION  / COLORT "
25     CURSOR 1,18:PRINT "2  FLASHING BACKGROUND"
26     CURSOR 31,18:PRINT "7  ............... "
27     CURSOR 1,16:PRINT "3  SCREEN LINE ADDRESS"
28     CURSOR 31,16:PRINT "8  ............... "
29     CURSOR 1,14:PRINT "4  SCREEN CURSOR ADDRESS"
30     CURSOR 31,14:PRINT "9  ............... "
31     CURSOR 1,12:PRINT "5  ANIMATION, COLOURS 1619"
32     CURSOR 30,12:PRINT "10  ............... "
40     CURSOR 30,2:INPUT "WICH PROGRAM ";P$:PRINT
41     IF P$="1" OR P$="2" OR P$="3" OR P$="4" THEN 46
42     IF P$="5" OR P$="6" THEN 46
43     IF P$="7" OR P$="8" OR P$="9" OR P$="10" THEN 64000
44     CURSOR 1,4:PRINT "WRONG INPUT ONLY THE NUMBER OF THE PROGRAM "
45     CURSOR 30,2:PRINT "WICH PROGRAM                    ":GOTO 40
46     P=VAL(P$)
47     ON P GOTO 100,1000,2000,3000,4000,10000,7,8,9,10
100    PRINT CHR$(12):PRINT :PRINT :PRINT
108    LIST 110-170
110    E%=#FF
115    COLORT 0 9 9 0
120    B%=#7FEF
125    FOR A%=0 TO 23
130    D%=B%-3
135    FOR C%=0 TO 65
140    POKE D%,E%
145    D%=D%-2:NEXT
146    RJ%=GETC:IF RJ%=32 GOTO 20
155    B%=B%-#86:NEXT
165    E%= INOT E% IAND #FF
170    GOTO 120
1000   PRINT CHR$(12):A5%=0
1010   FOR A%=0 TO 10
1020   POKE #79E4+2*A%,#FF
1025   POKE #79E4+2*A%+#86,#FF
1030   NEXT
1035   CURSOR 23,12:PRINT "WARNING"
1040   FOR B%=20 TO 1 STEP -1
1043   GOSUB 1200
1045   COLORT 0 9 A5% 15-A5%
1046   GOSUB 1100
1050   WAIT TIME B%
1055   COLORT 0 9 15-A5% A5%
1056   GOSUB 1100
1060   WAIT TIME B%
1065   NEXT
1070   GOTO 1040
1100   RJ%=GETC:IF RJ%<>32 THEN RETURN
```

```
130  PRINT :INPUT "LIST PROGRAM < Y/N > ":RJ$
140  IF RJ$="Y" THEN PRINT CHR$(12):GOSUB 64500:GOTO 20
141  IF RJ$="N" THEN PRINT CHR$(12):PRINT :GOTO 20
145  CURSOR 0,10:PRINT SPC(30):CURSOR 0,11
150  RETURN
200  A5%=A5%+1:IF A5%>15 THEN A5%=0
210  RETURN
000  GOSUB 2100
020  FOR A%=0 TO 23
035  PRINT 23.0-A%:SPC(9-CURX);"# ";HEX$(#7FEA-(#86*A%));
036  PRINT SPC(22-CURX);"# ";HEX$(#7FED-(#86*A%));SPC(37-CURX);
040  PRINT "# ";HEX$(#7F6A-(#86*A%));
041  PRINT SPC(52-CURX);"# "+HEX$(#7F6D-(#86*A%))
045  IF A%=11 THEN GOSUB 2150:GOSUB 2100
050  NEXT:PRINT :GOSUB 2150:GOTO 20
100  PRINT CHR$(12):PRINT
105  PRINT "          # LOCATION                          # LOCATION"
110  PRINT "LINE     COLOR CODE    # LOCATION";
111  PRINT "     COLOR CODE    # LOCATION"
120  PRINT "NUMBER   BEGIN LINE    BEGIN LINE";
121  PRINT "     END LINE      END LINE"
125  PRINT
130  RETURN
150  RJ%=GETC:IF RJ%<>32 GOTO 2150
160  RETURN
000  PRINT CHR$(12):PRINT :PRINT "CHARACTERS FROM  <-2 TO 61 > "
002  PRINT "LINES FROM         < 0 TO 23 > ":PRINT
003  PRINT "INPUT CURSOR EXAMPLE 31,12 FOR CENTER OF SCREEN":PRINT
004  INPUT "INPUT CURSOR ";B1%,A1%:PRINT :PRINT
005  IF A1%<0.0 OR B1%>61.0 OR A1%>23.0 THEN PRINT "WRONG INPUT":PRINT :GO
009  B1%=B1%+3
010  PRINT "POKE # ";HEX$((#7FEA-(#86*(23-A1%)))-((B1%*2)));" TO CHANGE CO
020  PRINT "POKE # ";HEX$((#7FED-(#86*(23-A1%)))-((B1%*2)));" TO CHANGE CH
030  PRINT :PRINT
035  PRINT "FOR OTHERS PRESS RETURN ,FOR OTHER PROGRAMS SPACE BAR"
040  RJ%=GETC:IF RJ%=32 GOTO 20
045  IF RJ%=0 GOTO 3040
050  GOTO 3004
4000  MODE 4
4110  FOR B=0.0 TO 2.0*PI STEP 0.2
4120  A=B-0.2:B%=16:GOSUB 4220
4130  A=B:B%=17:GOSUB 4220
4140  COLORG 0 10 0 10
4150  A=B-0.1:B%=18:GOSUB 4220
4160  A=B+0.1:B%=19:GOSUB 4220
4170  COLORG 0 0 10 10
4190  NEXT
4190  A=B-0.2:B%=16:GOSUB 4220
200  A=B-0.1:B%=18:GOSUB 4220
210  GOTO 4110
220  X%=XMAX/2+30*SIN(A)
4230  Y%=YMAX/2+30*COS(A)
240  DRAW XMAX/2,YMAX/2 X%,Y% B%
245  RJ%=GETC:IF RJ%=32.0 THEN MODE 0:GOTO 20
250  RETURN
0000 MODE 0:COLORT 8 0 0 8
0010 PRINT CHR$(12.0)
0020 A%=#7A28-2:B%=#79A8+2
0030 FOR C%=A% TO B% STEP -2
0040 POKE C%,#FF
```



```
10041 REM POKE C-2,#FF
10042 WAIT TIME 1:POKE C%+2,#0
10050 NEXT:POKE C%,#0
10060 FOR C%=B% TO A% STEP 2
10070 POKE C%,#FF:POKE C%-2,#0
10080 NEXT:POKE C%,#0
10090 JCC%=GETC:IF JCC%>0 GOTO 1
10100 GOTO 10030
64000 P%=P
64005 CURSOR 1,4:PRINT "
64006 PRINT "                          "
64010 CURSOR 1,4:PRINT "NO PROGRAM IN";P%
64020 GOTO 45
64500 PRINT :LIST 1000-1070:GOSUB 2150:RETURN
*
```

B I O R Y T H M

================

```
90    CLEAR 1000
95    PRINT CHR$(12)
100   DIM X$(31.0):DIM M$(12.0)
110   M$(1.0)="JAN"
111   M$(2.0)="FEB"
112   M$(3.0)="MAR"
113   M$(4.0)="APR"
114   M$(5.0)="MAI"
115   M$(6.0)="JUN"
116   M$(7.0)="JUL"
117   M$(8.0)="AUG"
118   M$(9.0)="SEP"
119   M$(11.0)="NOV"
120   M$(12.0)="DEC"
121   M$(10.0)="OCT"
200   P9=6.28318
210   P1=23.0:P2=28.0:P3=33.0
220   D1=P9/P1:D2=P9/P2:D3=P9/P3
230   DATA 31,28,31,30,31,30,31,31,30,31,30,31
300   INPUT "YOUR NAME PLEASE ";N$
311   PRINT
312   PRINT "BIORYTHM OF YEAR OR MONTH ";
313   INPUT X$
320   IF X$<>"YEAR" AND X$<>"MONTH" THEN GOTO 311
330   N1=0.0
340   GOSUB 8000
360   IF B1>2.0 THEN GOTO 400
370   IF B1=2.0 THEN IF B2=29.0 THEN GOTO 400
380   R=(B3-1900.0)/4.0
381   IF INT(R)<>R THEN GOTO 400
390   N1=1.0
400   GOSUB 8500
420   FOR J=1.0 TO B1
430   READ X
440   NEXT J
450   N1=N1+X-B2
460   IF B1=12.0 THEN GOTO 510
470   FOR J=B1+1.0 TO 12.0
480   READ X
490   N1=N1+X
500   NEXT J
510   IF C3-B3<2.0 THEN GOTO 560
520   FOR J=B3-1899.0 TO C3-1901.0
530   IF INT(J/4.0)=J/4.0 THEN N1=N1+1.0
540   N1=N1+365.0
550   NEXT J
560   RESTORE
570   IF C1=1.0 THEN GOTO 620
580   FOR J=1.0 TO C1-1.0
590   READ X
600   N1=N1+X
610   NEXT J
620   T=(C3-1900.0)/4.0
621   IF INT(T)<>T THEN GOTO 640
630   IF C1>2.0 THEN N1=N1+1.0
640   I1=N1:I2=N1:I3=N1
```



```
650   READ X
655   PRINT CHR$(12)
660   PRINT "  BIORYTHMIC CHART  ";N$
665   PRINT :PRINT
667   B2%=B2:B1%=B1:B3%=B3
670   PRINT "DATE OF BIRTH";B2%;" ";B1%;" ";B3%
680   PRINT :PRINT :PRINT
690   PRINT "I=INTELLIGENCE"
700   PRINT "P=PHYSICAL"
710   PRINT "E=EMOTIONNAL"
720   L=0.0
730   GOSUB 2000
740   D=0.0
745   L=L+1.0
750   FOR I=1.0 TO 31.0
760   X$(I)=" "
770   NEXT I
780   X$(16.0)=":"
800   V1=INT(15.0*SIN((L+I1)*D1)+16.5)
810   V2=INT(15.0*SIN((L+I2)*D2)+16.5)
820   V3=INT(15.0*SIN((L+I3)*D3)+16.5)
830   X$(V1)="P"
840   X$(V2)="E"
850   X$(V3)="I"
860   IF V1=V2 THEN X$(V1)="*"
870   IF V2=V3 THEN X$(V3)="*"
880   IF V1=V3 THEN X$(V1)="*"
890   D=D+1.0
900   IF D<X+1.0 THEN GOTO 1020
910   S1=S1+1.0
920   IF S1=12.0 THEN GOTO 1500
930   C1=C1+1.0
940   IF C1>12.0 THEN GOTO 980
950   READ X
955   IF X9=1.0 THEN GOTO 1500
960   GOSUB 3000
970   GOTO 1020
980   RESTORE
990   C1=1.0
1000  C3=C3+1.0
1010  GOTO 950
1020  D%=D
1021  IF D<10.0 THEN 1023
1022  PRINT M$(C1);" ";D%;"    ";:GOTO 1025
1023  PRINT M$(C1);"  ";D%;"    ";
1025  Y$=" "
1030  FOR J=1.0 TO 31.0
1050  Y$=Y$+X$(J)
1055  NEXT J
```

```
1056  PRINT Y$
1060  GOTO 745
1500  STOP
2000  IF X$="MONTH" THEN X9=1.0
2020  PRINT :PRINT " BIORYTHMIC CHART OF ";N$;:C3%=C3
2022  PRINT " FOR ";M$(C1);"  ";C3%
2030  PRINT
2040  PRINT "                        ";"(-)";
2045  PRINT "                        ";"(+)"
2050  PRINT
2060  D=1.0
2070  RETURN
3000  IF X$="MONTH" THEN X9=1.0
3002  PRINT
3004  D=1.0
3010  RETURN
8000  PRINT :PRINT "MONTH,DAY,YEAR OF BIRTH"
8002  PRINT "EXAMPLE   BIRTH ON 3D MAY 1942"
8003  PRINT "PRESS 5  RETURN 3  RETURN 1942"
8015  INPUT B1,B2,B3
8020  RETURN
8500  PRINT
8501  PRINT " GIVE MONTH OND YEAR FOR THE BIORYTHM"
8502  PRINT "EX FOR AND STARTING ON JANUARY    1980"
8503  PRINT "PRESS 1 RETURN 1980 RETURN"
8508  INPUT C1,C3
8510  IF B3>=C3 THEN GOTO 90
8520  RETURN
*
```


SIMON


```
1     MODE 3A:BST=0.0:CNT=0.0
2     CURSOR 0,3:PRINT "        LAST PLAY";
3     CURSOR 40,3:PRINT "BEST RESULT";
4     GOSUB 5000
10    REM CLEAR 1000
15    ENVELOPE 0 3,10:3,10:3,10:0
20    DIM A(4.0):DIM B(4.0)
30    A(1.0)=40.0:B(1.0)=40.0:A(2.0)=70.0
35    B(2.0)=70.0:A(3.0)=100.0:B(3.0)=40.0
37    A(4.0)=70.0:B(4.0)=10.0
40    DIM TUNE(100.0)
70    DIM NOTE(4.0)
90    NOTE(4.0)=262.0:NOTE(1.0)=330.0:NOTE(3.0)=392.0:NOTE(2.0)=523.0
100   DIM COLOR(4.0)
110   COLOR(1.0)=1.0:COLOR(2.0)=5.0:COLOR(3.0)=7.0:COLOR(4.0)=11.0
450   CNT=0.0
480   CNT=CNT+1.0
490   TUNE(CNT)=INT(RND(4.0))+1.0
500   WAIT TIME 30
520   FOR I=1.0 TO CNT
530   PLAY=TUNE(I)
540   GOSUB 2000
560   NEXT I
590   I=0.0
600   I=I+1.0
610   IF I<=CNT THEN 635
620   GOTO 480
635   GOSUB 6000
640   GOSUB 2000
645   IF BST<CNT THEN BST=CNT
650   IF PLAY=TUNE(I) THEN 600
670   GOSUB 5000
760   CURSOR 22,2:PRINT "PLAY BROKEN";:WAIT TIME 75
761   CURSOR 22,2:PRINT "                            ";:CURSOR.44,2
770   IF BST>CNT THEN GOSUB 5010
771   GOTO 10
2000  SOUND 0 0 10 0 FREQ(NOTE(PLAY))
2020  SOUND 2 0 10 2 FREQ(NOTE(PLAY)*4.0)
2040  FILL A(PLAY).B(PLAY) A(PLAY)+20.0,B(PLAY)+20.0 COLOR(PLAY)
3000  WAIT TIME 20
3050  SOUND OFF
4040  FILL A(PLAY).B(PLAY) A(PLAY)+20.0,B(PLAY)+20.0 0
4100  RETURN
5000  CURSOR 10,2:CNT%=CNT:PRINT CNT%;:PRINT "   ";
5010  CURSOR 44,2:BST%=BST:PRINT BST%;:PRINT "
5015  CURSOR 44,2
5020. RETURN
6000  WAIT TIME 5:G=GETC:IF G=0.0 GOTO 6000
6050  IF G=18.0 THEN PLAY=1.0
6060  IF G=16.0 THEN PLAY=2.0
6070  IF G=19.0 THEN PLAY=3.0
6080  IF G=17.0 THEN PLAY=4.0
6120  RETURN
```

*LOAD

PADDLE SOUND

```
1     REM MAKE SOUND WITH BOTH PADDLES
5     ENVELOPE 0 16
10    P=PDL(0):Q=PDL(2):R=PDL(3)
20    IF P>3.0 OR Q>31.0 THEN SOUND 1 0 R*3/52 0 FREQ(P*12.0+Q)
40    S=PDL(1):T=PDL(4):U=PDL(5)
50    IF S>3.0 OR T>31 THEN SOUND 2 0 U*3/52 0 FREQ(S*12.0+T)
90    GOTO 10
```

RANDOM POS TEST

```
1     MODE 0
2     COLORS 7 0 15 4
4     INPUT "TYPE H OR S . FOR HARDWARE OR SOFTWARE";RNT$
6     W%=1
7     MODE 4
10    DIM A%(XMAX)
15    IF RNT$="S" THEN K=RND(XMAX+1.0):GOTO 21
16    IF RNT$="H" THEN K=RND(0.0)*(XMAX+1.0):GOTO 21
20    GOTO 4
21    R=R+K
22    S%=S%+1
30    A%(K)=A%(K)+1.0
40    O%=A%(K)
50    P%=O%/W%
60    IF P%*W%<>O% THEN 20
69    IF P%>=YMAX+1 THEN DOT XMAX,0 14:GOTO 69
70    DOT K,P% 15
75    DOT T%,0 7
90    T%=(R/S%-((XMAX+1)*0.495))*100
91    IF T%<0 THEN T%=0
92    IF T%>XMAX THEN T%=XMAX
93    DOT T%,0 0
999   GOTO 15
```


LANDSCAPE V2

```
5     ENVELOPE 0 5,10:2,5:4,15:0
6     ENVELOPE 1 10,5:15,2:5,3:0
10    MODE 5:FLAG9%=0
20    FILL 0,0 XMAX,50 5
30    FILL 0,50 XMAX,YMAX 12
50    DRAW 0,0 150,50 0
60    DRAW 150,50 XMAX,0 0
70    FOR X=0.0 TO 2.0*PI STEP 0.1
80    DRAW 250,150 250+30*COS(X),150+30*SIN(X) 14
90    NEXT
95    GOSUB 1000
165   NOISE 1 15
166   WAIT TIME 3
170   FILL A,50 A+10,60 0
180   FILL A,50 A+1,60 12
185   NOISE 1 15
190   FILL A+10,50 A+11,60 0
195   IF A>50.0 GOTO 210
200   A=A+1.0:GOTO 165
210   FOR X=0.0 TO PI STEP 5E-2
220   DOT 150+50*COS(X),50+50*SIN(X) 0
225   SOUND 1 0 10 0 FREQ(X*100.0+31.0)
230   NEXT
240   A=150.0:B=150.0:C=50.0
250   FILL A,50 B,C 11
260   A=A-1.0:B=B+1.0:C=C+1.0
270   IF A<120.0 GOTO 300
280   GOTO 250
300   SOUND 1 0 15 2 FREQ(2000.0)
310   WAIT TIME 5
320   SOUND 1 0 10 2 FREQ(31.0)
325   NOISE 1 15
330   WAIT TIME 1
340   SOUND 1 0 15 2 FREQ(330.0)
350   SOUND 0 0 15 2 FREQ(440.0)
355   SOUND 2 0 15 2 FREQ(523.0)
360   WAIT TIME 100
370   SOUND 0 0 15 2 FREQ(370.0)
380   WAIT TIME 100
390   SOUND 0 0 15 2 FREQ(415.0)
400   SOUND 2 0 15 2 FREQ(494.0)
450   WAIT TIME 50
500   SOUND 1 0 15 2 FREQ(1318.0)
515   WAIT TIME 100
516   SOUND OFF
520   SOUND 1 0 10 0 FREQ(247.0)
530   WAIT TIME 13
```

```
540   SOUND 1 0 10 0 FREQ(277.0)
550   WAIT TIME 20
560   SOUND 1 0 10 0 FREQ(247.0)
570   WAIT TIME 13
590   SOUND 1 0 10 0 FREQ(208.0)
595   SOUND 1 0 5 0 FREQ(165.0)
600   WAIT TIME 20:SOUND OFF
610   FOR X=0.0 TO 200.0
620   DOT RND(XMAX),(50+RND(YMAX-50.0)) 15
621   NOISE 0 10
625   SOUND 1 0 1 0 FREQ(RND(1000.0)+31.0):WAIT TIME 1:SOUND OFF
626   NOISE OFF
630   NEXT
650   FLAG9%=1
1000  FOR X=0.0 TO 100.0
1100  DRAW 50+A,100 55+A,95 0
1110  DRAW 55+A,95 60+A,100 0
1120  DRAW 50+A,100 55+A,95 12
1130  DRAW 55+A,95 60+A,100 12
1140  DRAW 50+A,95 60+A,95 0
1150  DRAW 50+A,95 60+A,95 12:A=RND(50.0)
1155  SOUND 1 0 3 3 FREQ(3000.0+RND(1000.0))
1156  WAIT TIME 1:SOUND OFF
1160  NEXT X
1170  IF FLAG9%=1 GOTO 1000
1200  RETURN
*
```

*

POLYGONS

```
1     CLEAR 5000
5     INPUT "How many sides ";N
6     PRINT :INPUT "Radius (between 4 and 120) ";R
10    MODE 5
60    DIM B(N),C(N)
90    P1=2.0*PI/N
100   FOR I=1.0 TO N
110   B(I)=R+10.0+R*COS((I-1.0)*P1)
120   C(I)=R+10.0+R*SIN((I-1.0)*P1)
130   NEXT I
140   FOR I=1.0 TO N
150   FOR J=1.0 TO N
160   DRAW B(I),C(I) B(J),C(J) 15
170   NEXT J:NEXT I
180   WAIT TIME 100:GOTO 5
*
```


MUSIC V2

```
5     DIM F(20.0)
6     ENVELOPE 0 15,3:7,5:3,10:0
10    FOR N=1.0 TO 17.0:READ F(N):NEXT
15    FOR JCC%=1 TO 27
20    READ N,L
30    A=F(N):GOSUB 100:WAIT TIME L
35    NEXT
41    RESTORE:GOTO 10
100   SOUND 0 0 15 0 FREQ(A)
200   SOUND 1 0 15 0 FREQ(A*2.0)
300   SOUND 2 0 10 0 FREQ(A*4.0)
301   RETURN
1000  DATA 262,277,294,311,330,349,370,392,415,440,466
1005  DATA 494,523,554,587,622,659
1010  DATA 1,5,5,5,9,5,13,10,12,5,13,5,15,5,17,10,13,5
1020  DATA 8,5,5,5,1,10,17,10,13,10,9,10,5,10,1,10,1,1
1030  DATA 4,1,10,1,14,1,1,2,3,4,5,6,7,8,9,10,5,13,8
*
```

VIENNA V2

```
2     ENVELOPE 0 1,5:2,5:3,5:0
3     ENVELOPE 1 5,3:3,7:1,3:1
5     DIM F(20.0)
10    FOR N=1.0 TO 17.0:READ F(N):NEXT
15    DATA 262,277,294,311,330,349,370,392
16    DATA 415,440,466,494,523,554,587,622,659
17    FOR JCC%=1 TO 18
20    READ O,E,V,M,N,L
40    SOUND O E V M FREQ(F(N)):WAIT TIME L
45    NEXT
50    RESTORE:GOTO 10
100   DATA 0,0, 5,0, 7,0,1,0, 5,0, 4,50
110   DATA 0,0, 7,2, 8,0,1,0, 7,2, 5,20
120   DATA 0,0,10,2,17,0,1,0,10,2,13,80
130   DATA 0,0, 5,0,12,0,1,0, 5,0, 9,20
140   DATA 0,0, 7,0,13,0,1,0, 7,0,10,10
150   DATA 0,0,10,0,13,0,1,0 ,7,0,10,80
160   DATA 0,0,10,0,12,0,1,0,10,0, 9,20
170   DATA 0,0,12,0,13,0,1,0,12,0,10,10
190   DATA 0 0,15 0 8 0 1 1 15 2, 5 30
```

* * * * * * M U S I C   T U T O R * * * * * *

====================

THIS PROGRAM GENERATES MUSIC AND DISPLAYS THE NOTES.
IF YOU ANSWER YES BY TYPING Y TO THE FIRST QUESTION,
THE ONLY KEYS YOU CAN PRESS ARE THE A TO F <DO TO SI>
AND IF YOU ANSWER NO BY TYPING N ALL ALPHABETIC KEYS
ARE GIVING A NOTE. YOU CAN ALSO DISPLAY THE NOTES
LARGE OR SMALL SCALE BY TYPING L OR S TO THE QUESTION
BUT YOU NEED A 48K RAM FOR THE SMALL SCALE.

THE NUMERIC KEYS HAVE THE FOLLOWING FUNCTIONS:

1= NORMAL NOTES
2= TREMOLO
3= GLISSANDO
4= GLISSANDO+TREMOLO
5= SHORT NOTES
6= START RECORDING UP TO 2000 NOTES
7= ENDS RECORDING AND REPLAYS EACH TIME YOU PRESS IT
8= SCROLLS PAGE
0=CLEARS PAGE
SHIFT+ALPHA KEY=INVERT NOTES
TAB KEY RESTART THE PROGRAM

```
1     CLEAR 10000:LIMIT%=10:DIM ARRAY%(LIMIT%,200.0)
2     PAGE%=0:POINTER%=0:RECORD%=0:PLAYBACK%=0:TUTOR%=0:ACCENT%=0
3     PRINT CHR$(12):PRINT :PRINT :PRINT "TUTOR MODE  YES OR NO  < Y / N >"
4     ANS%=GETC:IF ANS%=0 GOTO 4
5     IF ANS%=ASC("Y") THEN TUTOR%=1:GOTO 7
6     IF ANS%<>ASC("N") GOTO 1
7     PRINT :PRINT "SIZE - LARGE OR SMALL. < L / S >"
8     ANS%=GETC:IF ANS%=0 GOTO 8
9     IF ANS%=ASC("L") THEN MODE 3:GOTO 15
10    IF ANS%=ASC("S") THEN MODE 5:GOTO 15
11    PRINT "ANSWER ONLY WITH ''S'' OR ''L''":GOTO 7
15    ENVELOPE 0 15,100;8,75;3,50;0:ENVELOPE 1 15,3;10,2;0:STYLE%=0
17    RESTORE:DIM NOTE(21.0,2.0),COMP%(21.0,1.0),SPOT%(21.0)
18    FOR I%=1 TO 13:FOR J%=0 TO 1:READ COMP%(I%,J%):NEXT J%
19    NOTE(I%,0.0)=FREQ(267.0*(2.0^(I%/12.0)))
21    NOTE(I%,1.0)=2.0*NOTE(I%,0.0):NOTE(I%,2.0)=NOTE(I%,0.0)/2.0:NEXT I%
22    FOR I%=14 TO 21:FOR J%=0 TO 1:READ COMP%(I%,J%):NEXT J%:FOR J%=0 TO 2
23    READ CHORD%:NOTE(I%,J%)=NOTE(CHORD%,0.0):NEXT J%:NEXT I%
24    FOR I%=1 TO 21:READ SPOT%(I%):NEXT I%
25    GOSUB 1500
28    FOR TIMER%=1 TO 100-99*ACCENT%
30    GOSUB 10000:IF KEY%=0.0 THEN NEXT TIMER%:SOUND OFF :GOTO 28
31    IF KEY%=53.0 THEN ACCENT%=0:GOTO 30
32    IF KEY%=54 THEN ACCENT%=1:GOTO 30
33    IF KEY%=48 THEN GOSUB 2000:GOTO 30
34    IF (KEY%=57) OR (WHERE=(-1)) THEN OFFSET=OFFSET-75.0:GOSUB 2010:GOTO
35    IF KEY%=9.0 THEN SOUND OFF :MODE 0:GOTO 3
36    IF (KEY%>48.0) AND (KEY%<53.0) THEN STYLE%=KEY%-49:GOTO 30
37    OCTAVE%=1:IF (KEY%>96) OR (KEY%=60) THEN OCTAVE%=2:GOSUB 3000
```



```
38    FOR J%=1+13*TUTOR%*(1-ACCENT%) TO 21
39    IF KEY%<>COMP%(J%,TUTOR%) THEN NEXT J%:GOSUB 3500:GOTO 28
40    FOR I%=0 TO 2
41    SOUND I% ACCENT% 15-10*SGN(I%) STYLE% NOTE(J%,I%)/OCTAVE%:NEXT I%
42    IF (SPOT%(J%)=100.0) OR (WHERE=(-1.0)) OR (OFFSET<0.0) GOTO 100
48    GOSUB 4000
50    FILL AA,BB CC,DD EE
55    DRAW FF,GG HH,II JJ
60    WHERE=WHERE+10.0:IF WHERE>XMAX-10.0 THEN WHERE=-1.0
100   GOTO 28
1000  DATA 90,67,83,67,88,68,68,67,67,69,86,70,71,67,66,71,72,67,78,65
1010  DATA 74,67,77,66,44,99,87,67,1,5,8,69,68,3,8,1,82,69,5,1,8,84,79
1015  DATA 6,10,13,89,,71,8,1,5,85,65,10,1,6,73,66,12,3,8,79,99,13,5,8
1020  DATA -10,100,-5,100,0,5,100,10,100,15,100,20,25,-10,-5,0,5,10,15,20,2
1500  OFFSET=YMAX-62.0:GOTO 2020
2000  FILL 0,0 XMAX,YMAX 0:GOTO 1500
2010  IF OFFSET<0 GOTO 1500
2020  WHERE=5.0
2030  FILL 0,OFFSET-12 XMAX,OFFSET+62 0
2040  FOR Z%=OFFSET TO OFFSET+40 STEP 10
2050  DRAW 0,Z% XMAX,Z% 12:NEXT Z%:RETURN
3000  KEY%=KEY%-32:IF KEY%=28 THEN KEY%=44
3010  RETURN
3500  TIMER%=TIMER%+1:NEXT TIMER%:SOUND OFF
3510  RETURN
4000  AA=WHERE-2.0:BB=OFFSET+(OCTAVE%-1.0)*35.0+SPOT%(J%)-2.0
4010  CC=WHERE+2.0:DD=OFFSET+(OCTAVE%-1.0)*35.0+SPOT%(J%)+2.0
4020  EE=SPOT%(J%)/5.0+8.0
4030  FF=WHERE+6.0-4.0*OCTAVE%:GG=OFFSET+SPOT%(J%)+(OCTAVE%-1.0)*35.0
4040  HH=WHERE+6.0-4.0*OCTAVE%:II=OFFSET+SPOT%(J%)+20.0:JJ=SPOT%(J%)/5.0+8.
4050  RETURN
5000  IF KEY%=56 THEN RECORD%=0:ARRAY%(PAGE%,POINTER%)=128
5010  RETURN
6000  IF POINTER%=200 THEN POINTER%=0:PAGE%=PAGE%+1:GOSUB 7000
6010  RETURN
7000  IF PAGE%>LIMIT% THEN PAGE%=LIMIT%:RECORD%=0:PLAYBACK%=0
7010  RETURN
10000 KEY%=GETC:IF KEY%=55 THEN GOTO 30000
10002 IF (KEY%=56) AND (RECORD%=0) THEN PLAYBACK%=1:POINTER%=0:PAGE%=0
10005 IF RECORD%=1 THEN ARRAY%(PAGE%,POINTER%)=KEY%:GOSUB 5000
10010 IF PLAYBACK%=1 THEN KEY%=ARRAY%(PAGE%,POINTER%)
10015 IF (RECORD%=1.0) OR (PLAYBACK%=1.0) THEN POINTER%=POINTER%+1:GOSUB 60
10020 IF KEY%=128 THEN PLAYBACK%=0
10030 RETURN
30000 RECORD%=1:PLAYBACK%=0:POINTER%=0:PAGE%=0
30010 KEY%=GETC:IF KEY%=0 GOTO 30010
30020 GOTO 10002
*
```

```
5     CLEAR 5000
10    MODE 6
16    DIM A(250.0),B(250.0)
20    COLORG 8 0 15 3
30    FOR X=0.0 TO 2.0*PI STEP 3E-2
40    A(N)=XMAX/2.0+100.0*COS(X):B(N)=YMAX/2.0+100.0*SIN(X*2.0)
45    N=N+1.0
50    NEXT
90    COLORG 8 0 15 3
100   FOR X=0.0 TO 209.0
110   DRAW 150,125 A(X),B(X) 0
115   DRAW 0,0 A(X),B(X) 3
116   DRAW A(X),B(X) XMAX,0 15
120   NEXT
300   FOR X=0.0 TO 50.0
320   COLORG 0 A 0 0
330   WAIT TIME 15
335   COLORG 0 0 A 0
337   WAIT TIME 15
338   COLORG 0 0 0 A
339   WAIT TIME 15
340   A=A+1.0:IF A=16.0 THEN A=1.0
345   NEXT X
400   FOR X=0.0 TO 50.0
410   COLORG RND(15.0) RND(15.0) RND(15.0) RND(15.0)
420   WAIT TIME 20
430   NEXT X
450   GOTO 90
```



```
1     MODE 0:PRINT CHR$(12):PRINT :PRINT
2     PRINT "...............TOWER OF HANOI..................."
3     PRINT :PRINT
4     PRINT "AN EXAMPLE OF ANIMATED GRAPHIC CAPABILITIES OF THE"
5     PRINT :PRINT "              D A I PERSONAL COMPUTER"
6     PRINT :PRINT :PRINT :PRINT "DO YOU WANT INSTRUCTIONS"
7     PRINT :PRINT "ANSWER YES OR NO  ":INPUT A$
8     IF A$="YES" GOTO 100:IF A$="NO" GOTO 200
9     PRINT CHR$(12):PRINT :PRINT "ANSWER ONLY YES OR NO":GOTO 2
100   PRINT CHR$(12):PRINT :PRINT
110   PRINT "          TOWER OF HANOI":PRINT :PRINT :PRINT
120   PRINT "YOU HAVE TO MOVE ALL HORIZONTAL BARS FROM COLUMN 1 TO"
130   PRINT "COLUMN 3 WITHOUT PLACING A LARGER BAR ABOVE A SMALLER"
140   PRINT "BAR.    FOR MOVING THE BAR YOU PRESS ON 1 , 2  OR  3"
150   PRINT "GIVING  THE NUMBER  OF THE COLUMN  FROM WHERE THE BAR"
160   PRINT "HAS  TO LEAVE  FOLLOWED  BY THE NUMBER  OF THE COLUMN"
170   PRINT "WHERE THE BAR HAS TO GO":PRINT :PRINT :PRINT
180   PRINT "PRESS ANY KEY TO START THE GAME"
190   T=GETC:IF T=0.0 GOTO 180
200   CLEAR 2000
210   DIM Z(100.0)
220   PRINT CHR$(12)
230   COLORT 7 0 0 0
240   COLORG 7 4 5 1
250   MODE 2A
300   JC1%=0:Y9=48.0:N=9.0:C1=4.0:C2=5.0:C3=1.0:C0=7.0
330   DRAW 0,0 70,0 C1
360   FOR I=1.0 TO 3.0
380   DRAW I*24-12,0 I*24-12,Y9 C2
400   Z(1.0)=0.0:Z(I*10.0)=10.0:NEXT
500   M=1.0:C=C3
600   FOR I=1.0 TO N
700   Z(1.0)=I:Z(10.0+I)=10.0-I
800   GOSUB 900:NEXT
900   GOTO 1100
1000  PRINT "INVALID MOVE"
1100  JC1%=JC1%+1:PRINT "YOUR MOVE  FROM (1,2 OR 3)  ";
1110  P=GETC:WAIT TIME 5:IF P=0.0 GOTO 1110
1120  M1=P-48.0:M1%=M1:PRINT M1%;:PRINT "  TO  ";
1130  P=GETC:WAIT TIME 5:IF P=0.0 GOTO 1130
1140  M2=P-48.0:M2%=M2:PRINT M2%;:PRINT "   ";:PRINT JC1%;:PRINT " MOVES"
1200  IF M1<>INT(M1) OR M1<1.0 OR M1>3.0 GOTO 1000
1300  IF M2<>INT(M2) OR M2<1.0 OR M2>3.0 GOTO 1000
1400  IF M1=M2 OR Z(M1)=0.0 GOTO 1000
1500  P1=Z(M1)+10.0*M1
1600  P2=Z(M2)+10.0*M2
1700  IF Z(P1)>Z(P2) GOTO 1000
2000  M=M1:C=C0:GOSUB 9000
2100  Z(M2)=Z(M2)+1.0:Z(P2+1.0)=Z(P1)
2200  Z(M1)=Z(M1)-1.0
2300  M=M2:C=C3:GOSUB 9000
2400  G=G+1.0
2500  IF Z(3.0)<N GOTO 1100
3000  PRINT "THAT TOOK YOU ",JC1%,"MOVES"
3100  STOP
9000  X=M*24.0-12.0
9100  Y=5.0*Z(M)
9200  X1=Z(Z(M)+10.0*M)+2.0
9300  DRAW X-X1,Y X-1,Y C
9400  DRAW X+1,Y X+X1,Y C
9500  RETURN
```

GRAPHIC OF SINUS

=============================

```
10  COLORT 0 15 0 0:PRINT CHR$(12.0):PRINT :PRINT
20  PRINT "THIS PROGRAM DRAW A SINUS WAVE ON THE SCREEN"
30  PRINT :PRINT :PRINT "IF YOUR MACHINE IS AN  8K RAM  YOU MUST CHANGE
40  PRINT "INTO 2A IN LINE 12 AND INTO 4A FOR A  12 K MACHINE"
50  PRINT "THIS IS ACHIEVED BY TYPING EDIT 30 AND PLACING THE"
60  PRINT "CURSOR ON THE ''6'' OF ''6A''WITH THE CURSOR ARROW"
70  PRINT "KEY AND PRESS CHAR DEL KEY AND ''2'' OR ''4''  KEY.":PRINT
80  PRINT :PRINT 'PRESS ANY KEY TO CONTINUE"
90  P=GETC:IF P=0.0 GOTO 9
100 MODE 5A:PRINT CHR$(12):PRINT " FUNCTION  =  A *SINUS B *(X - C)+ D"
110 PRINT "A=? ";
120 P=GETC:IF P=0.0 GOTO 14
130 WAIT TIME 5:A1=P-48.0:A1%=A1:PRINT A1%,"B= ?";
140 P=GETC:IF P=0.0 GOTO 16
150 WAIT TIME 5:A2=P-48.0:A2%=A2:PRINT A2%,"C= ?";
160 P=GETC:IF P=0.0 GOTO 18
170 WAIT TIME 5:A3=P-48.0:A3%=A3:PRINT A3%,"D= ?";
180 P=GETC:IF P=0.0 GOTO 20
190 WAIT TIME 5:A4=P-48.0:A4%=A4:PRINT A4%,
200 WAIT TIME 20:PRINT CHR$(12)
300 COLORG 0 15 5 10
310 PRINT "GRAFIC OF THE FUNCTION :"
320 PRINT A1;"SIN";A2;"(X-";A3;")+";A4
330 D=XMAX/4.0/PI
340 FOR N=0.0 TO XMAX STEP D
350 DRAW N,0 N,YMAX 5
360 NEXT N
370 A4=VMAX/2.0-A4*D
380 FOR M=0.0 TO A4 STEP D
390 DRAW 0,A4-M XMAX,A4-M 5
400 NEXT M
410 FOR M=0.0 TO YMAX-A4 STEP D
420 DRAW 0,A4+M XMAX,A4+M 5
430 NEXT M
440 DRAW 0,A4 XMAX,A4 10
450 FOR X=0.0 TO XMAX
460 DOT X,SIN(A2*(4.0*PI*X/XMAX-A3))*D*A1+VMAX/2.0 15
470 NEXT X
480 PRINT "PRESS ANY KEY TO CONTINUE"
490 W=GETC:WAIT TIME 10:IF W=0.0 GOTO 220:GOTO 12
500 PRINT :PRINT :PRINT :PRINT :PRINT "G R A P H I C   O F   S I N U S":PRIN
510 PRINT "=============================":PRINT :PRINT :PRINT
600 LIST
```


ARITHMETIQUE

======================

```
5   COLORT 12 0 0 0
10  A%=0:B%=0:C%=0:ANS%=0:R%=0:W%=0:POPER%=0:MODE 0
11  GOSUB 3000:GOSUB 3100:GOSUB 3300
20  CURSOR 12,21:PRINT "A R I T H M A T I C   T E A C H E R ":
22  CURSOR 15,19:PRINT "for add press..............1";
24  CURSOR 15,18:PRINT "for subtract press.........2";
26  CURSOR 15,17:PRINT "for take-away-add press.....3";
28  CURSOR 15,16:PRINT "for multiply press.........4";
30  CURSOR 15,15:PRINT "for divide press...........5";
32  CURSOR 15,14:PRINT "for multiply-divide press...6";
34  CURSOR 20,12:PRINT "SELECT YOUR CHOICE";
36  CURSOR 28,10:PRINT "?";:CURSOR 28,10
50  CR%=GETC
51  CR%=GETC:IF CR%=0 THEN 51
52  IF CR%=49 THEN 100:IF CR%=50 THEN 200:IF CR%=51 THEN 400
54  IF CR%=52 THEN 600:IF CR%=53 THEN 700:IF CR%=54 THEN 800
56  GOTO 50
100 A%=0:B%=0:MODE 0:GOSUB 3300:REM CLEAR TOP OF SCREEN
101 CURSOR 28,21:PRINT "ADD"
102 POPER%=0:E%=0:MODE 0
103 GOSUB 3304
104 XP%=19:YP%=19:CURSOR XP%,YP%:X%=A%:GOSUB 1000
105 XP%=27:CURSOR XP%,YP%:X%=B%:GOSUB 1000
106 XP%=35:CURSOR XP%,YP%:X%=ANS%:GOSUB 1000
107 GOSUB 2500:REM CALCULATE RANDOM NUMBERS
108 C%=A%+B%:XP%=20:YP%=13:CURSOR XP%,YP%+1
110 PRINT A%;"   +   ";B%;"   =  ?";
112 XP%=XP%-1:CURSOR XP%,YP%:X%=A%:GOSUB 1000
114 XP%=XP%+8:CURSOR XP%,YP%:X%=B%:GOSUB 1000
118 CP%=36:GOSUB 2040:GOSUB 2050:REM PRINT R% & W%
120 GOSUB 3000:REM DRAW BASIC FACE
122 IF E%=1 THEN E%=0:GOTO 128
124 GOSUB 3100:REM DRAW REWARD FACE
126 GOTO 130
128 GOSUB 3200:REM DRAW PUNISH FACE
130 CURSOR CP%,14:ANS%=0:DIG%=0
132 GOSUB 1500
134 IF POPER%=1 THEN 10:IF POPER%=2 THEN 102
136 ANS%=CR%-48+ANS%
138 IF ANS%>C% THEN W%=W%+1:GOSUB 2050:GOSUB 3200:E%=1:GOTO 3500
140 IF ANS%<C% AND DIG%>=2.0 THEN W%=W%+1:GOSUB 2050:GOSUB 3200:E%=1:GOTO 350
142 IF ANS%<C% AND DIG%=0.0 THEN PRINT ANS%;:ANS%=ANS%*10:DIG%=DIG%+1:GOTO 13
143 IF ANS%=C% THEN R%=R%+1:GOSUB 2040:GOTO 146
144 DIG%=DIG%+1:PRINT ANS%;:GOTO 132
146 DIG%=0:CURSOR XP%+9,14:PRINT ANS%;
148 REM X%=ANS%:XP%=XP%+8:CURSOR XP%,YP%:GOSUB 1000
150 WAIT TIME 50:CURSOR 20,14
152 IF E%=1 GOTO 108
154 GOTO 102
200 PRINT "SUBTRACT"
202 GOTO 202
400 A%=0:B%=0:C%=0:MODE 0:GOSUB 3300:REM CLEAR TOP OF SCREEN
401 CURSOR 21,17:PRINT "TAKE-AWAY-ADD";
402 E%=0,0:MODE 0
407 XP%=16:YP%=19:X%=A%:CURSOR XP%,YP%:GOSUB 1000
408 XP%=26:X%=C%:CURSOR XP%,YP%:GOSUB 1000
409 XP%=33:X%=B%:CURSOR XP%,YP%:GOSUB 1000
410 GOSUB 2500:REM CALCULATE RANDOM NUMBERS
```

```
15    C%=A%-B%:XP%=17:YP%=13:CURSOR XP%,YP%+1
20    PRINT A%;"    ?    ?    = ";B%;
25    XP%=XP%-1:CURSOR XP%,YP%:X%=A%:GOSUB 1000
30    XP%=XP%+17:CURSOR XP%,YP%:X%=B%:GOSUB 1000
35    CP%=23:GOSUB 2040:REM PRINT R%
40    GOSUB 2050:REM AND W%
45    GOSUB 3000:REM DRAW BASIC FACE
50    IF E%=1 THEN GOTO 465
55    GOSUB 3100:REM DRAW REWARD FACE
60    GOTO 470
65    E%=0:GOSUB 3200:REM DRAW PUNISH FACE
70    CP%=CP%:CURSOR CP%,14
75    GOSUB 1500
80    IF POPER%=1.0 THEN GOTO 10
85    IF C%=0.0 AND CR%=79.0 THEN PRINT "-";:R%=R%+1:GOSUB 2040:GOTO 525
90    IF C%=0 AND CR%=81 THEN PRINT "+";:R%=R%+1:GOSUB 2040:GOTO 525
95    IF C%>0 AND CR%=79 THEN PRINT "-";:R%=R%+1:GOSUB 2040:GOTO 525
00    IF C%<0.0 AND CR%=81.0 THEN PRINT "+";:R%=R%+1:GOSUB 2040:GOTO 525
05    IF POPER%=2.0 THEN GOTO 400
10    W%=W%+1:E%=1:GOSUB 3200:REM PUNISH FACE
15    CURSOR CP%,14:GOSUB 2050
20    GOTO 475
25    CP%=CP%+5:CURSOR CP%,14
30    GOSUB 1500
35    IF POPER%=1 OR POPER%=2 THEN GOTO 475
40    D%=CR%-48
41    IF D%=ABS(C%) THEN N$=CHR$(CR%):PRINT N$;:R%=R%+1:GOSUB 2040:GOTO 560
45    W%=W%+1:GOSUB 3200:REM PUNISH FACE
50    E%=1:GOSUB 2050
55    GOTO 530
60    IF E%=1 THEN MODE 0:GOTO 415
65    C%=VAL(N$):XP%=XP%-7:YP%=YP%:X%=C%:CURSOR XP%,YP%:REM GOSUB 1000
66    WAIT TIME 50
70    CURSOR XP%+7,YP%+1:GOTO 402
00    PRINT "MULTIPLY"
02    GOTO 602
00    PRINT "DIVIDE"
02    GOTO 702
00    PRINT "MULTIPLY-DIVIDE"
02    GOTO 802
000   REM SUBROUTINE TO PLACE DOMINO DOTS
001   REM EXPECTS TO HAVE DEFINED BEFORE CALL
002   REM THE X AND Y CURSOR POSITION OF THE FIRST DOT
003   REM SPECIFIED BY (XP%) AND (YP%)
004   REM THE NUMBER OF DOTS TO BE PRINTED
005   REM SPECIFIED BY (X%)
009   M%=0
010   IF X%=0 THEN RETURN
015   IF X%<0 THEN X%=X%+5:GOTO 1030
020   IF X%>=5 THEN V%=5:M%=M%+1:GOSUB 1040:CURSOR XP%,YP%-M%:X%=X%-5:GOTO 1[illegible]
030   V%=X%:GOSUB 1040:RETURN
040   FOR P%=1 TO V%:PRINT ".";:NEXT:RETURN
500   REM ROUTINE TO GET A CHARACTER AND TEST
501   REM FOR OTHER FUNCTIONS AS TAB AND REPT
503   REM SETS VARIABLE POPER% TO EQUAL 1
504   REM WHEN DESIRABLE TO RESELECT A NEW PROGRAM
510   CR%=GETC
511   CR%=GETC:IF CR%=0 THEN 1511
512   IF CR%=19 THEN POPER%=2:R%=0:W%=0:GOSUB 2040:GOSUB 2050:RETURN
515   IF CR%=16 THEN POPER%=1:RETURN
```



```
1520  RETURN
2000  REM ROUTINES THAT PRINT VALUES OF R% & W%
2001  REM IT RETURNS CURSOR TO POSITION OF CP%
2040  CURSOR 1,3:PRINT R%;:CURSOR CP%,14:RETURN
2050  CURSOR 48,3:PRINT W%;:CURSOR CP%,14:RETURN
2500  REM CALCULATES TWO RANDOM NUMBERS
2501  REM THEY ARE (A%) AND (B%)
2510  A%=10*RND(1.0):A%=INT(A%)
2520  B%=10.0*RND(1.0):B%=INT(B%)
2530  RETURN
3000  FR%=0:GOSUB 3005:FR%=47:GOSUB 3005
3005  CURSOR FR%+1,12:PRINT "#######";
3010  FOR F%=7 TO 11
3020  CURSOR FR%,F%:PRINT "# ~   ~ #";:NEXT
3030  CURSOR FR%+1,6:PRINT "#     #";
3040  CURSOR FR%+2,5:PRINT "#####";
3050  CURSOR FR%+2,10:PRINT "o   o";
3060  CURSOR FR%+2,9:PRINT "  *  ";
3061  IF FR%=47.0 THEN CURSOR 49,12:PRINT "^   ^"
3062  CURSOR 16,3:PRINT "PRESS ";CHR$(9);" KEY TO RESET SCORE"
3063  CURSOR 18,1:PRINT "PRESS ";CHR$(94);" KEY TO RESELECT"
3100  FR%=0:GOSUB 3250:FR%=47:GOSUB 3253:RETURN
3200  FR%=0:GOSUB 3253:FR%=47:GOSUB 3250:RETURN
3250  CURSOR FR%+2,8:PRINT "'   '";
3251  CURSOR FR%+2,7:PRINT " ''' ";
3252  RETURN
3253  CURSOR FR%+2,8:PRINT " ''' ";
3254  CURSOR FR%+2,7:PRINT "'   '";
3255  RETURN
3300  CURSOR 0,20:PRINT "                                        ";
3301  PRINT "                                        ";
3302  CURSOR 0,21:PRINT "                                        ";
3303  PRINT "                                        ";
3304  CURSOR 0,22:PRINT "                                        ";
3305  PRINT "                                        ";
3306  CURSOR 0,23:PRINT "                                        ";
3307  PRINT "                                        ";
3308  RETURN
3500  CURSOR 20,14:MODE 0:GOTO 108
*
```

A G E N D A

===========

```
2     CLEAR 15000
5     DIM NAME$(50.0),SURNAME$(50.0),ADRESS$(50.0)
10    PRINT CHR$(12):FOR X1=0.0 TO 59.0
20    PRINT CHR$(1);
30    NEXT X1
40    CURSOR 0,0
50    FOR X2=0.0 TO 59.0
60    PRINT CHR$(1);
70    NEXT X2
99    CURSOR 0,20
100   PRINT "*                 This is a  demonstration program                  *
110   PRINT "*                 for people who do not know about                  *
120   PRINT "*                           COMPUTER.                               *
130   PRINT "****************************************************************"
140   GOSUB 10000
160   PRINT CHR$(12)
170   FOR X=0.0 TO 59.0
180   PRINT CHR$(2);
190   NEXT X
195   CURSOR 0,18
200   PRINT "################################################################
210   PRINT "#                                                              #
220   PRINT "#  We shall make a list of i.e. 50 persons with                #
240   PRINT "#                                                              #"
250   PRINT "#          1) NAME                                             #"
260   PRINT "#          2) SURNAME                                          #
270   PRINT "#          3) NUMBER                                           #
280   PRINT "#          4) ADRESS                                           #"
290   PRINT "#                                                              #"
300   PRINT "################################################################"
400   GOSUB 10000
405   PRINT CHR$(12)
410   PRINT "################################################################"
420   PRINT "# NOTE :- If you type an error press on    !CHAR DEL!          #"
430   PRINT "#         - NEVER press on the reset button                    #
440   PRINT "#         - Every command to the computer must be              #
450   PRINT "#           followed by pressing RETURN.                       #
455   PRINT "#         - When you have typed all the names you wanted       #"
457   PRINT "#           to enter just type HALT and the same if you        #"
459   PRINT "#           want to pass to an other part of the program       #
460   PRINT "################################################################
470   GOSUB 10000
500   PRINT CHR$(12)
510   PRINT "================================================================"
520   PRINT "+                          M E N U                             +
530   PRINT "+                          -------                             +
540   PRINT "+        1) New data base              ->> NEW                 +"
550   PRINT "+        2) Look the data              ->> LOOK                +"
560   PRINT "+        3) Search ONE of the data     ->> SEARCH              +"
570   PRINT "+        4)                            ->> HALT                +
580   PRINT "+                                                              +
590   PRINT "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
600   PRINT CHR$(13)
```



```
610   DIM OPTIE$(1.0):INPUT "Type now one of those options !":OPTIE$
630   IF OPTIE$="NEW" GOTO 1000
640   IF OPTIE$="LOOK" GOTO 2000
650   IF OPTIE$="SEARCH" GOTO 3000
660   IF OPTIE$="VUL" GOTO 4000
670   IF OPTIE$="HALT" GOTO 5000
680   PRINT
690   PRINT "Please answer only with NEW, LOOK, SEARCH or HALT."
700   GOTO 600
1000  REM ********************* NEW *********************
1010  I%=1
1020  GOSUB 20000
1030  CURSOR 54,20
1040  PRINT I%
1050  CURSOR 8,21
1060  INPUT NAME$(I%)
1070  IF NAME$(I%)="HALT" GOTO 500
1080  CURSOR 12,20
1090  INPUT SURNAME$(I%)
1100  CURSOR 14,19
1110  INPUT ADRESS$(I%)
1120  I%=I%+1
1130  IF I%<=20 GOTO 1020
1140  PRINT "Sorry , but you have filled the data base!!!"
1150  GOSUB 10000
1160  GOTO 500
2000  REM ********************* LOOK*********************
2010  I%=1
2020  IF NAME$(I%)="HALT" GOTO 500
2025  GOSUB 20000
2030  CURSOR 54,20
2040  PRINT I%
2050  CURSOR 8,21
2060  PRINT NAME$(I%)
2070  CURSOR 12,20
2080  PRINT SURNAME$(I%)
2090  CURSOR 14,19
2100  PRINT ADRESS$(I%)
2110  GOSUB 10000
2120  I%=I%+1
2130  IF I%<=20.0 GOTO 2020
2140  PRINT CHR$(12):PRINT "You have now looked to the 50 persons  !"
2150  GOSUB 10000
2160  GOTO 500
3000  REM ********************* SEARCH *********************
3005  PRINT CHR$(12)
3010  PRINT " YOU WANT TO SEARCH A PERSON."
3020  PRINT " Which characteristic do you know???"
3030  PRINT "     1)Name        ->>NAME"
3040  PRINT "     2)Surname     ->>SURN"
3050  PRINT "     3)Adress      ->>ADRE"
3060  PRINT "     4)Number      ->>NUMB"
3070  PRINT "     5)None ....   ->>NONE"
3080  PRINT CHR$(13)
3090  DIM KOMMANDO$(1.0):INPUT KOMMANDO$
3100  IF KOMMANDO$="NAME" GOTO 3200
3110  IF KOMMANDO$="SURN" GOTO 3300
3130  IF KOMMANDO$="NUMB" GOTO 3500
3140  IF KOMMANDO$="ADRE" GOTO 3400
3150  IF KOMMANDO$="NONE" GOTO 2010
3160  PRINT :PRINT "Answer  only with NAME,SURN,NUMB,ADRE or NONE!"
```

```
3100  GOTO 3000
3200  REM ---------------------- SEARCH NAME -------------------------
3201  PRINT CHR$(12)
3202  DIM D$(1.0):INPUT "Do you know the name YES or NO ":D$
3203  IF D$="NO" GOTO 3210
3204  IF D$="YES" GOTO 7000
3205  PRINT :PRINT " Answer only with YES or NO .":PRINT :GOTO 3202
3210  PRINT :PRINT " Here follow the list of the names : "
3220  I%=1
3225  IF NAME$(I%)<>"HALT" THEN 3230
3226  GOTO 3260
3230  PRINT I%;"  ";NAME$(I%)
3240  I%=I%+1
3250  IF I%<=20 GOTO 3225
3260  INPUT "Wich number do you want to see";I%
3270  GOTO 3540
3300  REM --------------------- SEARCH SURNAME--------------------
3301  PRINT CHR$(12)
3302  DIM F$(1.0):INPUT " do you know the surname  type YES or NO";F$
3303  IF F$="NO" GOTO 3320
3304  IF F$="YES" GOTO 7100
3305  PRINT :PRINT " Answer please only wit YES or NO !!!":PRINT :GOTO 3302
3320  PRINT " Here follows the list of the surnames : "
3330  I%=1
3340  IF NAME$(I%)<>"HALT" THEN 3360
3345  GOTO 3385
3360  PRINT I%;"  ";SURNAME$(I%)
3370  I%=I%+1
3380  IF I%<=20 GOTO 3340
3385  INPUT "Wich number do you want to see  ";I%
3390  GOTO 3540
3400  REM ------------------ SEARCH ADRESS--------------------
3401  PRINT CHR$(12)
3402  DIM G$(1.0):INPUT " Do you know the adress , type YES or NO";G$
3403  IF G$="NO" GOTO 3420
3404  IF G$="YES" GOTO 7200
3405  PRINT :PRINT " Answer only with YES or NO  ":PRINT :GOTO 3402
3420  PRINT " Hereunder the list of all the adresses : "
3430  I%=1
3440  IF NAME$(I%)<>"HALT" THEN 3460
3445  GOTO 3490
3460  PRINT I%;"  ";ADRESS$(I%)
3470  I%=I%+1
3480  IF I%<=20 GOTO 3440
3490  INPUT " Wich number do you want to see ";I%
3495  GOTO 3540
3500  REM ----------------SEAR NUMBER---------------------------
3510  PRINT CHR$(12)
3520  INPUT " Wich number do you want to see";I%
3540  GOSUB 20000
3545  GOSUB 30000
3570  GOSUB 10000
3580  GOTO 500
4000  REM *********************** FILL *************************
5000  REM *********************** HALT **************************
7000  REM ---------------------- NAME KNOWN-------------------
7010  I%=1:PRINT
7014  DIM GEKEND$(1.0):INPUT "Wich name do you want to see ";GEKEND$
7020  IF NAME$(I%)=GEKEND$ GOTO 7050
7030  I%=I%+1
7040  IF I%<=20 GOTO 7020
```



```
7045  GOTO 500
7050  GOSUB 20000
7060  GOSUB 30000
7070  GOSUB 10000
7080  GOTO 7030
7100  REM -------------------- SURNAME KNOWN-------------
7110  I%=1:PRINT
7114  DIM GEKEND$(1.0):INPUT " Wich surname do you want to see ";GEKEND$
7120  IF SURNAME$(I%)=GEKEND$ GOTO 7150
7130  I%=I%+1
7140  IF I%<=20 GOTO 7120
7145  GOTO 500
7150  GOSUB 20000
7160  GOSUB 30000
7170  GOSUB 10000
7180  GOTO 7130
7200  REM ----------------    ADRESS KNOWN-------------
7210  I%=1:PRINT
7214  DIM GEKEND$(1.0):INPUT " Wich adress do you want to see ";GEKEND$
7220  IF ADRESS$(I%)=GEKEND$ GOTO 7250
7230  I%=I%+1
7240  IF I%<=20 GOTO 7220
7245  GOTO 500
7250  GOSUB 20000
7260  GOSUB 30000
7270  GOSUB 10000
7280  GOTO 7230
9999  REM *********************** RETURNSUBR ***************
10000 CURSOR 5,3
10010 PRINT "                         ----------"
10020 CURSOR 5,2
10030 PRINT " ***    NOW PRESS ON  ! RETURN !     ***"
10040 CURSOR 5,1
10050 PRINT "                         ----------"
10060 DIM TERUG$(1.0):INPUT TERUG$
10070 RETURN
19999 REM *********************** LABELSUBR ***************
20000 PRINT CHR$(12)
20010 PRINT "*************************************************************"
20020 PRINT "* NAME :                                           *********"
20030 PRINT "* SURNAME :                                        *Nr.*   *"
20040 PRINT "* ADRESS :                                         *********"
20050 PRINT "*************************************************************"
20060 RETURN
30000 REM ***************** PRINT SUBR **********************
30045 CURSOR 54,20:PRINT I%
30050 CURSOR 7,21:PRINT NAME$(I%)
30055 CURSOR 12,20:PRINT SURNAME$(I%)
30060 CURSOR 14,19:PRINT ADRESS$(I%)
30070 RETURN
*
```