

your computer

February, 1982

\$2.00*

NZ \$3

FOR BUSINESS AND PLEASURE

Australia's \$399 Computer!



Unveiling Applied Technology's Bee — With a 32-Page Owners Manual Inside

**BEGINNERS TUTORIALS - ASSEMBLER AND BASIC • FREE MAIL LIST
MANAGER • WE TRY OSBORNE'S PORTABLE COMPUTER • TASMANIA'S
TEACHING TRIUMPHS • MICROCOMPUTING'S MODEL T • BUSINESS
APPLICATIONS • HOW TO MAKE MONEY, WRITING ABOUT YOUR HOBBY**

**Build the
most powerful
state of the art
kit computer
complete
for \$399⁰⁰**



With the release of the MicroBee, you now have the chance to build a truly state of the art microcomputer. A computer you won't outgrow.

MicroBee is unique among kit computers. It offers facilities which make it comparable to machines costing 2 to 4 times its price.

This computer is the outcome of a brilliant design effort using revolutionary new technology.

The result is a machine which has the very finest instructional capabilities.

MicroBee, the Complete Computer

MicroBee is physically complete. You get a full case and chassis; you get the power supply; you get full manuals for assembly, BASIC programming and software development. IC sockets are supplied. And the advanced 16K BASIC is supplied in ROM – not on cassette.

In performance terms, MicroBee comes standard with features which are extra on TRS-80 and Apple. Like upper/lower case and RS232 interface. And features available on either machine like continuous memory and built-in sound.

MicroBee achieves this breakthrough at such a low price by using the latest technological developments and by taking advantage of the huge drops in IC prices in recent years.

MicroBee is a tremendous machine to use whether you're a complete novice or an advanced enthusiast.

The 16K BASIC in ROM makes MicroBee the finest instructional computer on the market. Advanced error reporting with 33 comprehensive error messages gives you instant feedback on programming errors. The BASIC is extremely 'friendly' – making it ideal for the beginner gaining computer literacy. Powerful high and low resolution graphics can be combined with alphanumerics to make MicroBee the perfect tool for learning to write educational software.

And with MicroBee you have the support of a great software base. You can run the whole range of MicroWorld BASIC software. This includes a wide range of game, educational and utility software. And the range is increasing all the time thanks to the enthusiasm of the MicroWorld Users' Group.

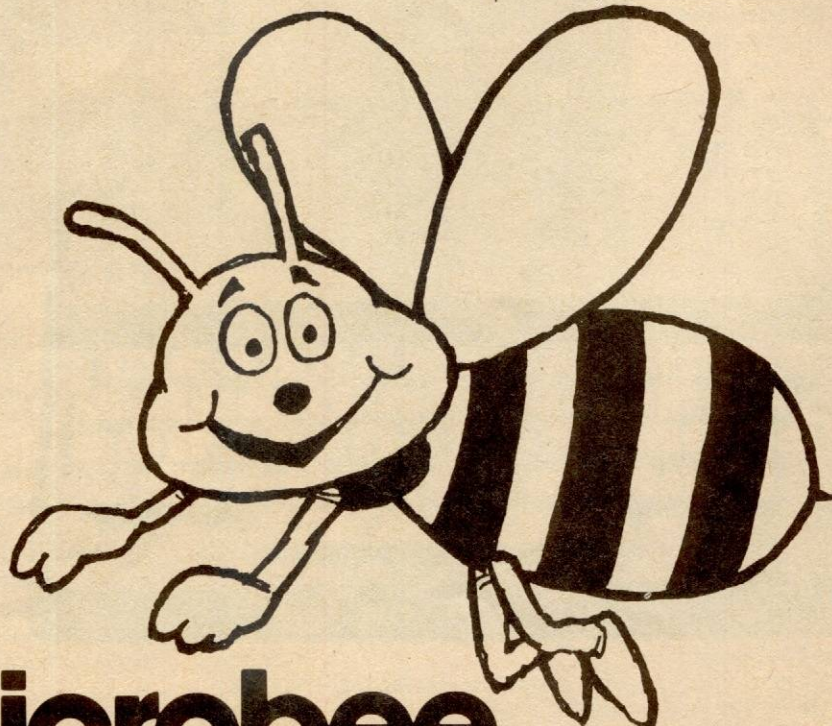
Continuous memory is here

MicroBee uses newly developed CMOS static RAM chips with battery backup – so your computer can be switched off and moved to a new location – without losing data or programs! You simply switch it back on and keep going. The possibilities are endless.

Fully expandable

MicroBee is constructed with an interchangeable 'Core' board. By replacing it with optional alternative boards your machine will be changeable from a BASIC/ROM/RAM computer to a 48K/CP/M/Disk system running the most advanced software.

MicroBee will also expand to a full S100 system and has built-in facilities for connection to printers, modems and other peripherals.



microbee



This exclusive book, produced as a supplement to the February issue of 'Your Computer', introduces Australia's newest personal computer - MicroBee!

Several very informed sources have declared the MicroBee to be a first class personal computer destined to make its mark on the world market. Here is your chance to own one for yourself and join the fascinating world of microcomputing.

ABOUT THIS BOOKLET

This exclusive book, produced as a supplement to the February edition of 'YOUR COMPUTER' introduces Australia's newest personal computer, the MicroBee.

Here is your chance to get in on the ground floor and save a bundle by building your own computer. The version of the MicroBee described here is in kit form ahead of the fully assembled and tested version due for release to the Australian and world market shortly. Although designed specifically for educational applications, the MicroBee is also an ideal personal computer as this publication indicates.

What makes a successful computer? Of all the various personal computers at present on the market only the TRS80, APPLE II and the ZX81 appear to have each sold over 250,000 to date. Surely this in itself is some real success indicator as it is really the customer not the glossy brochures that determines if a personal computer is to be a success. The MicroBee has been designed by a large team of Australia's leading microcomputer professionals and as this publication should show you, it incorporates all of the best features of the above computers with some world class innovations of its own.

The MicroBee is the first personal computer in the world to use the latest CMOS static RAM in a non-volatile configuration. It is also the first computer in the world to be prereleased to the readers of any popular magazine through a comprehensive publication such as this. This ideas book will appeal to your imagination and should leave you fired with enthusiasm with the possibilities of such a computer without the normal barrage of colourful brochures written in careful verbiage that still says very little about the things you would really like to know.

ORGANISATION OF THIS MANUAL:

This applications manual and ideas book has been arranged to enable you to see just how the MicroBee works and how it will work for you.

The first section outlines the operating manual for the computer. Here we discuss the keyboard and its operation, loading and saving programs with cassette tapes, the VDU including the graphics and programmable characters. We deal with the aspects of getting started, interconnecting the system and operating the MicroBee.

Next we discuss how the MicroBee operates. This section should appeal to those technically minded individuals.

MICROWORLD BASIC is then summarised for your evaluation together with helpful hints and suggestions. Lastly we have printed some very interesting program examples and ideas which best illustrate the advantages and power of the MicroBee. Really, it is through a careful examination of programs generating graphic illustrations, music, time clocks, games and general purpose applications that you can best judge how the MicroBee will meet your needs. You owe it to yourself to consider this section carefully!

A note on the preparation of this manual

You may be interested to know that the text for this manual was prepared entirely on Applied Technology computers, including the MicroBee, running WordStar word processing. The text was then transferred directly to the Itek Phototypesetter that produced the finished typesetting. So if you've ever asked 'What can these computers do?' - here's one answer at least!

INTRODUCTION

Here at last is a fully Australian designed personal computer packed with features not found in computers costing several times the price! One major breakthrough is the use of non-volatile CMOS RAM – a WORLD FIRST which greatly simplifies the operation of the MicroBee. Another breakthrough is the unique construction featuring a compact mainframe PCB and a range of interchangeable 'core' or memory boards which enable the MicroBee to be expanded from a BASIC/ROM/RAM configuration to a mighty 48K/CP/M/DISK system capable of running world class software. Other options include a fully programmable 8 bit I/O port, RS232 interface for printers or modems, full Z80 expansion bus to S100 or other systems and the ability to operate in a network with other computers.

The MicroBee is much more than a personal or educational computer, it is a personal communications terminal operating as your own information window to the world!

Designed in Australia by Applied Technology, the MicroBee is the result of the efforts of a large team of highly motivated and inspired individuals who are well in tune with emerging world trends in personal computers. The MicroBee is a third generation machine combining the most successful features of the TRS 80, the APPLE, the ZX81 yet with NON-VOLATILE RAM, 16K extended Level II Microworld BASIC, full function self scanned keyboard and built in sound and networking capability. It is both simple to operate and, in kit form, easy to build. What's more, the has been designed to grow and expand as your needs change so your computer investment is protected against changing technology in the fast-moving world of microprocessors.

The MicroBee in the BASIC/ROM configuration is without doubt one of the most cost/benefit effective computers on the market today and well worthy of serious comparison with TRS 80, APPLE II, ZX81 and other BASIC in ROM machines. However, by upgrading the core board to 48K and adding one or more disk drives, you gain entry to the emerging CP/M 'club' which includes computers from some of the most formidable names in the business such as IBM, Hewlett Packard, DEC, Xerox. So the MicroBee in the CPM/DISK configuration is in powerful company!

SPECIFICATIONS: MicroBee

CPU Z80 microprocessor 2Mhz clock rate

RAM 16K CMOS static with battery backup (expandable to 32K on-board)

ROM 16K containing MICROWORLD BASIC (expandable to 28K)

BASIC 16K MICROWORLD LEVEL II extended BASIC with extensive error reporting, line renumbering, supporting over 80 key words, statements and functions.

VDU Self contained, memory mapped generating 128 alphanumeric characters in a 16 line by 64 character format with full upper and lower case.

GRAPHICS Controlled under BASIC
High Resolution: 512 by 256 (PCG)
Low Resolution: 128 by 48

KEYBOARD Full sized, 60 key, QWERTY standard layout

CASSETTE INTERFACE: Built in to load and dump programs at 300 and 1200 BAUD.

SERIAL I/O RS232 option to connect to printers, modems and network with other computers.

PARALLEL I/O Optional 8 bit I/O fully programmable to connect to joysticks, digital plotters, A/D converters etc.

tone Internal loudspeaker driven under BASIC
EXPANSION 50 way Z80 bus fully buffered and decoded for future expansion, S100 bus and other peripherals.

VIDEO Standard 1V p to p composite video with neg sync

POWER 12V DC at 1 Amp

GETTING STARTED:

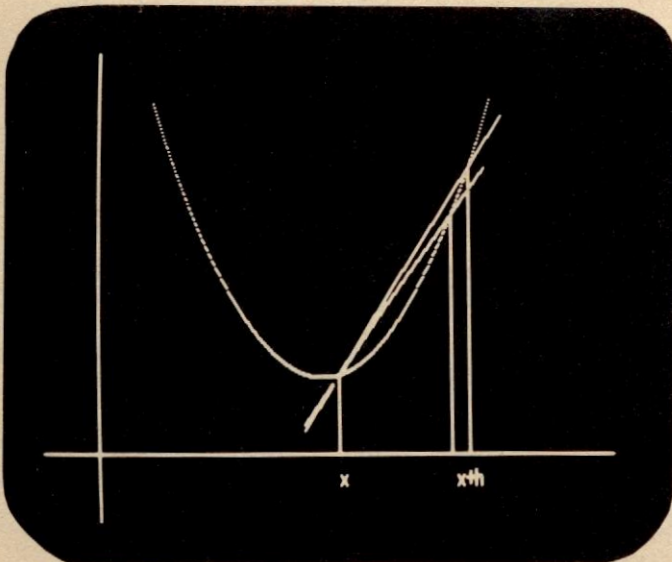
Setting up the system

Before you can start using your MicroBee you will need to interconnect the major system modules. As a check list they are itemised below.

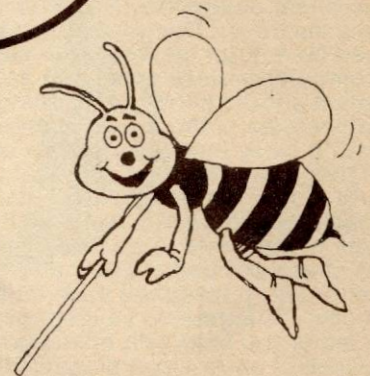
- MicroBee computer console
- MicroBee power pack and connector cable
- TV monitor or modulator with normal TV set
- MicroBee Users Manual

OPTIONAL

- Tape recorder
- Serial Printer
- RS232/network option
- I/O port option
- MICROWORLD BASIC users manual



My HiRes Graphics lets you show maths problems on the screen.



SETTING UP THE SYSTEM:

Refer to the connection diagram on the last page and familiarize yourself with the various items as illustrated.

Read through this section thoroughly before even commencing to interconnect the modules or start to use the BASIC. After a short time you will be able to set up your MicroBee without any reference to this section but for now it is best to 'hasten slowly'. (Most of us would rather risk a catastrophe than read the directions.)

Connecting the Power Pack to the MicroBee console:

1 Carefully plug the 5 pin DIN plug into the DIN socket at the rear of the MicroBee console. Note the various leads radiating from the plug and identify them against the connection diagram. Connect the leads to the MicroBee power pack if they are not already connected (watch the polarity).

2 Connect the single shielded video cable to the monitor or TV modulator.

3 If you intend to use a tape recorder connect the leads fitted with the 3.5mm plugs to any low cost portable recorder. The RED is for record and the BLACK is for playback and they must be plugged into the appropriate sockets on the tape recorder.

4 Recheck all the cables and then plug the power pack into the 240V mains and switch on both the TV and the power pack.

5 Turn on the TV monitor at the switch and watch the screen. If all is well and there is no program in your MicroBee you should see the screen clear, a sign on message announcing MICROWORLD BASIC and hear a 'beep' from the speaker. Adjust the brightness and contrast controls until a clear picture is obtained and turn the volume control to its lowest setting. Note, it is a good idea not to run the VDU screen at maximum brightness as this

RS232 Interface
for printers, modems etc.

can result in damage to the phosphor over extended periods. If a program is present you will see the screen clear and print

Ready (this prompts ready for you to input a line of BASIC.)

All you need to do is type RUN«CR» (this means type the word 'run' in upper or lower case then press the 'RETURN' key to insert the command). The program in the MicroBee will then start to execute, according to the actual BASIC lines stored in the MicroBee.

SETTING UP THE TAPE RECORDER:

It is a good idea to connect a tape recorder as this will become an ideal medium for SAVEing and LOADING back programs. Do this as follows:

1 Connect the RED and BLACK leads from the MicroBee as detailed above.

2 Connect the tape recorder to the 240V mains and switch it ON.

3 To SAVE a program (as explained shortly) you will have to RECORD a cassette. To do this insert a blank tape, wind it until the clear leader has run past the recording head and press the 'PLAY' and 'RECORD' buttons down simultaneously. When you type SAVE under BASIC and hit «CR» the program will be saved on the tape. Soon you will discover that if you type SAVE F «CR» the program will be saved at 1200 BAUD which is obviously 4 times as fast as at 300 BAUD!

4 To LOAD a program previously saved on tape you will have to rewind the cassette to the start, press the 'PLAY' button on its own and type LOAD«CR» on the MicroBee keyboard.

CONNECTION DIAGRAM

Power Pack

Note: MicroBee may be supplied with alternative types of power pack, as they are available.

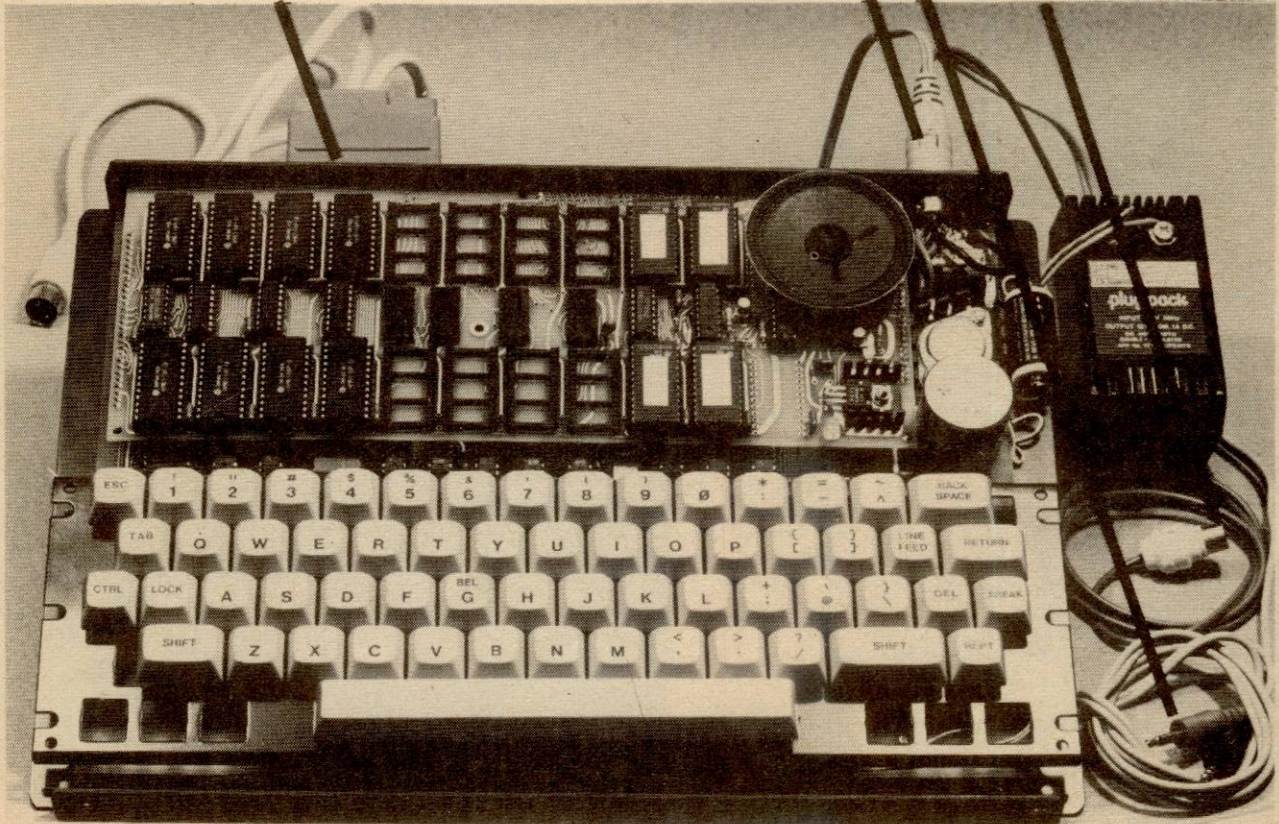
Video output

Connect to video monitor, or to TV via modulator (not included)

To cassette recorder

RED to Aux in
BLACK to earphone

5 pin DIN lead



OPERATING THE MicroBee EXPLORING THE KEYBOARD:

Now you are almost ready to start using the MicroBee. But first let us examine the vital interface between you and the computer. This is the keyboard.

The MicroBee keyboard should already look familiar to most people as it has been arranged as a normal typewriter standard layout (often called QWERTY) and in fact most keys serve the same purpose. There are however extra keys which give the computer additional information. These are called FUNCTION keys.

The ALPHANUMERIC keys are used to type 'messages' on the VDU screen just the same way as on a typewriter. You can try this by typing your name or another message on the screen. To transfer this message to the BASIC, press the RETURN key. Note that the RETURN key is ALWAYS used to terminate an instruction to BASIC and in future we will use the symbol «CR» to represent 'press the RETURN key'. Whatever you have typed on the screen will now be interpreted by the BASIC. Most likely the computer will not like what you have just typed and will respond with an appropriate message. Don't worry just now but type the following:

CLS «CR» (Remember «CR» means press the RETURN key)

Notice that the VDU screen has cleared and the cursor is resting in the top left hand corner of the VDU screen. You have instructed the BASIC to clear the screen. If you were a beginner this has been your first program step.

Now let's deal with the FUNCTION keys by describing what each does in turn.

SHIFT Upper case select, the same as with a typewriter. Holding down SHIFT selects the capital letters and the symbols above the number keys.

LOCK Acts the same as SHIFT LOCK on a typewriter. LOCK is controlled by the computer; pressing it once locks the keyboard into upper case; pressing it again reverts to lower case. This LOCK only operates on the alpha characters and is called an 'alpha lock' for this reason. To generate the symbols at the top of the numeric keys you use the SHIFT key in the normal way.

TAB Not normally used in BASIC. Some software needs this key to tabulate across the screen.

BACK SPACE Same as a typewriter. Causes the cursor on the screen to move back a character space at a time.

LINE FEED Causes the cursor on the screen to move vertically down the screen by one character line.

RETURN Just like the RETURN on an electric typewriter, this key moves the cursor from its present position to the start of the next line. It is a most important key to BASIC as it is used to enter the line currently to the left of the cursor into the BASIC program file. Some computers even label this key ENTER. Often, when it is necessary to press the RETURN key we will give you the prompt «CR».

DEL DE:lete is used to 'rub out' characters already printed on the screen. When pressed, this key erases any character to the left of the cursor. Note that when you are in the EDIT MODE, DEL will delete the character directly under the cursor. It is easier to remember that DEL deletes the character that has just been typed and in the EDIT mode the cursor itself is used to select the target character.

BREAK This special key is used to interrupt a running BASIC program.

CTRL We always seem to leave the tough ones till last! CTRL is the CONTROL key and is used in conjunction with other keys on the keyboard to generate special codes to the computer. These code are really two-key inputs and are often abbreviated to ^A, ^S (CONTROL A, CONTROL S) and correspond

with actual ASCII codes used by the computer.

Below we describe the more significant ones:

^G (CONTROL G) sounds the bell (or in our case, the loudspeaker) to get the operator's attention. Try it...hold down CTRL and press G. Now you will know what ^G means, won't you?

^A Used in the EDIT mode of the BASIC to move the cursor to the left without erasing characters as DEL would do.

^S Used in the EDIT mode to move the cursor to the right without erasing characters as would the space bar. When used in the LIST mode, this combination is used to 'freeze' the VDU display. (Press any other key to resume listing).

^C Performs the same operation as the BREAK key.

^J Performs the same operation as LINE FEED

REPT This is the warm reset or REEPROMPT.

Pressing REPT on its own will reset a BASIC program leaving the variables intact. If you hold down the ESC (ESCAPE) key for longer than 1 second and operate the REPT key (still holding down the ESC key) the program will be totally eliminated from memory and the computer will be reset to a 'hard' start.

ESC Apart from the two key usage above to achieve a 'hard' reset, the ESC key can be used to reposition the display on the VDU screen. This is very helpful if the VDU image area is not exactly central on the TV screen.

To move the image to the right press ESC, let it go and press S.

To move the image to the left press ESC, let go and then press A

To move the image up one line press ESC, let go and then press W.

Similarly to move the image down the screen press ESC, then Z.

Try this for yourself (if you haven't already) and see what happens when you type ESC, A then ESC, A and ESC,A once more. Did the image on your TV screen move across three character spaces? Magic isn't it?

REPEAT You are right, there is no key! The MicroBee is equipped with AUTO REPEAT. To engage this feature just hold down any key for longer than one second and the key will function as if you were continually hitting it. When you lift your finger the repeating stops.

My full function keyboard comes with all these great features, including auto repeat!!

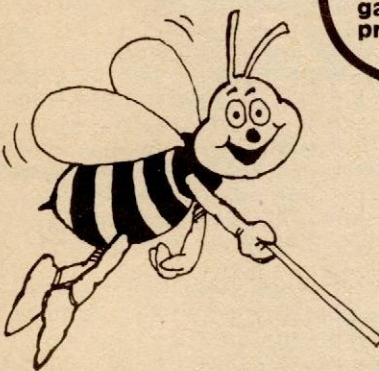


microbee

MICROWORLD LEVEL II BASIC

Your MicroBee contains the powerful MICROWORLD LEVEL II BASIC in ROM (Read only memory). When you connect power to the MicroBee the operation of the computer is immediately directed to this program. The BASIC performs the various 'housekeeping' chores for the computer such as setting up the VDU screen, initialising the memory if required and readying the system to receive commands from you via the keyboard. Don't worry too much at this point because we will discuss how the computer works and explain the MICROWORLD BASIC is considerably more detail shortly. The important thing to remember is that from the moment you switch on your MicroBee is under program control - ready to undertake your commands. (Makes you feel a little power crazy?)

Your MicroBee, in addition to being capable of running programs written in BASIC can run programs written in Z80 machine code. These are loaded with a cassette tape and carry a file type 'M' while the normal BASIC programs carry the file type 'B'. Confusing? The best way is to try things for yourself. When you type LOAD «CR» and play back a program tape you will notice the file name appear on the screen with either a B or an M appearing beside it, this is the file type. Once a machine code file has been loaded all you need to do is type EXEC «CR» and the program will automatically run. For more detail refer to the EXEC command explanation in the section on MICROWORLD BASIC.



MicroBee lets you programme your own characters. Great for games and educational programmes.

THE VDU SCREEN

Just as the keyboard is the input to the computer, the most important output device is the TV screen or, in computer terminology the VDU or Visual Display Unit. We will discuss some aspects of this now.

The MicroBee generates the characters on the TV screen in a format which gives 64 characters (including spaces) across the screen and 16 lines down. Under some circumstances this can be altered but this is definitely not for beginners.

The characters generated are upper and lower case and the font has been carefully selected for maximum readability and least tiring display. Various attributes (if you like, enhancements) have been built into your MicroBee so that you can emphasise some point. These are UNDERLINE and INVERT. The purpose of these is probably very obvious but for completeness let's examine each.

If during your BASIC program you type UNDERLINE all output to the VDU will be underlined. Similarly, if you type INVERT, all information output to the VDU after this statement will appear as black on white. Only one attribute may be selected at a time. To deselect the attributes you simply type NORMAL and the display reverts to just that.

In addition to all these display features, the user can actually create his own characters for Greek letters, maths symbols, or it can even be used for quick, easy graphics.

For example a character of a stick figure could easily move around the screen, and focus attention on the right thing at the right time

See ... 

And ... 

Portions of lines may also be underlined for another effect,

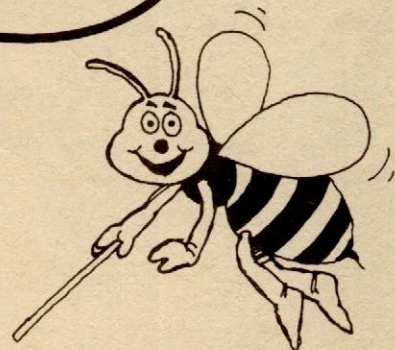
e.g. This procedure must be followed at all times.

or in case of special notation ...

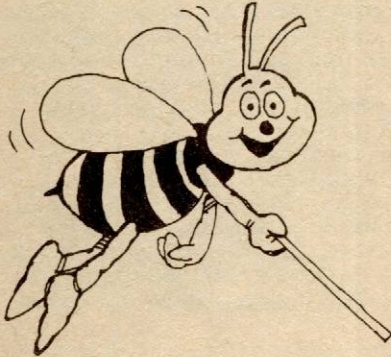
$$r = a \cos(\omega) + b \sin(\omega)$$

The verb in this sentence is swam

This great underlining feature lets you emphasise points. Give your programmes more impact.



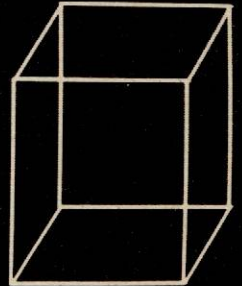
MicroBee lets you combine graphics and text. Great for educational programmes. You can draw maps and charts, and then label them.



```

06000 REM This subroutine draws a square of lengths l1,l2
06005 REM with the bottom corner at a1,b1
06010 VAR(A1,B1,L1,L2)
06020 REM Draw left side, then top, then right, then bottom
06030 GOSUB [ A1,B1,A1,B1+L2 ] 4000
06040 GOSUB [ A1,B1+L2,A1+L1,B1+L2 ] 4000
06050 GOSUB [ A1+L1,B1+L2,A1+L1,B1 ] 4000
06060 GOSUB [ A1+L1,B1,A1,B1 ] 4000
06999 RETURN
65000 END

```



Your MicroBee also contains a programmable graphics generator which can be used to generate special characters not readily available in the character generator ROM. This device is activated by typing PCG under BASIC.

The MicroBee can also be used to generate graphics by setting 'dots' on the screen in assigned locations. Two modes are available, these are high resolution and low resolution. The high resolution uses the PCG described above to generate the required graphics characters but this approach is limited to 128 different characters. The high resolution mode allows you to generate picture elements or 'pixels' with a resolution of 512 dots across the screen and 256 dots down. No wonder we run out of 'steam' because of memory limitations. To overcome this problem a low resolution graphics mode is included which divides the screen into 128 by 48 dots and this mode is not subject to memory constraints.

To cater for the graphics capability, the screen has been broken down into various sections. In the HIRES mode you can specify and point between 0 and 511 across the screen and between 0 and 255 up the screen. Best we use an example:

SET(511,255) sets a point in the top right hand corner of the screen.

SET(256,128) sets the point in the middle of the screen.

In the LORES mode you use the same designations except the range of the numbers changes because you can only use an array of 128 dots across the screen and 48 dots down.

SET(127,47) sets a dot (if you like turns on a point) at the top right hand corner or the VDU screen.

In the alphanumeric mode you can position the cursor anywhere on the screen. Since the screen is broken down into 64 'character boxes' across the screen and 16 down we have 64 multiplied by 16

different locations or 1024 if you do the arithmetic. MICROWORLD BASIC enables you to use the CURS command to position the cursor anywhere you want it. You can type

CURS 1023:PRINT '' «CR»**

and the cursor will be positioned briefly at the lower right hand corner of the screen and print the character * on the bottom of the screen. Similarly CURS 0 will do a similar thing in the top left hand corner.

If you are not sure of the numerical position of a particular point on the screen you can also address it using the character number format and the line number i.e suppose we want to print HELLO on the 4th line from the top of the screen and starting at the 10th character on that line then you would type:

CURS 10,4:PRINT 'HELLO' «CR»

and you will see the word HELLO appear on line 4 at the position of the 10th character along.

MICROWORLD BASIC also supports commands to draw lines from one point to another in the graphics mode. These apply to the LORES and HIRES modes. If you type (in the HIRES mode):

PLOT(0,0) to (511,0) to (511,255) to (0,255) to (0,0)

you will draw a border around the screen area and, at the same time get a very good idea of where each point actually lies.

MUSIC AND SO ON

Computers are ideal music generators. The MicroBee has a built in loudspeaker which can be driven by the BASIC to generate music over two octaves. The command is PLAY A , where A can be any integer variable from 1 to 24. Although the computer only generates one note at a time (i.e it is monophonic) some very impressive results can be obtained. You are referred to the program ideas later in this manual for more details.

How the MicroBee works

This description of how the MicroBee works has been written to reduce the 'computerese confusion' that the common abbreviations cause most people who, although they have some grounding in digital electronics, have not yet 'learnt the lingo'.

When acronyms occur for the first time, they are spelt out in full with bold letters forming the acronym, and from then on, the acronym will generally be used. Note that all digital signals which are active at the low voltage level are indicated by adding an asterisk to the name, for example RD* . (A signal is called 'active' at a particular time when its state at that time agrees with it's name).

Power Supply:

The external power supply connection to the MicroBee must be approximately 12v DC, bridge rectified, but not necessarily filtered, as there is a 10,000 uF capacitor internal to the unit which smoothes the pulsating DC.

This smoothed DC then goes on to two 5 volt regulators which regulate the 12v down to 5 volts, one regulator being on each of the two boards. The 12v unregulated voltage also goes to the power-fail detect circuitry on the top (core) board so that the **Complementary Metal Oxide Semiconductor Random Access Memories** can be powered down before the processor loses power.

Clock circuitry:

The clock circuitry is that surrounding the 12Mhz oscillator. This oscillator provides all the timing for both the **Visual Display Unit** section, and the **Central Processing Unit** as well, it has nothing to do with a 'time of day' clock. The 12MHz oscillator, using inverters from a 74LS04 package, IC23, is a standard crystal based design with a following buffer to 'square up' the signal further and allow it to be used with many other devices.

This 12Mhz square wave signal is divided by 6 in IC32, a 74LS92 of which the first divide by two stage is not used. This gives a 2Mhz processor clock which is pulled up by a 330 ohm resistor to give the correct voltage levels required for the CPU and the **Parallel Input/Output** interface controller. 12MHz is also sent to the VDU section (see VDU description).

Immediate CPU environs:

This circuitry is that which handles and buffers (increases current drive capability) the data, address and control signals of the CPU.

The Z80 CPU, IC25, transmits and receives all its data, whether it is instructions being executed, or **American Standard Code for Information Interchange** characters, or binary numbers, through its DATA lines, named D0 through D7. D0 is the **Least Significant Bit**, meaning that it's place value is 2 to the zeroth power, or 1.

These bidirectional data lines from the CPU are buffered by IC19, a 74LS245 octal bus transceiver. This chip has two control inputs.

Pin 1 controls the direction of data flow through the device. This pin is set high when either a READ cycle is indicated by the CPU's RD* signal, or when the IORQ* and M1* signals are concurrently active, indicating an interrupt acknowledge read from a device such as the PIO.

Pin 19 either selects (low), or deselects (high) the chip. When the chip is deselected, the CPU DATA lines are isolated from the external data bus which connects to all the devices such as the PIO the 6545 CRT controller chip. These buffers are disabled for one of two reasons.

The first case is when an external device tries to gain direct access to the MicroBee's internal BUSes by the use of the BUSREQ* signal. When the Z80 detects that this signal is low, it finishes the current bus cycle, and responds with the BUSAK* signal. This line is used to disable processor control of the

data bus, address bus, and all control signals. It is inverted once by IC23 to disable the address and control buffers (ICs 28, 22, 24). This inverted line is then reinverted to give a buffered BUSAK* signal to external devices, and also an input to IC29 on pin 2, which will disable the data bus buffer chip when BUSAK* is low.

The second reason to disable the data input buffers is to perform the power on jump (actually 'jump after reset') which forces the processor to execute 'nops' until the desired start-up program is reached. The 'nop' is an 'instruction' for the Z80 which does exactly as the name suggests, NOTHING. Therefore we make the Z80 scream through its addresses, starting at address 0 until it reaches the BASIC roms.

In a disk based system, this power on jump would not jump to the BASIC, but would start executing when it found a 'bootstrap' **Read Only Memory**, which is merely a program with sufficient intelligence to bring in a bigger program from disk, and execute it.

Thus the MicroBee avoids the need for having ROM at location zero, where the Z80 would normally start executing after a reset, so it is a 'natural' for operating systems such as CP/M which use memory starting at zero.

This power on jump function is mainly performed by one half of IC33, a 74LS74 dual D flip-flop.

This flip-flop is reset on pin 13 whenever the processor is reset (the reset signal comes from the top board.) When this flip-flop is reset, the Q output, pin 9 goes low, and therefore, IC29 disables the data buffer chip, IC19. With the data buffer chip disabled, the CPU will read nops (which has an op-code of 00 for the Z80) because of the high-valued pulldown resistor pack, RN2, and so the CPU will just continue through its address space until it reaches the required ROM to start executing from.

When the address reaches this ROM, the top board ROM decoder sends down a low logic level on the ROM SELECT* line and this goes to the SET input, pin 10 of the flip-flop so consequently the data buffer chip is enabled, and execution starts with the first byte of the ROM.

This power on jump circuitry also serves the purpose of protecting the RAM from spurious writes which the Z80 seems to like doing some time after the reset signal has been applied to it. This is done by gating the WR* line through IC30 and IC31 so that writing is disallowed from as soon as the reset is applied up until the ROM is found.

The Z80, a typical 'Von Neumann' architecture CPU indicates where its data is to go to or come from through its ADDRESS BUS. This BUS is a collection of 16 address lines which each specify one bit of a 16 bit binary number. This 16 bit binary number gives each memory location (or place to store 8 bits of data) a unique address, working the same way as house numbers in a street do, where each different number specifies one house.

This address bus is buffered by two 74LS244 octal buffer chips, IC22 and IC24. These chips differ from 74LS245 devices in that they only transmit digital signals in ONE direction. This is all that is required because the Z80 address bus is always the originator of addresses. These chips can be disabled by taking pins 1 and 19 to the high level, as is done when an external device requests the bus, and the Z80 grants it by asserting the BUSAK* output.

Input / Output Devices:

This section describes the decoding of all I/O enable signals, and the circuitry attached to the PIO.

IC27 is a 74LS85, a four bit comparator which decides when the CPU is trying to access an I/O port at any address in the range 00-0F. The A=B input of this IC, pin 3 is driven by an inverted IORQ* as it uses positive logic for its inputs. The output is also positive logic, so it is inverted to enable the active low input of the first half of the 74LS139, IC34. This IC is a dual 2 to 4 line decoder, which means that each half of the IC takes 2 address lines and an enable, producing 4 discrete active low outputs which correspond to the 2 bit binary word given to it.

The first half, then, divides the 00-0F address range into 4 parts. Two of these 'block of 4' outputs are used to drive the chips selects of the PIO and the 6545 CRT controller directly at addresses of 00-03 for the PIO and 0C-0F for the 6545. One output, corresponding to addresses 08-0B enables the second half of the 74LS139, and allows the decoding of some single I/O ports. Port 0B is used to control access to the character generator ROM for generation of INVERSE and UNDERLINE characters.

The PIO

The PIO is a Z80 family device which allows 16 bits of fully independent input or output. As a Z80 family device, the PIO's configuration is fully programmable by the CPU at system initialization time. The MicroBee uses port B of the PIO (the second half) for cassette interface, RS232 interface/network control and speaker driving functions.

Port A is NOT used for any internal functions of the MicroBee, and is therefore fully available for use in whatever way is desired by the MicroBee user. It is normally set up in the software as an 8 bit parallel input port which uses the strobe line, and interrupts the CPU when such a strobe is received. Therefore, it is easy to plug an external keyboard, for example, into the port A socket, and this function is provided for in the input stream setup of the MicroBee BASIC. It may also be used as anything at all by modifying the I/O initialization data in the BASIC scratchpads.

Most of the Z80 CPU signals are used by the PIO to control direction of data, interrupt control, etc. The 'interrupt enable in' line of the PIO is tied high, and as such, it becomes the highest priority interrupting device in any MicroBee.

Speaker

The speaker, used to generate music and error signals, is driven by one bit of port B in the PIO, namely bit 6. For this line to act as an output, it is set up that way in the BASIC ROMS. This digital signal controls the BC548 transistor, TR3 which in turn drives the speaker in grounded emitter mode.

RS232

With respect to the RS232C standard (Australian Standard V24/V28), the MicroBee is configured as if it was 'data terminal equipment', which allows standard connection to MODEM devices.

The RS232 output line, or 'Transmit data' on pin 2 is driven by a transistor which puts 12v on the line for a low (spacing) digital signal from B5 of the PIO, and turns off, allowing 0v on to the line for a high (mark) level from B5 of the PIO. The actual RS232 TX DATA line will rest at 0v when no character is being output, corresponding to the MARK condition. A 470 ohm resistor protects the output line from damage in case of external shorts.

The RS232 data input line (pin 3) takes RS232 signals over a voltage range (-12v to +12v generally). The resistor and diodes limit the PIO input to a 0v to 5v range, the necessary inversion of the signal being accomplished as part of the software which reads characters from the RS232 port.

The Clear To Send input from the RS232 port (pin 5) allows transmission from the transmit data line, and is usually used as a 'printer ready' signal. This signal is voltage converted as for the data input line. Transmission of characters is allowed when the CTS line is at a HIGH voltage. When two pieces of equipment set up as 'data terminals', swapping over pins 5 (CTS) and 20 (Data Terminal Ready) in the connection line as well as pins 2,3 will allow their use with each other. (Pin 7 on RS232 is the signal ground pin).

The other circuitry marked CLOCK, and the capacitor into B7 of the PIO is provided for use when NETWORKING MicroBees.

The cassette interface of the MicroBee is mainly software driven, but the MicroBee turns this into a major advantage by realising far greater RELIABILITY than is often obtainable using hardware designs. This is because the MicroBee cassette interface software can actually 'track' the data coming in on a bit-by-bit basis, and therefore cope with much greater speed irregularities between tape recorders.

The cassette output consists merely of an RC network which accepts a signal from B1 of the PIO, attenuates and then decouples it before sending it to the cassette recorder 'aux' input. (Microphone inputs can be used if the 4k7 ohm resistor is increased in value.)

The cassette input circuit is slightly more complicated, consisting first of an attenuator / decoupler, then an op-amp to allow a wide range of input levels to be squared up before being passed to the PIO input on B0. The function of the two diodes is to clip any input signals greater than the diodes' forward voltage in either direction. The 47pf capacitor is required by the CMOS op-amp for precompensation.

The Visual Display Unit

The VDU is that part of the computer which generates the characters and graphics that are seen on the monitor and allows the CPU to look at and modify these.

The MicroBee's VDU is one of its greatest strengths. By basing it around a 6545 'crt controller', great flexibility has been realised. Although the MicroBee BASIC uses a 64 characters by 16 line display for reasons of compatibility and ability to use modified television sets, it is very easy to 'reprogrAM' the 6545 chip to allow configurations such as 24 lines of 80 characters as often used in up-market CP/M systems for which you will pay a small fortune. A total of 2048 video characters maximum has been provided for just such a use.

The software reprogrammability of the 6545 also allows advanced features such as keyboard control of screen positioning (which is retained during power down thanks to the CMOS memories).

The 6545 device also forms the heart of the MicroBee's keyboard circuitry, allowing 2 simple external ICs for a full QWERTY keyboard with 2 key rollover and automatic repeat !

The other major breakthrough in the MicroBee VDU is the inclusion and FULL software use of programmable character generator circuitry to generate all graphics (both LORES and HIRES) and video attributes (UNDERLINE and INVERSE).

To achieve graphics capability as well as high speed output and compatibility with earlier Applied Technology systems, the MicroBee's VDU circuitry is 'memory mapped'. This means that all of the VDU's normal and graphics characters are accessible as if they were everyday memory. The 'screen display' RAM sits from F000H (Hexadecimal) to F7FFH (CPU addresses in hexadecimal, or base 16), while the PCG RAM sits from F800H to FFFFH. There is

no separate section for attributes, because the only modifier allowed for the characters is that of selecting PCG characters instead of normal characters by setting B7 of the screen display RAM high. This is why only one VDU attribute is available at one time.

General MicroBee VDU operation

This section will try to explain how the VDU works conceptually without referring to details of the circuitry yet.

To complete one scan of a display monitor, the VDU must generate what are called SYNC and VIDEO signals. The SYNC signals are timing marks which tell the monitor when to retrace vertically so as to start at the top of the screen, and also when to retrace to the left hand side of the monitor, moving one scan line down so as to be ready for the beam to traverse the next scan line from left to right. Do not confuse 'scan lines' with 'character rows'; one scan line is the thinnest line discernible on the monitor screen, whereas the character row contains the complete height of a character, normally 16 scan lines high.

The VIDEO signal tells the monitor how bright the point over which the beam is currently passing should be. The MicroBee uses only 'on', and 'off' here, so that a dot is either bright or dark on the monitor.

Therefore, to successfully display one frame of the display screen, the VDU must produce and co-ordinate VIDEO and SYNC signals which combine to give a picture showing what is 'on the VDU'.

To describe how the VDU generates the frame, we are starting right at the beginning of one frame scan.

As the beam of the monitor flies from the left of the screen to the right hand side, the VDU must get the dots in the right order to be output as the VIDEO signal. Therefore, to start off, the VDU must output the top scan line of the first character on the screen (top left hand character). After it has put out the 8 dots for that section, it must put out the top scan line of the second character on the screen, and continue this same process for the top scan line of each of the first 64 characters on the screen.

At this stage, the VDU must realize that it has

reached the end of one scan line, and so must stop sending out VIDEO, and tell the monitor to return to the left hand side of the screen by giving a LINE SYNC PULSE.

When enough time has passed for the monitor to be ready to start the second scan line, the VDU will start sending out the SECOND scan line of each of the same first 64 characters in the same order as the first scan line.

This process is then repeated until the 16th scan line has been completed. At this stage, the VDU knows that it has fully sent out the first character row and that it must now move on to the second character row, which starts on the monitor's 17th scan line.

Thus, on the 17th monitor scan line, the VDU sends out the first scan line of the second character row. On the 18th monitor scan line it sends out the second scan line of the second character row.

Finally, after 16 full character rows have been sent out, (or 256 monitor scan lines), the VDU must stop sending VIDEO, and tell the monitor to retrace to the top of the screen and do it all again by sending out a FRAME SYNC PULSE.

The MicroBee VDU must do all of this fifty times every second to keep up the illusion that all the dots on the screen are illuminated at once!

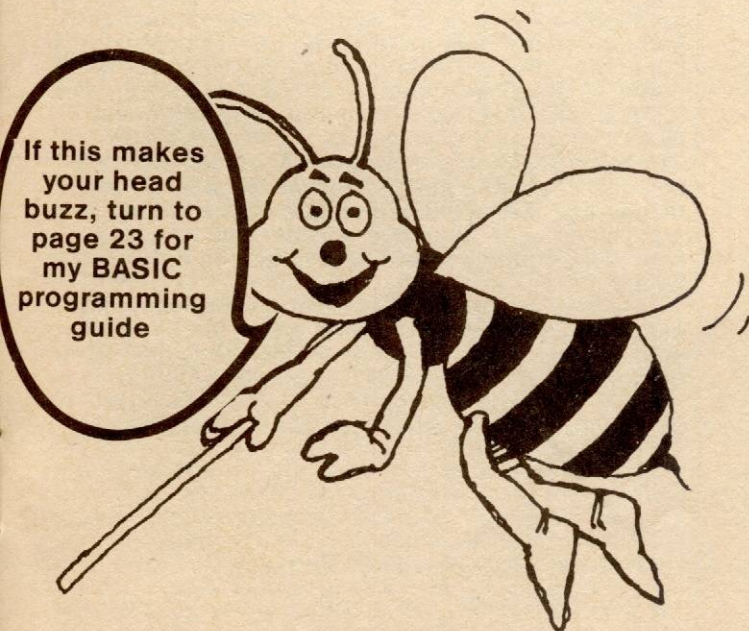
To fill-in some detail, it is necessary to realize that for each character position, 8 bits of data are stored in the screen display ram. This data is output on what is called the 'secondary data bus', because it is electrically separate from the main system DATA bus as described under the CPU environs section. 7 bits of this data give a code of which character to display. This is exactly the right number, as there are 128×2 to the 7th power characters in both the Character ROM and the PCG ram. Bit 7 FROM the display RAM selects the Character ROM if it is low, and the PCG RAM if it is high. Obviously though, the Character ROM or the PCG RAM must also be told which actual scan-line of the 16 possible should be sent out.

Once this information has been given to (say) the Character ROM, it will produce 8 bits of output which represent the desired on/off condition of the 8 dots for one character's one particular scan line. These 8 bits of data (on a bus called the tertiary data bus), appear all at the same time, but for the monitor to display them, they must be output serially.

A shift register takes care of converting the parallel dot data into a serial dot stream which, subject to inversions in case of a cursor and blanking so that rubbish is not sent to the monitor when outside the displayed range of characters, becomes the VIDEO signal which is mixed with the SYNC signal and sent to the monitor.

We can now refresh the monitor screen with the data already in the VDU rams, but how does it get there in the first place? Just as the VDU circuitry generates addresses which correspond to the various positions on the screen, the CPU may also read and write from any character position in the screen display ram, and may examine and alter any of the data contained in the PCG RAM which defines what the programmable characters look like.

The CPU gains access to the VDU RAM and ROM address lines through multiplexers which swap between scanning addresses and CPU addresses as CPU accesses are made. The secondary and tertiary data buses are available through two octal (8 bit) bus transceivers, allowing access to the data lines of the screen display RAM and the PCG ram/Character ROM. The black streaks which are obvious on an inverse screen when VDU accesses are made show where some time has been stolen by the CPU during the VDU's monitor refresh period.



The VDU's specific details

The 'brain centre' of the MicroBee VDU is the 6545 crt controller chip, IC9. It is important to understand just what the 6545 does and does not do. It does organise all addresses, scan line row selects, all sync pulses, blanking and cursor positioning. It does not ever actually see any of the display data, and it does no serialisation of this data to produce the video output.

The data bus on the 6545 is purely to talk to the CPU about initialization data, where the cursor should go, how many lines on the screen and so on.

The only timing input to the 6545 is what is called the 'character clock'. This is (in this case) a 1.25MHz signal derived externally from the 12MHz clock by the 74LS161, IC26. This 1.25MHz CCLK is one eighth of the frequency at which the dots are clocked out, and so tells the 6545 when to change addresses for the next character on the screen and gives it the correct timing for the SYNC pulses.

The 74LS161 does more than just divide by eight, though, because it also provides divide by 4 and divide by 2 outputs which are combined by IC30, inverted by IC23 to give the LOAD* signal, an output which stays low for one 12MHz cycle once for each character. This output causes IC12, the 74LS166 to accept the dot data in parallel (or 'broadside') on the next rising edge of the 12MHz clock. The time of this LOAD* signal has been calculated so as to allow for the propagation delays in the 6545, and access time of the screen display RAM and then either the PCG RAM or Character generator ROM before the actual parallel bit data becomes available.

The 74LS175 device, IC15 manages the task of synchronizing the start of a new character (the LOAD* pulse) with the 6545 blanking output (which indicates when the 6545's output addresses no longer reflect characters to be output), and the cursor output, which tells when one character wide section of the scan line should be inverted. IC17 is an exclusive or gate which is used here to allow the inversion of the serial output from the 74LS166 shift register in case of cursor (or if the INVERT line from the 82S123 were to go high).

Access blanking is the process of setting VIDEO output to black whenever the CPU is using the memory and the monitor refresh must be temporarily abandoned. If access blanking was not used, the screen would be filled with random scratches whenever a CPU access was attempted.

The F000* signal initiates access blanking by activating the master clear input on the 74LS175. This forces the Q3 output of the '175 low, and thus clears the shift register, IC12, to zero. When the next LOAD* pulse's positive edge comes along, the Q3 output of the '175 is still instantaneously low, and so the shift register stays cleared for the next character as well. The same LOAD* pulse then strobes the state of pin 13 of IC15, D3 (high) through to the Q3 output, and thus allows the character following to be serialized as per normal.

Thus an access forces the currently displayed and next character on the current scan line to be blanked. This digital blanking ensures minimal interruption to the monitor refresh.

The other type of blanking, off-screen blanking, is controlled by the 6545, and synchronized with the LOAD* pulse as described earlier. This DISPLAY ENABLE (pin 18) on the 6545 is an output which is high whenever the current scan position is one that should be looked up and serialized. It is inverted as it passes through the 74LS175 since the Q* output is used, allowing it to be connected in the video line with a 74LS02 (IC20) gate, so as to set the VIDEO to black when off the displayed section of the screen.

The video from this gate now has the desired polarity, and is mixed with the combined HORIZONTAL and VERTICAL sync line, again provided by a NOR gate of IC20. The resistors serve to mix sync and video in the correct proportions. The 22uF capacitor decouples the combined signal before it is passed to a buffer transistor which drives the video output circuitry.

Internal VDU buses

It is important to overall understanding of the VDU to distinguish the separate internal VDU buses over which data and addresses are sent and to realize where the actual dot data comes from.

The address line inputs to the screen display RAM, IC5, select which of the 1024 possible characters on the screen IC5's data outputs should present the code for (or where a code on the data inputs is to be written). During screen refresh, the address lines used are the 'MA' lines from the 6545 which select the correct characters as described earlier under display scanning. When the CPU is accessing the screen display RAM, however, the relevant address lines are the CPU signals A0-A10. Thus, what we need is a way of switching over large groups of signals between two alternatives. The quad 2 line to 1 line multiplexers, 74LS157 devices, fulfill this need and the control signal which switches them either way is the F000* signal.

The screen display RAM DATA lines are bidirectional, which means that during a read, data flows out of the device, and when writing, data flows into it. Therefore these lines form what is called the secondary data bus, which is linked to the system (or primary) data bus through a 74LS245 bus transceiver, IC11, so as to give the CPU access to the screen display RAM.

An electrically separate data bus is necessary here because it is very likely that, for example, while the CPU is receiving instructions from the BASIC ROMs, the VDU will also need to transfer information from IC5 into the second stage of multiplexers (IC6, 10, 21).

The address and data lines of the Character ROM, IC13, and the PCG RAM, IC18 are connected in parallel because they perform similar tasks and the situation is always that only one is ever used at one time. The address lines to these devices perform two functions. A4-A10 select which number character to look up the dot data for (this line usually comes from the screen display RAM outputs). A0-A3 select which of the 16 possible scan line rows to find the dot data for. Normally this row information comes from the 6545, but on a CPU access of the PCG RAM or Character ROM, the information about which row to select comes from the 4 lowest address lines. This means that the layout of a character in memory consists of each character in order with its sixteen scan lines running from top to bottom in consecutive bytes.

The data outputs from these chips represent the actual dot data which is serialized by the 74LS166 shift register and sent out as VIDEO. They also form the tertiary data bus, which is connected to the primary data bus through IC11, a 74LS245 bus transceiver, allowing CPU access to the Character ROM and PCG RAM.

VDU Memory map controls

The decoding of addresses, and handling of read/write and data direction controls is done by IC16 and one half of IC33.

IC16 is an 82S123 fusible link prom which is programmed specially for this application to derive the complex logic with a minimum of parts.

It is given an F000* signal on pin 13 whenever the address lines match the F000-FFFF range and MREQ* is low. The WR* signal on pin 10 tells it when a write operation is being requested by the processor so as to modify the VDU's contents. The MREQ* signal merely indicates when valid CPU accesses are possible. The SELECT line is a line which indicates when the currently displayed character is a PCG character (normally) or the state of A11 from the CPU (during an access, i.e. when F000* is low).

The ROMREAD* signal on pin 12 is a special signal which comes from a 1 bit output port at address 0BH provided by one half of IC33. This flip-flop is normally in the reset state for the memory map as described above. If, however, the processor temporarily writes a '1' into this port and sets the flip-flop, the 82S123 is instructed to allow access to the character generator ROM, IC13, when a read from F000H to F7FFH is done. This is how the inverse and underline attributes are gained without special data in the BASIC ROMS: when the INVERSE keyword is found, the CPU temporarily sets this flip-flop and takes each byte of the character generator ROM, inverts it, and places it in the PCG RAM. For underline, the bottom display line is set to FFH so as to give a solid line underneath each character. The outputs of the 82S123 are as follows:

Pin 1 This line controls writing into the PCG RAM, and is normally high, going low when the CPU wants to write into the PCG.

Pin 2 This line turns on the output buffers in either the Character ROM, IC13, or the PCG RAM, IC18 depending upon which one is selected and is normally low, except when a write is done into the PCG RAM.

Pin 3 SELECT* is merely the inversion of the SELECT input to the 82S123, thus forming a complementary pair which is used to select either the PCG RAM or the Character ROM.

Pin 4 INVERT is always low, but allows creation of inverse characters for when a PCG is not present.

Pin 5 This line enables writing into the screen display RAM for when the CPU requests it.

Pin 6 Similarly, reading is enabled from the screen display RAM when this output is low (therefore it is normally low).

Pin 7 The 'tertiary' data bus transceiver is enabled by this line whenever an access to the PCG RAM or Character ROM is requested by the CPU. (Active low signal)

Pin 9 The 'secondary' data bus transceiver is enabled by this signal when an access to the screen display RAM is desired by the CPU.

Some control signals perform tasks directly as well as through the 82S123.

The F000* signal is connected to the select line on all the 74LS157 quad 2 line to 1 line multiplexers.

These multiplexers are like big multi-ganged switches for digital signals. When the F000* line is high, then the VDU is not being accessed by the CPU, and should therefore be refreshing the monitor screen. The multiplexers ICs 3,2,8 then give the screen display RAM the relevant MA address lines from the 6545 which tell which character out of the 1024 possible to look at. ICs 6,10,21 will give the character generator ROM or PCG RAM the code of the character as obtained from the screen display

RAM and the scan line number from the 6545 (RA0-RA3).

If the CPU wishes to access the VDU, though, F000* goes low, and the multiplexers switch over to connect the address lines of the VDU RAMs and ROMs to the CPU's buffered address bus.

The SELECT signal which emerges from IC21 on pin 12 thus becomes different things when the CPU is accessing the VDU and when it isn't. When F000* is low, the VDU is being accessed, and SELECT follows A11 which will select whether the PCG RAM or Screen display RAM is being accessed (PCG RAM or Character ROM when ROMREAD*0). When F000* is high, SELECT becomes data bit 7 from IC5, the screen display ram, and thus B7 selects whether the character code defined by the B0-B6 outputs of IC5 are shown as a character from the ROM, or a PCG character.

The screen display RAM is always being used for some reason, and therefore its chip select line on pin 18 is tied to the low level. Similarly, the Character ROM is never written to, so the Vpp input on pin 21 is tied high.

See the back page for ordering details

The Keyboard

The MicroBee keyboard interface is one example of the way clever design can bring down costs dramatically. The entire count of chips devoted only to the keyboard is two, a demultiplexer and a multiplexer.

The keyboard interface relies upon the 'light pen' feature of the 6545 crt controller chip. This consists of a LPEN input which strobes the currently accessed character address into a 16 bit register pair internal to the 6545 chip.

Accordingly, if some of the lower address lines, which are constantly being counted through in order to display characters on the screen, are attached to some circuitry which assigns each different address to one particular key on the keyboard, the 'address' of that key can be captured inside the 6545 and then converted to ASCII by a keyboard decoding program in the BASIC ROMs (or CP/M BIOS).

The MA9, MA8, MA7 lines from the 6545 go to a 74LS156 device, which is an open collector dual 2 to 4 line decoder wired as a single 3 to 8 line

decoder by joining pins 1 and 15, which become an extra address line. This 3 to 8 line decoder is enabled by the logical or of two conditions, one being display enable, the other being the RA4/update strobe used in this context as an update strobe (see 6545 data for more information).

Therefore, as the screen is refreshed, each of the eight scanning (or x) lines are driven low in turn. This action cannot be observed on a CRO until a key is pressed due to the open collector nature of the 74LS156. If one particular key is pressed, that point on the x-y matrix is joined, and the corresponding y line (or input to the 74LS151 multiplexer) will go to the low logic level (otherwise it is pulled up by the RN1 resistor pack).

As the MA4, MA5, MA6 inputs change, the relevant input will be selected and its complement placed onto the LPEN input of the 6545, so that as the right 'x' address matches the right 'y' address, a high going edge is presented to the 6545 LPEN input, and the data is latched in, ready to be read by the Z80 and converted into the ASCII code.



The CORE board

The CORE board, or TOP board is the plug on board which contains all of the MicroBee's executable memory.

The normal BASIC board contains room for 32k (1kx1024 bytes or characters) of continuous CMOS RAM and 28k of ROM, 16k of which is supplied as the MicroWorld BASIC interpreter.

The CORE board contains its own voltage regulator and handles all the details of 'powering down' the CMOS RAM by itself. Thus when the power to the MicroBee is off, it should be possible to unplug a CORE board and still retain all of the memory. A **Battery Backup** voltage of between 3 and 5 volts is required to keep memory intact after power-down.

A BB rail has been supplied which takes its power from the 5 volt rail when the power is on through a 1N914 diode, and the battery through an OA97 diode when the power is off (Germanium diodes have low forward voltage drops giving longer battery life).

This BB rail supplies all of the RAMs, all of the CMOS multiplexers, and the 4584 or 74C14 schmitt trigger device which detects power-down conditions and battery voltage level.

The power down circuit works by monitoring the voltage of the unregulated input to the regulators through a potential divider. When the unregulated voltage drops below 7.5 volts, the 4584 device changes state and discharges the 0.22 uF capacitor through the diode. This holds the RESET* line low, which both holds the processor in the reset state, and switches the RAMs over to the power-down mode, under which they are protected from spurious writing from the dying Z80.

The 0.22 uF capacitor holds the MicroBee reset for about 1/3 of a second after any power interruption, and also provides the 1 second delay before the 'REPROMPT' key has any effect (it must slowly

discharge the capacitor first through the 680 kohm resistor). Such a reset delay is not necessary for a BASIC only system, but if a MicroBee was for example running CP/M, a keyboard mounted reset switch **MUST** be protected in this way.

Another gate of the 4584 and a BC558 act as both a 'power on indicator' and a 'battery dead' warning. If the power is on, but the voltage of the battery drops below about 2.3v, then the 4584 gate will switch so as to turn the LED off, indicating lack of battery power. This dual purpose indicator is not available on the MicroBee kit version.

The MicroBee memory setup is fairly conventional in most ways other than the BB circuits.

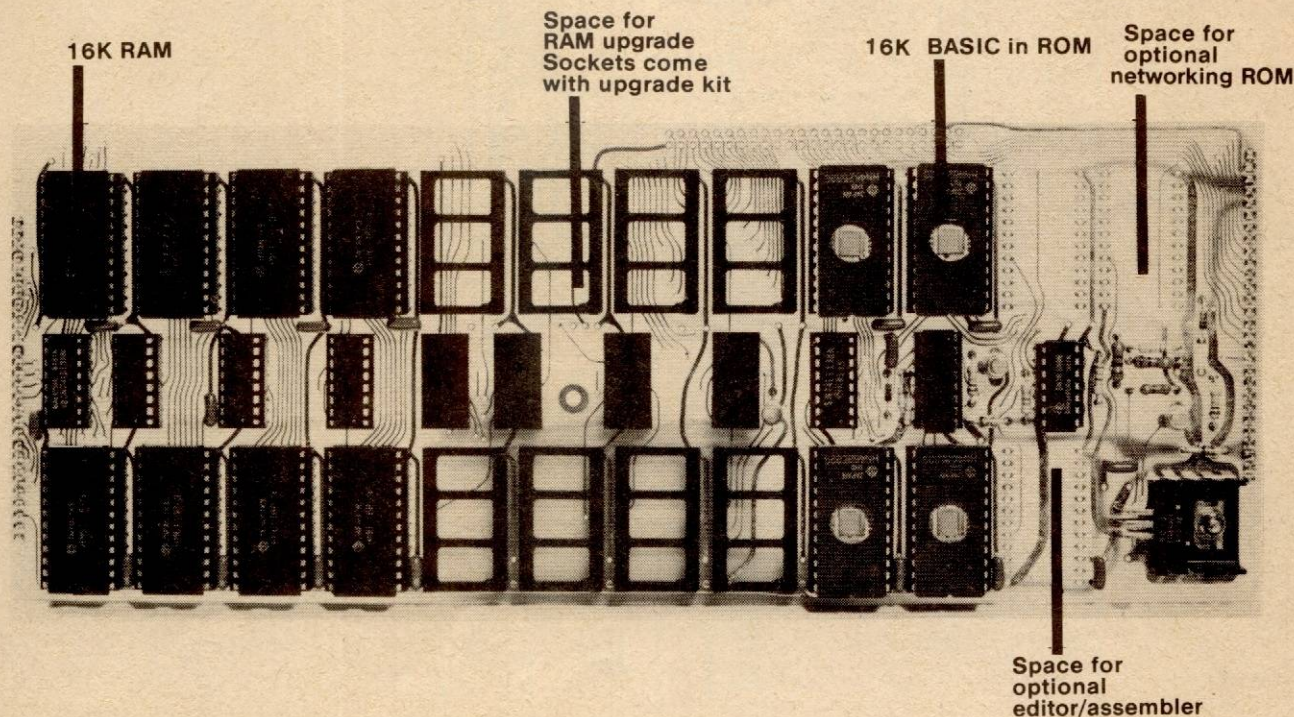
The A0-A10 address lines from the processor connect to all the 24 pin memory type devices on the CORE board in parallel, as do all the data lines from the CPU.

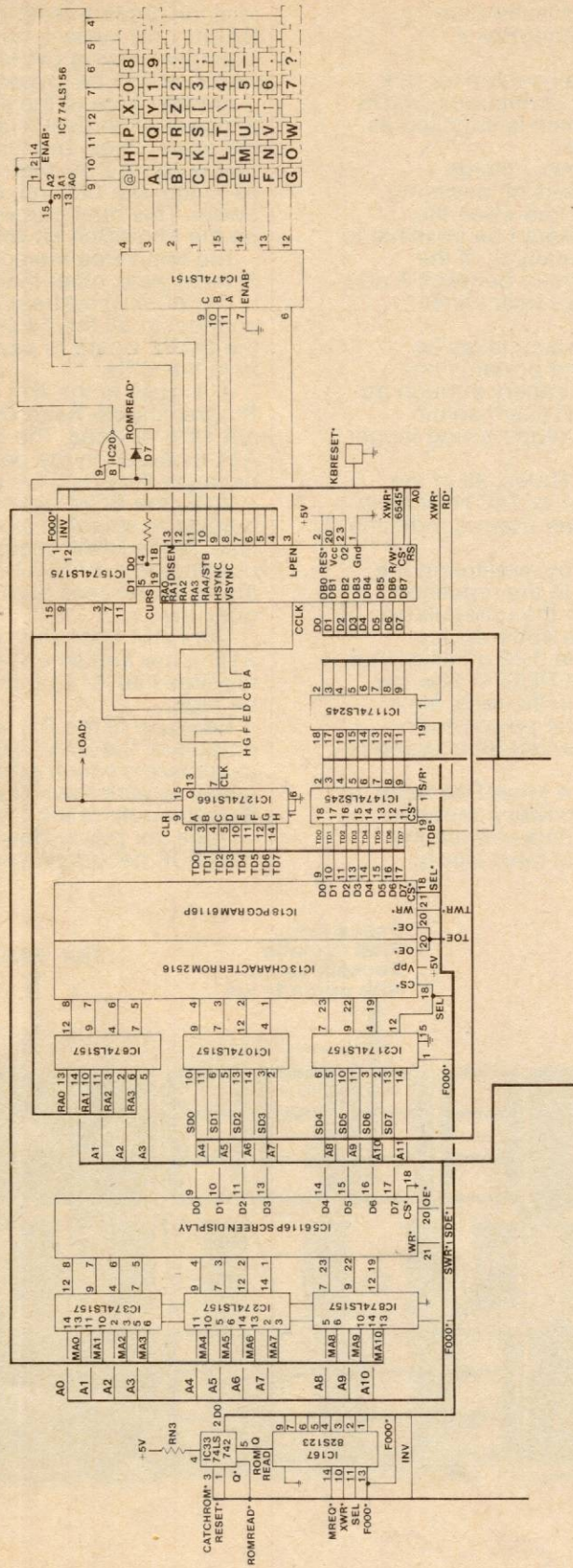
A11 goes to the A11 input of the 2532 **Erasable Programmable Read Only Memories** because these are 4k*8 devices. The 6116 2k*8 rams need to have A11 included in the decoding chips (IC2, IC14) because they only have half the capacity of a 2532.

IC26, the ROM decoder provides active low outputs to select at most one of the 2532s at one time. Since ROM occupies memory from 8000H upwards in a BASIC board, IC26 is enabled when the A15 line goes high, MREQ* goes low, and RD* goes low.

The RAM decoders, IC2, IC14 are enabled when A15 is low and when MREQ* is low. IC2 further requires that A14 be low, while IC14 requires A14 to be high.

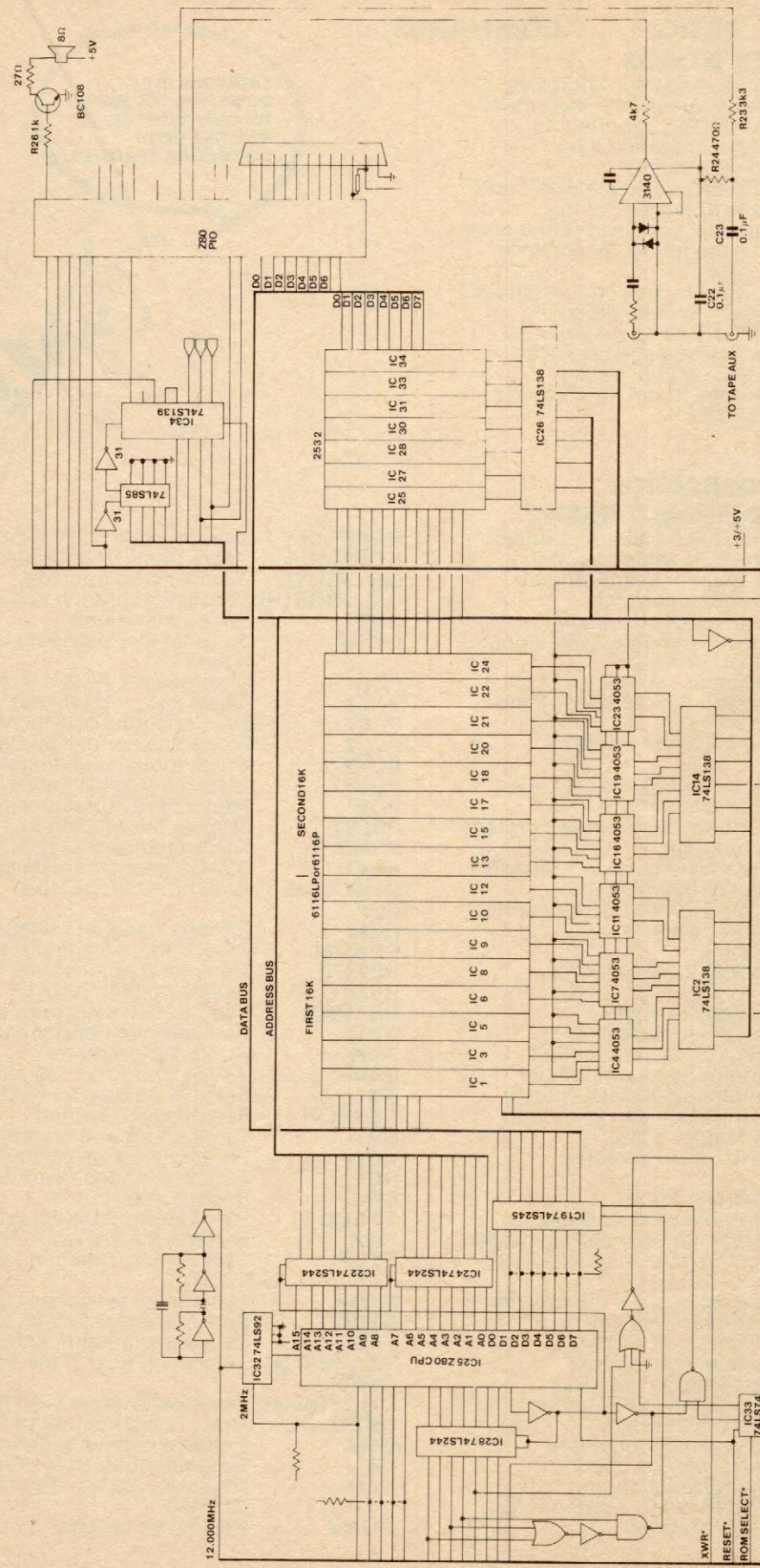
Because of the BB requirements, the active low outputs of the RAM decoders pass through CMOS multiplexers which provide sufficient speed at a low power requirement. These devices switch the **Chip Select** inputs of the CMOS rams between BB voltage (for power-down) and the outputs of the 74LS138 decoders (when running).





MICROBEE SCHEMATIC DIAGRAM

Note: Due to publishing deadlines, this circuit diagram is not complete. Complete circuit diagrams will be included with each MicroBee sold.

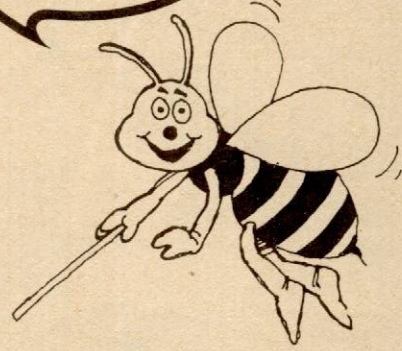


MICROWORLD BASIC: COMMANDS, FUNCTIONS AND KEY WORDS

Now before you can start to use your MicroBee, it is a good idea to familiarise yourself with the actual STATEMENTS and COMMANDS supported by MICROWORLD BASIC. If you are a first time computer user you should refer to the MICROWORLD BASIC REFERENCE MANUAL and, of course the program ideas at the end of this booklet. However, if you are still confused don't give up, the light is at the end of the tunnel (or whatever they say!).

MICROWORLD BASIC is a very powerful extended BASIC conforming as closely as possible to the proposed ANSI standards for BASIC. As you may already realise there are many 'dialects' of BASIC and as such you may have to slightly modify programs from various sources to run them under this particular version of BASIC.

Compare my powerful 16K BASIC. Its very 'friendly', and has error correcting features.



STATEMENTS and COMMANDS supported by MICROWORLD BASIC:

ABS	Produces ABSolute value of a real no	NEXT var	Used to exit a FOR...NEXT loop
ASC	Returns the ASCII value of a string	NORMAL	Clears the VDU attributes
ATAN	Returns the geometric arc-tangent	ON...GOTO	Conditional branching
AUTO	AUTOMatic line numbering	ON...GOSUB	Conditional branch to a subroutine
CHR	Converts number codes to strings	OUT	OUTputs to a port
CLEAR	Erases values of variables and strings	OUT#	OUTputs to a programmed data stream
CLS	Clears the VDU screen	PCG	All print uses PCG characters
COS	Returns geometric cosine	PEEK	Gets a byte from memory
CURS	Used to position the CURSor on the VDU	PLAY	Sounds a tone on the internal speaker
DATA	Provides the DATA for a READ statement	PLOT	Draws lines under graphics
DELETE	DELETes lines from BASIC source	POKE	Writes a byte of data into memory
DIM	Used to DIMension arrays	POINT	Tests if a point is set on the VDU
EDIT	Enter EDIT mode to change a program	POS	Returns the cursor position
END	Terminates program	PRINT	Outputs information to the VDU
EXEC	Auto start for machine code program	PRMT	Alters prompt character for INPUT
EXP	Returns natural logarithm	READ	READs from DATA statements
FLT	Converts INTEGER into REAL number	Relational ops	<, >, or = can be used
FN	User defined function	REM	REMark used for comment
FOR...NEXT	Controls looping	RENUM	RENUMbers the lines in a program
FRACT	Returns the fractional part on a number	RESET	Turns off a point on the VDU
FRE(0)	Returns total memory left	RESTORE	Resets pointer in DATA statements
FRE(\$)	Returns total string space left	RETURN	Used to RETURN from a GOSUB
GOSUB	Transfers control to a subroutine	RND	Random number generator
GOTO	Transfers control to a line number	RUN	RUNs the program
GX	Special search and replace routine	SAVE	SAVES a program on cassette
HIRES	Selects High RESolution graphics	SD	Sets the number of decimal places
IF...THEN..	Conditional test for a line	SEARCH	Searches for a string
INPUT	Used to INPUT data from the keyboard	SET	Turns on a point on the VDU
IN	INputs data from a port	SGN	Returns the sign on the expression
IN#	Selects programmed data streams	SIN	Returns the geometric sine
INT	Converts a real number into an integer	SPC	Prints a space
INVERSE	Inverts output to VDU to black on white	SPEED	Slows down the output to VDU
INVERT	Inverts one graphics dot on the screen	SQR	Returns the square root
KEY\$	Returns character from key board	STEP	Sets the increment in FOR...NEXT loops
LEN	Returns the length of a string	STOP	Terminates execution of a program
LET	Assigns a value to a variable	STR\$	Sets aside memory for strings
LIST	LISTs the program to the VDU	TAB	Tabulates the print across the page
LLIST	Same as above but to the printer	TRACE(ON/OFF)	Traces the flow of the program
LOAD	Loads a file from cassette	UNDERLINE	Does just that on the VDU
LOG	Returns the common logarithm	USED	Returns the number of PCG characters
Logical Ops	NOT, OR, AND can be used	USR	Call to machine code routine
LORES	Selects LOW RESolution graphics	VAL	Returns 'value' of a string
LPRINT	Same as PRINT but to a line printer	VAR	Used to pass the arguments in a GOSUB
NEW	Erases the old program	ZONE	Sets width between ,, in PRINT

CONDENSED MICROWORLD BASIC MANUAL

In MICROWORLD BASIC, statements and commands preceded by a number are entered into memory and become part of the current program. Line numbers may range from 1 to 65534. If a statement or command is entered without a line number, it is executed directly. Multiple statement lines are created by using the colon as a separator.

AUTO int1, int2

This command causes the BASIC to automatically insert line numbers into the file as each line of code is entered. The user may provide one of two parameters after the word auto, brackets are optional. If one parameter is given it will be the first line number inserted. If two parameters are given, the second will be used as a step between line numbers. If no parameters are given the program defaults to line 100 with a step of 10.

To exit from the AUTO mode, simply type a null line («CR» only). If called again, AUTO will restart at the number that it finished the last time.

AUTO mode will automatically edit lines which already exist, so when an occupied line is reached, the edit mode is invoked. To exit this AUTO-editing feature, use the break key.

CLEAR

This command erases all values of all variables and strings and also erases any data structures such as array DIMensions.

CLS

This function clears the VDU screen and places the cursor at the top left hand corner, and then turns it off for elimination of the cursor in graphics work.

CURS int-exp

CURS int-exp1, int-exp2

The first form of this command places the cursor at the position specified by the integer expression. Position 0 is at the top left hand corner and 1023 is at the right. The expression cannot exceed 1023 without causing an error.

The second form of this command allows x-y cursor addressing, where int-exp1 specifies the column number from left (1) to right (64), and int-exp2 specifies the vdu line number from the top of the screen (1) to the bottom line (16).

DATA expr1,expr2,...
Provides data for a READ statement. Data must agree in mode to the corresponding READ variable. Note that data values may be expressions as well as constants. DATA statements must appear singly on a line.

Example

```
10 READ A,B0
20 DATA 10,2*6.4
```

after execution A will be 10 and B0 will be 12.8

DIM list

Used to set up storage for arrays of integer or real numbers. Arrays may have one or more dimensions. The dimension arguments MUST be integer values.

Example

```
10 LET I=10
20 DIM A0(5),B1(10,10*1)
```

EDIT line number

This command edits an already existing line number in the current users program. When in EDIT mode the line is printed onto the VDU and the cursor is placed at the left hand end of the line.

The following keys have special meaning during EDIT

CONTROL S (¶S) Moves the cursor to the right

CONTROL A (¶A) Moves the cursor to the left

CONTROL W (¶W) Moves the cursor one word to the right

DELETE This key deletes the character under the cursor.

«CR» (CARRIAGE RETURN) Terminates the EDIT and the altered line is entered into the program file, just as if it were a new line.

any other key Pressing any other key will cause that character to be entered into the line to the left of the cursor.

END

Statement used to terminate program execution. No message is printed. Return is made to the command mode where the prompt '>' is made.

EXEC

This command will jump to the auto-start address of the machine language tape which was last loaded into the machine. If no tape has been loaded, an appropriate error message is issued.
FOR var=expr1 TO expr2 (STEP expr3)

FOR...TO statements are used to control looping. Integer indexed loops are substantially faster and require less memory than real indexed loops. Real loops do permit the fractional steps. If the output STEP is not specified, the default value is 1.

GOSUB ([expr1,expr2,...]) line no

Transfers execution to a subroutine with a label equal to the value of the integer expression, 'line no'. Such a GOSUB is referred to as a 'computed GOSUB'. The optional expression list may be used in conjunction with a VAR statement to pass values to the subroutine. The number of expressions or arguments or arguments must not exceed the number of variables in the VAR statement. (See VAR below). Also, arguments must agree in mode to the VAR list. The effect is similar to performing a number of LETs before the GOSUB.

Subroutine nesting is permitted. Exiting a subroutine with other than a RETURN will cause random data to remain on the stack, thus wasting memory. Examples:

```
10 GOSUB 100
20 PRINT "END":END
100 PRINT "HERE IS THE "
110 RETURN
```

This short program immediately transfers to the subroutine at 100 and prints the message 'HERE IS THE '. On encountering the RETURN, execution transfers back to line 20, the statement following the original GOSUB. Here the word 'END' is printed and the program terminates.

GOTO int-exp

Transfers program execution to the line number given by the evaluation of the integer expression, int-exp. If the value of the int-exp is not a valid line number, an error condition results.

Example:

```
10 PRINT "TESTING..."
.....
60 GOTO 10
```

When line 60 is executed, unconditional branching to line 10 occurs.

GX

The GX command provides a global search and replace facility which allows easy modification of things which occur many times throughout a BASIC source program.

The general form of GX is:

GX/string1/string2/

The BASIC will then search from the start of the program for lines containing string1, the search string. If any such are found, the line containing that string is listed, with the exact position of the found string highlighted. If it is desired to replace string1 with string2, the period key is pressed. Any other key will not replace that occurrence of string1.

The BASIC will then continue to look for more references until the end of the program is found.

HIRES

This command initialises the scratch RAM used for PCG graphics, and must be used in the program before either SET or RESET etc. are used. HIRES will wipe the screen, but will not affect the actual cursor position.

IF expr rel-op expr **THEN** statements (**ELSE** statements)

IF...THEN is used to cause conditional execution of the statement or statements following the 'THEN' or optional 'ELSE'. The relational operator, rel-op, may be < (less

than), > (greater than), = (equal to) or a combination of any two of these.

The statement/s to the right of the 'THEN' are executed only if the relational test is true. Otherwise, either the next numbered line or the statements to the right of the 'ELSE' are executed. Examples make this clearer.

```
10 IF I<6 THEN 60
```

```
20 PRINT "YES"
```

```
60 PRINT "NO"
```

If I is less than 6, branching to line 60 will occur. If I is equal to or greater than 6, the program continues at line 20.

INPUT (literal {;} {;} var {var}....;)

Statement used to input data from the keyboard and to assign this to program variables. The optional literal can be used to print a message just before the inputting is to begin. The literal, if used MAY be followed by a comma, or a semicolon.

A semicolon placed after the last input variable will inhibit the «CR» and «LF» that normally occurs after an INPUT statement. Example:

```
10 INPUT A, B, A0
20 PRINT A+6, A0*A0
30 END
```

The program immediately outputs a '?' and waits for the user to enter the appropriate data separated by commas. If the user enters a «CR» before supplying all the required data, BASIC will print '??' indicating that more data is required. If part of the user data is illegal, MICROWORLD BASIC will print 'R?' indicating that data should be reentered from the start of the input statement.

INVERSE

All PRINT output after this will be black on white. If the last display mode was a HIRES, LORES, or PCG, the screen will be cleared first.

LET var=expr

Statement used to assign the computed value of an expression to the variable to the left of the equal sign. The entire expression must agree in mode, either in integer or real, with the assigned variable. LET is optional and can be deleted.

Example:

```
10 LET I=2*6+3 After execution, I has the value of 15.
```

LIST

Lists on the VDU the current program in whole or in part, depending on the optional specification. Below are the 5 variations of the LIST command:

```
LIST LISTS entire program
LIST I1 LISTS only line numbered I1
LIST I1, LISTS from line I1 to end of program
LIST I1,I2 LISTS from start to line I1
LIST I1,I2 LISTS from line I1 to I2 inclusive
```

Listing can be stopped at any time by striking the BREAK key and may be paused by CONTROL S (¶S) and hitting any other key will continue the listing.

See also SPEED

LLIST

Same as LIST but output goes to the 'printer' data stream which can be directed to RS232, cassette or vdu output.

LOAD

Loads a file from cassette tape. The file must have been created by MICROWORLD BASIC using a SAVE command. BASIC files:

MICROWORLD BASIC will load the first file encountered on the tape with a 'B' file type (that is any file created with the BASIC SAVE command) if no filename is specified. If a filename is specified, each name will be shown as it is reached on the tape, but BASIC will continue searching for the correct file.

The filename can be up to six characters enclosed by " double quotes as for the SAVE command.

MACHINE LANGUAGE files:

MICROWORLD BASIC accepts files of filetype 'M' as machine language files which are loaded into the memory and optionally auto-executed (program starts automatically when loaded). Otherwise, the user may use EXEC to start up the program.

LOGICAL OPERATORS

NOT, OR and AND can be used with relational operators in IF statements or to perform logical operations with integer arithmetic. Examples

```
10 IF A<2 and B>6 OR C<10 then 80
```

Note the operation precedence is

(1) NOT

(2) AND

(3) OR

Parentheses may be used to alter precedence.

LORES

Lores will initialize the graphics RAM and prepare to receive set, reset, plot etc. commands for low resolution graphics mode (128*48). Note that the screen is not cleared by this command, and inverse or underlined characters on the screen when this command is executed will change to random graphics characters.

LPRINT

Same as PRINT but on the 'printer' output stream.

NEW

Erases current program and variables, then WARM starts the interpreter.

NORMAL

Clears INVERSE and UNDERLINE and PCG modes and returns PRINT output to normal format.

ON...GOTO

ON int exp GOTO line-no1, line no2, (line-no3), etc

A conditional branch that depends on expr. in the following way:

int-exp=1 Branch to line-no1

int-exp=2 Branch to line-no2

int-exp=3 Branch to line-no3 etc

The ON-GOTO differs from the computed GOTO in that the line numbers line-no1, line-no2, etc do not have to be calculated from the value of the expr.

If the int-exp is zero or greater than the number of line numbers given, then execution will continue with the next statement.

EXAMPLE:

```
5 INPUT A
10 ON A GOTO 125,230,400,650
```

```
125 REM HERE IF A=1
```

```
230 REM HERE IF A=2
```

```
400 REM HERE IF A=3
```

```
650 REM HERE IF A=4
```

OUT int1,int2

Outputs the value of the integer expression, int2 as a data byte to a PORT with the address given by integer expression, int1. Of course, int1 and int2 must have values between 0 and 255 decimal.

PCG

All print output after this will use the PROGRAMMABLE CHARACTER GENERATOR.

PLAY n (,m)

This command will sound one note from the speaker for multiples of 1/4 second.

There are two forms:

The first form without the optional length will play note number n for 1/4 second.

The second form will play note n for m/4 seconds.

The 25 notes provided are the musical notes taken from the section of a piano starting at the A below middle C and then up for two octaves. The frequencies are reasonably precise and allow simple melodies to be played:

Number	Note	Frequency
0	rest	
1	A	220
2	A#	233
3	B	247
4	C	262
5	C#	277
6	D	294
7	D#	311
8	E	330
9	F	349
10	F#	370
11	G	392
12	G#	415
13	A	440
14	A#	466
15	B	494
16	C	523
17	C#	554
18	D	587
19	D#	622
20	E	659
21	F	698
22	F#	740
23	G	784
24	G#	831

PLOT(I)(R)(H)x1,y1 TO x2,y2 (TO x3,y3 (... TO xn,yn))

The plot command allows graphics commands to be performed automatically along straight lines joining two points. Plot can be used in either LORES or HIRES graphics mode.

Plot will normally be used in the set dot mode which is the default for a PLOT keyword without suffixes.

Lines can also be inverted or reset by the addition of the letters 'I' and 'R' respectively after the PLOT keyword.

The usage PLOTH will plot a line with the Y-axis inverted, but note that it is impossible to have 'PLOTI' for example.

POKE int-exp1,int-exp2

This command writes a byte of data defined by int-exp2 into RAM memory at the address specified by int-exp1. Note both integer expressions are in decimal.

PRINT list

Statements used to output information and data to the console device. The 'list' consists of variables, constants, expressions, quoted literal and special printing functions separated optionally by commas, semi-colons or back slashes (/).

A comma produces zone spacing with the size of the ZONE determined by the ZONE command.

If a PRINT is terminated by a semicolon, the final CR and LF «CR»,«LF» is suppressed. Backward slashes (/) may be used within the 'list' to produce additional CRs and LFs.

The special print functions are TAB(ie) and SPC(ie). TAB moves the print to the position equal to the value of the integer expression. SPC produces the number of spaces determined by the value of the integer expression.

Numeric values may be output in one of two ways: formatted and unformatted. Consider the latter first. Unformatted printing is that used in standard BASIC. If the value is between 0.01 and 999999, it will be printed in ordinary decimal notation.

For smaller values and larger than these, exponential notation is used. Best we resort to examples:

```
PRINT 65+3          Produces 68
PRINT 500000*2      Produces 1.0 E+6
```

The formatted printing of numerical values is useful in business programming where specific fields must be set up to accommodate the printed values. The format specification may appear at any point in the PRINT 'list'. It takes one of four forms:

[int int-exp] Integer format:

The value of the integer expression is printed right justified in a field of width 'int'. 'int' must of course, be greater than the number of digits in the value to be printed plus one (to take care of the sign). If this is not the case, plus signs will be printed in the field.

[Fn1.n2 real-exp] Real format:

The value of the real expression is printed in a field n1 wide and a decimal point n2 digits from the right of the field. The field width must be TWO characters greater than the number of digits printed.

The number of digits printed should not exceed the number of significant digits. Again, plus signs will be printed in the field if the specification is incorrect. [Dn real-exp] Exponential format: The value of the real expression is printed with exponential format in a field n7 wide with n decimal places.

[An int-exp] ASCII format:

Equivalent to the PRINT CHR\$(int-exp). The value of the integer expression is output as its equivalent ASCII character a total of n times. This format is useful for sending special control characters to the output device. It also permits printing long strings of the same character.

The following examples illustrate the points:

```
10 PRINT I,J:A0*2
20 PRINT "OK" TAB(8);
30 PRINT "DONE"
40 END
```

This program would produce the following output for the values, I=2, J=6, A0=12.4 and ZONE set at 14

```
2 6 24.8
OK DONE
```

The format specifications may be linked with semicolons as below:

```
10 PRINT [A6 66];[F8.2 A0]-
20 END
```

For A0=42.6, this program would produce the output

```
BBBBBB 42.60
```

```
READ ((line-no)) var1, var2, var3, ...
```

This command is used to store values from DATA statements into the specified variables. The optional line number is used to reset the data pointer to a specific DATA statement. Example:

```
10 INPUT I
20 READ (10*I+30) A0$
30 PRINT A0$: GOTO 10
40 DATA "MESSAGE NUMBER 1....."
50 DATA "MESSAGE NUMBER 2....."
60 DATA "MESSAGE NUMBER 3....."
RUN
```

```
? 2 «CR»
```

```
MESSAGE NUMBER 1.....
```

```
? etc.
```

An attempt to read more data than available will result in an error.

RELATIONAL OPERATORS

Relational operators enclosed in parenthesis can be used in integer expressions. The values are as follows:

```
TRUE = -1
FALSE = 0
```

Example:

```
A = (6>2) returns -1 for A
```

```
A = (6<2) returns 0 for A
```

REM

Used as the last or only statement of a line to insert user comments in the program. All text between REM and the end of the line is ignored during execution.

```
RENUM { new start [,increment [,start [,finish] ] ] }
```

The renumber command is a complex program which will regularize the line numbering of a BASIC program, changing all references to line numbers in the process. There are five forms of renum:

RENUM

This form will renumber the entire program so that when renumbered, the first line number will be 100 and the rest will increment by 10 from there on.

RENUM n

This form rennumbers the entire program so that when renumbered, the first line number will be n and the rest will increment by 10.

RENUM n,i

This form rennumbers the entire program so that when renumbered, the first line number will be n and the rest will increment by i.

RENUM n,i,s

This form rennumbers the section of the program from the original line number s through to the end of the program, so that the line which was s will have line number n, incrementing by i.

RENUM n,i,s,f

This form rennumbers the section of program between original line numbers s and f so that the line which was s will have line number n, incrementing by i.

RESET (H) int-exp1,int-exp2

This function 'turns off' a graphics point on the VDU screen. The command is the same for both HIREs, and LORES graphics modes, however the maximum co-ordinate values differ.

The int-exp1 must be in the range 0 to 127 and specifies a position across the screen, int-exp2 is in the range 0 to 63 and specifies a position down the screen for LORES graphics.

int-exp1 must be in the range 0 to 511 across the screen to the right, and int-exp2 must be in the range 0 to 255 up the screen for HIREs graphics.

RESET 0,0 resets a point at the bottom left hand corner of the screen whereas RESET 63,24 resets a point near the centre of the screen (assumed LORES mode).

SAVE

This command saves a program to tape at either 300bd or 1200 bd. The 1200 bd speed is selected using an 'F' suffix after the save, e.g.

SAVE "FRED" ; will save at 300bd with filename FRED
SAVEF "JANE" ; will save at 1200bd with filename JANE

Note that the filename is obligatory and must be 0..6 characters in length. See LOAD for retrieving programs from tape.

SD int-exp

Used to set the number of significant digits used in REAL calculations for greater precision or speed. SD can be set from 4 to 14 places. The default value is 8.

As an example, for drawing a circle on the screen, you would need quick, but not very accurate trigonometric functions, so use SD 4, but for solving trigonometric equations, use SD 8.

SET(H) int-exp1,int-exp2

Set is equivalent to RESET, but will turn a graphics dot ON, not off.

SPC

SPC(int-exp) is used to direct PRINT to output int-exp spaces before the next item in the list. SPC is different from TAB(int-exp) because SPC(n) will always print n spaces no matter where the cursor is when it is invoked.

SPC must appear only in a print list.

SPEED

Slows down the VDU output by introducing a delay between characters. FORMAT: SPEED n where n=0 to 255 (0 is the fastest and the default.)

STOP

Used in a program to terminate execution. The following message is printed:

STOP AT line-no

Where line-no is the line number where the STOP was encountered.

This command, although performing roughly the same function as an END command, is normally used during program fault-finding. Execution can be restarted by using the CONT command.

TAB(ie)

TAB is used to direct PRINT to start at a particular point on a line. The argument must be an integer and if the required TAB has already been passed over, the program will ignore the integer argument.

TAB must be used in a print list.

TAB(0) is equivalent to TAB(255).

TRACE ON, TRACE OFF

When TRACE is turned ON the line number of each line executed is listed on the VDU between square [] brackets. TRACE OFF removes the facility after troubleshooting is over.

UNDERLINE

The underline command sets up the VDU to print underlined characters in all print statements after this one. If the previous display mode was a graphics one (PCG, HIREs or LORES), the screen will also be cleared first.

VAR (var1,var2,...)

The first statement of a subroutine to which arguments are passed. The variables in the VAR list receive from the calling - GOSUB. The variables MUST correspond in position and mode to the expression in the GOSUB. These variables are common to the main program and thus can be used to pass values back to the main program.

For advanced programs, it should be noted that the VAR statement can be used to simulate the PRINT USING type of command found in other BASICs. Such statements permit the PRINT statement to be used with several different sets of variables.

```
10 INPUT P
20 GOSUB [P,"CATS"] 100
30 GOSUB [4*P, "CAT FEET"] 100
40 GOSUB [4*P*5, "CAT CLAWS"] 100
```

```
100 VAR (D,D0$):PRINT "MY HOUSE HAS ";D;D0$:RETURN
```

For P=6 this program will produce the following output:

```
MY HOUSE HAS 6 CATS
MY HOUSE HAS 24 CAT FEET
MY HOUSE HAS 120 CAT CLAWS
```

ZONE (int)

The statement of the integer 'int' sets the ZONE width applied when commas are used in PRINT statements. The value of 'int' may range from 1 to 16.

I support all these common functions.



FUNCTIONS IN MICROWORLD LEVEL II BASIC

MICROWORLD BASIC was developed as an interpreter for use in business and game applications. The choice of functions reflects this interest. Functions are either real or integral and produce either real or integral results depending on the type. Integer functions should only appear in integer expressions and real functions only in real expressions.

Note: A function is different from a command in that it must always appear on the right hand side of an equation. Example:

```
N = PEEK (300) Gets the byte from location 300 and PUTS it into variable N.
A0 = RND * 500 Generate a random number and put it into variable A0.
```

PRINT PEEK (0) This is an exception which still follows the rule (above): it means, get the byte from location 0 and store it in a temporary store, then PRINT the temporary store.

REAL FUNCTIONS

2ABS (real-exp)

Produces the absolute value of the real expression, if it is positive, does nothing. If -ve then returns same value but positive.

ATAN(real-exp)

This function returns the trigonometric arc-tangent of the real expression evaluated in RADIANS. (Accuracy is 0.01%) To convert RADIANS to DEGREES simply multiply the result by 57.29577951 (to the correct number of significant digits).

COS (real-exp)

This function returns the trigonometric cosine of the real expression, assumed to be expressed in RADIANS.

EXP (real-exp)

This function returns the value of e (2.718281828) raised to the 'real-exp' power. Accuracy is .01% for normal range. This is the equivalent to taking the NATURAL antilogarithm of the expression.

FLT (int-exp)

This function converts integer expressions into real numbers.

FRACT (real-exp)

This function returns the fractional part of the real expression.

Example:

```
PRINT FRACT(6.84) produces 0.84
PRINT FRACT(120.) produces 0.0
```

FRE(O)

This integer function returns a number representing the total memory available for program and variable storage. If you are running with 48K of RAM this number will come out negative! Don't worry, this is only a restriction caused by the method used to represent integer numbers. In the immediate mode use PRINT FRE(0) to give an immediate indication of memory left at any particular point in time.

FRE(\$)

Gives the amount of 'string space' available as a real number. Use STRS (int-exp) to set up more 'string space'.

LOG (real-exp)

This function returns the common logarithm of the real expression. The natural logarithm (to base e) can be found by using 2.30258*LOG (real-exp).

ANTILOGS can be calculated by 10↑(real-exp)

RND-

The random number generator, returns a real number between 0 and 1. To get other sized random numbers see examples below:

RND*150 Returns a number between 0 and 150.

RND*200-100 Returns a number between -100 and +100

INT (RND*6)+1 Returns an integer between 1 and 6

SGN (real-exp)

This function returns one of three values as follows:

-1 if real-exp < 0

0 if real-exp = 0

1 if real-exp > 0

SIN (real-exp)

This function returns the trigonometric sine of the real expression considered to be in radians.

SQR (real-exp)

Produces the positive square root of the value of the real expression. Note in this particular case the real-exp must be a positive number.

INTEGER FUNCTIONS-

INT (real-exp)

Converts the value of the real expression into an integer.

IN (int-exp)

Inputs a data byte from the input port with the address given by the value of the integer expression.

PEEK (int-exp)

Reads the data byte stored in memory location addressed by the value of the integer expression.

POINT (int-exp1,int-exp2)

POINT returns a value depending on whether the specified dot is set or not.

If the co-ordinates are out of range for the relevant graphics mode, the value returned is -1.

If the specified point is set, POINT returns -1.

If the co-ordinates are in range, and the dot is not set, POINT returns 0.

The values 0 and -1 were chosen to correspond to the two boolean values for an integer which can be used in an IF statement directly, so

```
10 IF POINT(X,Y) THEN LET A=-A:B=-B
ELSE SET X,Y
```

will negate A and B if the dot is set, and if it was not set, this statement will set it.

POS-

This integer function returns an integer representing the cursor position on either the VDU or the line printer.

SEARCH (str-exp1,str-exp2,int-exp)

String str-exp1 is searched for «int-exp» th occurrence of substring str-exp2. The integer value returned is the position of the beginning of the substring, if found, or zero if not found. The default value of int-exp is 1. Examples always help to illustrate the point.

```
A0$ = "HOW ARE YOU" define the string
```

```
PRINT SEARCH (A0$,"ARE") look for the 1st occurrence of "are"
```

```
it occurs from character 5 onwards.
PRINT SEARCH (A0$,"O",2) find the second 'O' in A0$
```

10 it occurs in the 10th character
NOTE: Integer functions may be used
directly in print statements.

USED-

This function returns the number of PCG characters used when in HIREG graphics mode.

This information is very useful because when the graphics subroutines attempt to use more than 128 PCG programmable characters to draw the hires graphics, an error is generated which aborts the program.

String functions

CHR \$(int_exp)

This function returns a string consisting of the ASCII character which corresponds to the integer expression given by int_exp.

e.g. ring the bell: PRINT CHR\$(7)

KEY \$()

This function of no arguments allows inspection of the MicroBee keyboard without stopping as would be necessary for an "input" statement.

If no key has been pressed, the null string is returned. If a character is available from the keyboard, the string returned consists of the character pressed only.

STR \$(num_exp)

STR converts integers or reals into strings representing a 'human readable' string of digits such as would be seen from a 'PRINT' statement.

e.g. A1\$=STR\$(32.78) assigns the string '32.78' to the string variable A1\$.

USER DEFINED FUNCTIONS

MICROWORLD BASIC also has provision for the user to define special functions for himself. The basis is the FN statement.

FNn = expr

The defining function can be real or integer or both depending on its construction. The FORMAT is FNn where n is an integer between 1 and 7. If, when defining an expression a dummy variable is required, a '#' sign should be used to indicate where the dummy variable should appear. Example:

```
10 FN2 = 3.14159 * # * 1.8
```

When a reference is made to the defined expression in a program, it appears as FNn(expr). The value of the argument would be passed into each '#' symbol of the defining statement. Example:

```
10 FN0 = # + #
```

```
40 PRINT FN0(6)
```

This program will print the value 12 when line 40 is executed.

Some special functions:

FNn SIN(#) / COS(#) This real function calculates the trigonometric tangent of an angle. The angle must be expressed in RADIANS.

FNn ATAN(# / SQR(1 - # * #)) This real function calculates the inverse trigonometric sine (ARC SINE).

MACHINE LANGUAGE SUBROUTINES- USR (int-exp1 ,int-exp2)

This integer function produces a call to a machine language routine given by the integer expression int-exp1. The value of int-exp2 is passed in the BC register pair. The int-exp2 is optional, in which case BC will contain zero. The machine language program may use all registers, but the stack must be managed so that PUSHes and POPs (if that's how you say it) are equal in number. In this case a machine language return instruction will produce reentry back to MICROWORLD BASIC. To pass a value back it should be placed in the B and C registers before return.

Learn as you practice
with my easy guide
into BASIC programming



GETTING STARTED WITH MICROWORLD BASIC

About time! Let's get down to what it's all about. This section is included to help you get started. It is not an exercise in how to write well-structured BASIC programs and is by no means an exhaustive description of how to use your MicroBee.

The first step, if you have not done it already is to connect up your MicroBee as outlined earlier. You must use a TV monitor equipped for 1V video with combined sync or, alternatively an ordinary TV set equipped with a modulator. If you are in doubt contact the Technical Support Section at APPLIED TECHNOLOGY (Phone (02) 487 2711) for a detailed application note. Note if you have an ordinary TV set such as a small portable black and white set any TV service technician will convert it for you at a modest cost.

The next step is to connect the MicroBee power pack to the MicroBee console using the cables provided. NOTE: use only the cables as supplied as incorrect connection could result in damage to your MicroBee and may invalidate your warranty. Connect the power pack to the console and connect the video lead to the TV monitor. If all is well plug in the TV and the MicroBee power pack to the 240V main supply and switch on. You should hear the speaker sound out a tone indicating that BASIC is running.

On the TV screen you should see some form of display indicating that the MicroBee VDU is outputting characters. If it does all is well. If not don't panic, just recheck everything carefully.

Now for the best part. Because your computer has non-volatile memory and if it has been used beforehand there may be a program already there! Just type the following on the keyboard:

RUN and hit the RETURN key. Note that we will now abbreviate this to
RUN «CR»

Also note that MICROWORLD BASIC doesn't care if you type in lower case or upper case. So just type 'run' and hit «CR».

If a program is in memory just follow the instructions on the screen. If not we will have to move to the next section, won't we.

Before we do however we should think about connecting a tape recorder (a low cost portable one is excellent) to the MicroBee so that we can save programs and reload them at a later stage. Again refer to section 2 for the connection details.

SOME FUN PROGRAMS

Now for the good bit! Let's really start to learn about MicroBee the easy way. Don't forget that if you don't understand a particular point refer back to the manuals as this is the most effective way to learn how to get the most from your MicroBee.

PROGRAM 1: THE FIRST ONE!

Type NEW «CR» This clears out memory for a new program

Type AUTO «CR» Sets up auto line numbering and the screen will respond with

00100 - (you write here)

Now type the following on the line after the 100-

PRINT "HELLO, WHAT IS YOUR NAME "; and press RETURN «CR» as this is the way we enter every line in BASIC.

The computer will respond with

00110 _

Type INPUT A1\$ and press «CR»
the computer will respond with

00120 _

Type PRINT "GLAD TO MEET YOU , ";A1\$ «CR»
The computer will respond with

00130 _

Type END and press «CR»
Press «CR» again to exit AUTO mode
The computer will respond with

>_

Then type LIST and press «CR»
The computer should respond with

```
00100 PRINT "WHAT IS YOUR NAME ";
00110 INPUT A1$
00120 PRINT "GLAD TO MEET YOU , ";A1$
00130 END
```

If it does, all is well. If it doesn't, the best approach is for you to start again and type NEW. Later on you should look at the MICROWORLD BASIC manual under the section on EDITING a BASIC program so that you can avoid this time consuming step.

Now, assuming you have this program and the computer outputs the correct LISTING you should type RUN «CR» and watch what happens. Notice that the computer outputs the question WHAT IS YOUR NAME ? , (your own name) Fascinating!!! You are now a programmer, you have programmed the computer to ask a question and, after receiving an input from the key board, it has combined your own name into its next reply. Your MicroBee is under your control and already has intelligence!

Now let's clean up the program a bit.
Type 50 CLS «CR» This will clear the VDU screen before

we run the program

```
Next type
130 PLAY 10,10 «CR»
140 GOTO 50 «CR»
```

and then type LIST «CR»

the computer should respond with

```
00050 CLS
00100 PRINT "WHAT IS YOUR NAME , ";A1$
00110 INPUT A1$
00120 PRINT "GLAD TO MEET YOU ";A1$
00130 PLAY 10,10
00140 GOTO 50
```

Now press RUN «CR» and the program will execute again.

Notice that the screen now clears before the computer asks the question. Much tidier isn't it. Notice that now we play a tone for a short time and repeat the program. We have now caused the computer to loop back, ask questions, answer them, play a tone after clearing its own screen. Not bad after 10 minutes programming!!!

Now for a couple of other tricks. Let's try RENUMbering the program. Recall that we have used line numbers for the BASIC source file and until we started editing, the lines started at 100 and incremented by 10 lines at a time. Notice also that the program still ran but let's tidy things a little bit by introducing new feature: LINE RENUMBERING. With the program file still in memory type RENUM 1,1 «CR» and then type LIST «CR» and notice that the program now starts at line number 1 and numbers the lines consecutively.

Perhaps you would like to save your program on cassette tape to avoid having to retype it at a later date. Connect a small cassette tape recorder to the appropriate leads from the MICRO-BEE power pack, insert a blank cassette, plug in (and don't forget to turn on!) to the 240 Volt supply, press down the PLAY and RECORD buttons and make sure the tape runs past the clear leader section till the recording head is well onto the oxide (brown) section of the tape.

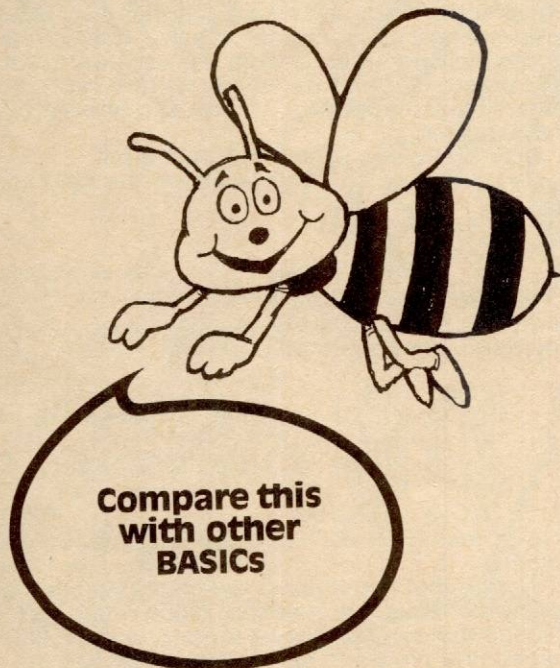
Then type
SAVE "PROG1" «CR»

Notice the display will show a '*' symbol which will slowly flash on and off as the tape is being recorded. When the dumping of the program is finished, the MicroBee will sound a tone (the same as ↑G) and reprompt ready to accept the next instruction.

To reload the program at a later time, connect the recorder as before but DO NOT PRESS PLAY or RECORD.

Type
LOAD «CR»

Then press the PLAY button on the tape recorder (rewind the tape to the start if you have not already done so) and watch the VDU screen. As the tape starts to load you will see the file name appear on the screen and as the loading continues you will see the '*' slowly flashing indicating that the load is in progress. When the program has finished loading the loudspeaker will sound a brief tone and the BASIC will return to the prompt (>) ready to accept the next instruction.



BASIC applications programs

About these programs ...

The only real way to learn how to be proficient in programming is to practise. One easy way to do this is to work through specific examples which demonstrate certain programming techniques.

The programs detailed below have been carefully selected to enable you to evaluate MicroWorld BASIC for the MicroBee.

To simplify the typing required to get these programs running, you can eliminate any REM statement and anything following in the same line.

TUNES is a short program designed to demonstrate the music making facilities of the MicroBee using data and read statements. Why not try writing your own tunes.

To add more tunes, start inserting lines at line 80, starting with a
00080 DATA -2
to separate the new tune from the 'on top of old smokey',
and then add your notes as described under 'PLAY' in the BASIC section.

```

00005 DATA "Westminster chimes"
00010 DATA 13,9,11,4,0,4,11,13,9
00011 DATA -2 : REM end of first tune
00020 DATA "Twinkle twinkle ..."
00030 DATA 4,4,-3,11,11,-3,13,13,11
00031 DATA -2
00035 DATA "On top of old smokey"
00040 DATA 4,4,8,11,16,16,16,13,13,13
00050 REM
00060 REM Insert new tunes in here after line 70
00070 REM
00999 DATA -1 : REM end of ALL tunes
09000 READ A$
09010 CLS:PRINT A$:CURS 0
09100 FOR Q=1. TO 0300:NEXT Q
10000 READ A
10009 IF A=-2 THEN 9000
10010 IF A=-1 THEN END
10020 IF A>=0 THEN PLAY A ELSE GOSUB 20000
10030 GOTO 10000
20000 FOR Z=0 TO 20:NEXT Z
20010 RETURN

```

ART is a simple LORES graphics program, which produces quite abstract images on the VDU screen.

```

00100 PRMT()
00110 CLS
00120 CURS 200:UNDERLINE:PRINT "MICRO-BEE
ABSTRACT ART PROGRAM"
00130 PRINT:PRINT:NORMAL:PRINT" Prints
various random patterns of rectangles"
00140 PRINT " Please give me your
requests":PLAY 10,3:PLAY 12,4:PLAY 15,5
00150 INPUT " What is your name, please ? ";N$
00160 FOR W=1 TO 1500:NEXT W : REM Time
delay
00170 INPUT " How many different sized
rectangles (1 to 4) ? ";F
00180 IF F<0 OR F>4 THEN 170
00190 INPUT" How many rectangles do you want
in the picture (10 to 30) ? ";Q
00200 IF (Q<10) OR (Q>30) THEN 190
00210 CLS:LORES :Z=1
00220 PLAY INT(RND*24),4:M=INT(RND*119)
00230 N=INT(RND*39)
00240 P=INT(RND*FLT(F))

```

```

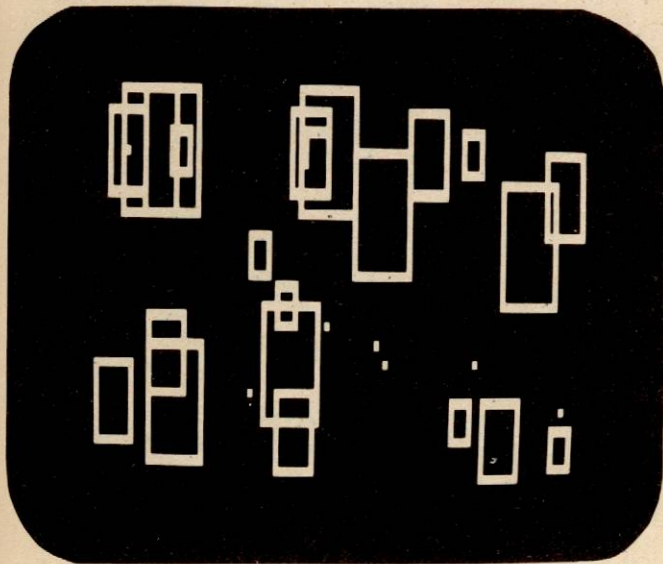
00250 IF M<8 THEN 490
00260 IF N<8 THEN 480
00270 S=N-2*P
00280 FOR R=M-2*P TO M+2*P
00290 SET(R,S)
00300 NEXT R
00320 FOR S=N-2*P TO N+2*P
00330 SET(R,S)
00340 NEXT S
00360 FOR R=M+2*P TO M-2*P STEP -1
00370 SET (R,S)
00380 NEXT R
00400 FOR S=N+2*P TO N-2*P STEP -1
00410 SET (R,S)
00420 NEXT S
00430 Z=Z+1
00440 IF Z=Q+1 THEN 460
00450 GOTO 220
00460 CURS 0:PRINT "Drawn by ";N5$;" age 5"
00470 END
00480 N=8:GOTO270
00490 M=8:GOTO260

```

```

00260 REM - are all deposits entered?
00270 IF A1=0 THEN 350
00280 REM - no, test for validity
00290 IF A1>0 THEN 330
00300 REM - invalid, print standard error, loop to
reenter
00310 GOSUB 830
00320 GOTO 250
00330 D1=D1+A1
00340 GOTO 250
00350 PRINT
00360 PRINT" Enter amount of each cheque not
on statement"
00370 PRINT" (Enter zero when all outstanding
cheques are entered)"
00380 C1=0
00390 INPUT" "; A1
00400 REM - are all outstanding cheques entered?
00410 IF A1=0 THEN 490
00420 REM- test for valid entry
00430 IF A1>0 THEN 470
00440 REM- invalid, print standard error, loop to
reenter
00450 GOSUB 830
00460 GOTO 390
00470 C1=C1+A1
00480 GOTO390
00490 PRINT
00500 PRINT"Account Balance = +$";(E1+D1)-C1
00510 PRINT
00520 PRINT "Enter your chequebook balance";
00530 INPUT B1
00540 PRINT"Enter the amount of service charges";
00550 INPUT S1
00560 REM- test for valid entry
00570 IF S1>=0 THEN 610
00580 REM - invalid. print standard error, loop to
reenter
00590 GOSUB 830
00600 GOTO 540
00610 PRINT
00620 PRINT " Adjusted Account Balance = $";B1-
S1
00630 Z1=E1+D1-C1+S1-B1:IF Z1=0 THEN 710
00640 PRINT
00650 PRINT "Your account is out of balance"
00660 PRINT " Make sure you have included all
transactions against"
00670 PRINT" this account, including automatic
deposits, interest"
00680 PRINT" payments and authorized
withdrawals."
00690 REM
00700 REM-end of program or restart?
00710 PRINT
00720 PRINT"Would you like to rerun this program
for new data Y/N?"
00730 INPUT Z1$
00740 IF Z1$="y" THEN 100
00750 IF Z1$="Y" THEN 100
00760 IF Z1$="N" THEN 870
00770 IF Z1$="n" THEN 870
00780 GOTO 720
00790 PRINT
00800 PRINT " error: enter a valid dollar amount
only!"
00810 PRINT
00820 RETURN
00830 PRINT
00840 PRINT " error: enter a positive valid dollar
amount only."
00850 PRINT
00860 RETURN
00870 END

```

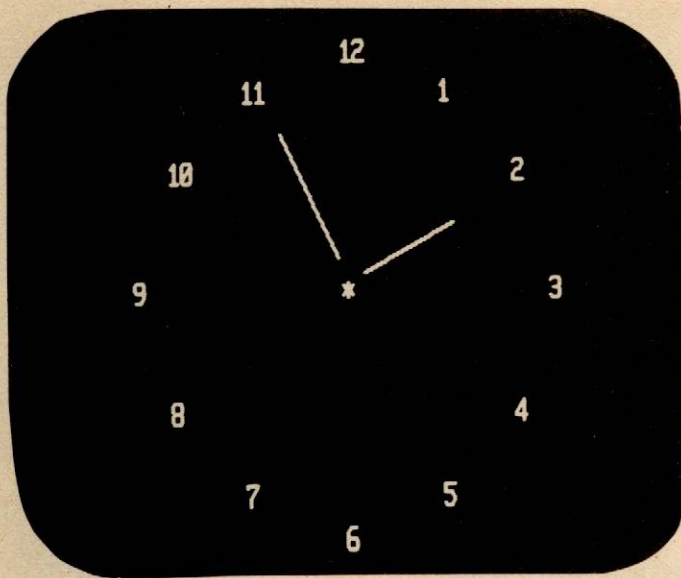


CHEQUE is a program which will help you balance your personal cheque account.

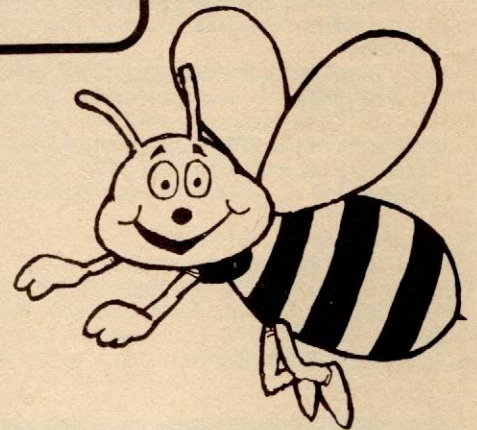
```

00100 CLS
00110 PRINT " Personal Cheque Book
Reconciliation"
00120 REM - determine if positive dollar amount
entered
00130 PRINT
00140 PRINT " What is the closing balance from
the statement ";
00150 INPUT E1
00160 REM- test for valid input
00170 IF E1*100=INT(E1*100) THEN 210
00180 REM- invalid amount. display error, loop to
reenter
00190 GOSUB 790
00200 GOTO140
00210 PRINT
00220 PRINT" Enter amount of each deposit not
shown on the statement"
00230 PRINT" (Enter zero when all outstanding
deposits are entered)"
00240 D1=0
00250 INPUT" "; A1

```



This great clock programme tells you it's time for a MicroBee



CLOCK is a fascinating program demonstrating the use of mixed HIRESolution graphics and text. It simulates an analog clock, and if you read the REMarks, you will see how to adjust the time, and the speed of the clock. This program is a sure winner to impress your friends.

```
00100 REM This is a mixed hires graphics and text
00110 REM program for the MicroBee.
00120 REM 6/1/82
00130 REM
00140 REM To speed up clock, type 5 ... 9 (9 is
fastest)
00150 REM to slow down clock, type 5 ... 1 (1 is
slowest)
00160 REM to exit program, type q
00170 REM to move the minute hand on, type m
00180 REM this also resets the "seconds" timer
00190 REM to move the hour hand on, type h
00200 REM
00210 D=1 : REM D=2000 : rem large main timing
delay
00220 P1=355/113 : REM close approximation to PI
00230 Z=5 : REM small delay
00240 R6=80:R7=100 : REM set lengths of small
and large hands
00250 CLS
00260 HIRES
00270 REM print the numbers on the clock ...
00280 CURS 32,8:PRINT"*"
00290 CURS 32,2:PRINT"12"
00300 CURS 32,14:PRINT"6"
00310 CURS 49,8:PRINT"3"
00320 CURS 15,8:PRINT"9"
00330 CURS 40,3:PRINT"1"
00340 CURS 46,5:PRINT"2"
00350 CURS 46,11:PRINT"4"
00360 CURS 40,13:PRINT"5"
00370 CURS 24,13:PRINT"7"
00380 CURS 18,11:PRINT"8"
00390 CURS 18,5:PRINT"10"
00400 CURS 24,3:PRINT"11"
00410 CURS 0
00420 A0=0:B0=0: REM reset hours and minutes
00430 REM hours jumps to here
00440 GOSUB [B0] 1000
00450 REM minutes to here
```

```
00460 GOSUB [A0] 2000
00470 Y=0
00480 GOSUB 5000 : REM look at keyboard
00490 Y=Y+1:IF Y<D THEN 480
00500 FOR Y=0 TO 100*Z
00510 NEXT Y
00520 GOSUB [A0] 2000
00530 A0=A0+1
00540 REM Chime on the quarter hour ...
00550 IF A0=15 OR A0=30 OR A0=45 THEN
PLAY 12
00560 IF A0=<60 THEN 450
00570 GOSUB [B0] 1000
00580 A0=0:B0=B0+1
00590 REM Play Westminster Chimes
00600 PLAY 13:PLAY 9:PLAY 11:PLAY 4:PLAY
0:PLAY 4:PLAY 11
00610 PLAY 13:PLAY 9
00620 FOR Y=1 TO INT(B0):PLAY 0:PLAY 1:NEXT Y
00630 IF B0<12 THEN 430
00640 PLAY 1+INT(RND*24)
00650 GOTO 420
00660 GOTO 660
01000 REM This draws small hand at posn p0
01002 VAR (P0)
01010 T0=P1/6*(3-P0)-P1/150
01020 PLOT1
251+INT(14*COS(T0)),136+INT(14*SIN(T0)) TO
251+INT(R6*COS(T0)),136+INT(R6*.688*SIN(T0))
01050 RETURN
02000 REM This draws big hand at posn p0
02002 VAR (P0)
02010 T0=P1/30*(15-P0)+P1/150
02020 PLOT1
251+INT(14*COS(T0)),136+INT(14*SIN(T0)) TO
251+INT(R7*COS(T0)),136+INT(R7*.688*SIN(T0))
02050 RETURN
05000 REM look and keyboard and do delay
05010 A1$=KEY$
05020 IF A1$="" THEN RETURN : REM if no key
pressed
05030 IF A1$ >= "0" AND A1$ <= "9" THEN LET
Z=INT(VAL(A1$))
05040 IF A1$="q" THEN CLS:END:REM quit clock
program
05050 REM allow the use of the "h" key for setting
hours
05060 IF A1$="h" THEN GOSUB [B0] 1000:LET
B0=B0+1:GOSUB[B0] 1000:IF B0=12 THEN LET
B0=0
```

```

05070 REM and the "m" key for minutes
05080 IF A1$="m" THEN LET Y=0:GOSUB [A0]
2000:LET A0=A0+1:GOSUB[A0] 2000:IF A0=60
THEN LET A0=0
05090 RETURN

```

This program gives you a way of entering machine language programs into some spare memory high up in the MicroBee "memory map". This BASIC program was designed to enter a "glass typewriter" machine language program.

The bytes to be entered are as follows:
205,6,128,71,205,12,128,195,0,244,0

For different length programs (if you know Z80 machine code) you could alter the "TO" value in lines 100 and 140.

```

00090 REM *** this program writes machine code
programs***
00091 REM *** and stores them at F400 (62464)***
00092 REM *** to operate type USR (62464,0) ***

```

```

00093 REM
00100 FOR B=62464 TO 62474
00110 INPUT "type in a byte";A
00120 POKE B,A
00130 NEXT B
00140 FOR W=62464 TO 62474
00150 PRINT PEEK (W)
00160 NEXT W
00170 END

```

To run the machine language program AFTER you have used the BASIC program to enter the bytes, type (in the immediate mode),
PRINT USR(62464) «CR»

Then type any key and see it appear on the VDU without response from the BASIC. Press REPT to return to BASIC.

MATRIX is an example of the complex arithmetic handling powers of MicroWorld BASIC. It is also a good illustration of the use of arrays.

The best way to see how this program works is to consider the following example, and then play with the program yourself.

YOU: RUN «CR»

BEE: Enter command as follows ...

...->
YOU: a= «CR» Or any lower case a-j variable name

BEE: Number of rows ?

YOU: 2 «CR»

BEE: Number of columns ?

YOU: 2 «CR»

BEE: ?

YOU: 1 «CR» 2 «CR» Entering data into matrix
3 «CR» 4 «CR»

BEE: ->

YOU: b=a*a «CR» Matrix multiplication

BEE: ->

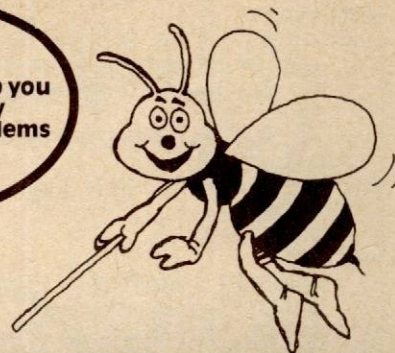
YOU: b «CR» Print matrix b (result of a*a)

```

00001 REM "Matrix arithmetic package for Z80
MicroWorld Basic"
00003 REM
00005 REM NUMBER OF POSSIBLE VARIABLES ..
00010 N=10
00015 REM MAXIMUM NUMBER OF ROWS OR
COLUMNS PER MATRIX ..
00020 S=10
00110 DIM A0(N,S,S), R0(S,S), R(N), C(N)
00120 REM SET ALL MATRICES TO 0*0

```

Let MicroBee help you solve those really tough Maths problems



- 1) print a matrix 'a'
- 2) entry 'a='
- 3) assignment between variables 'a=b'
- 4) matrix multiplication 'a=b*c'
- 5) scalar multiplication 'a=b.<real>'
- 6) matrix addition 'a=b+c'

-> a

Unassigned

-> a=

Number of rows ? 2

Number of columns ? 2

? 1 ? 2

? 3 ? 4

-> b=a*a

-> -

```

00130 FOR I=1 TO N
00140 R(I)=0 : C(I)=0
00150 NEXT I
00200 CLS
00210 PRINT "Enter command as follows ..."
00215 PRINT "1) print a matrix 'a' "
00220 PRINT "2) entry 'a=' "
00230 PRINT "3) assignment between variables 'a=b' "
00240 PRINT "4) matrix multiplication 'a=b*c' "
00245 PRINT "5) scalar multiplication 'a=b.<real>' "
00250 PRINT "6) matrix addition 'a=b+c' "
00260 PRINT
01000 REM GET INPUT STRING AND DO AS ASKED
01010 INPUT "->" C0$
01020 IF C0$="" THEN 1000
01025 IF C0$="quit" THEN END
01030 IF C0$="help" THEN 200
01070 P=1 : REM POSITION IN STRING SO FAR
01100 REM GET FIRST VARIABLE
01110 GOSUB 40400
01120 IF X=0 THEN 9000 : REM ERROR ***
01130 A=X
02000 IF P-1 = LEN(C0$) THEN 4000:REM PRINT A
MATRIX
02010 GOSUB 40000 : REM SKIP SPACES
02020 IF C0$(P,P) <> "=" THEN 9200
02050 IF P=LEN(C0$) THEN 3000 : REM GET A
MATRIX
02060 GOSUB 40000 : REM SKIP SPACES
02070 IF C0$(P,P)=' ' THEN 3000:HAD SPACES AFTER
= SO GET MAT
02100 P=P+1 : REM POINT TO NEXT VARIABLE
CHAR
02110 GOSUB 40400 : REM GET VARIABLE NUMBER
IN X
02120 IF X=0 THEN 9000 : REM ERROR ***
02130 B=X : REM FIRST ARGUMENT OF SOURCE
02135 IF R(B) = 0 THEN 9500 : REM *** UNASSIGNED

```

```

02140 IF P-1 = LEN(C0$) THEN 5000 : REM STRAIGHT
ASSIGNMENT
02180 REM IT IS AN OPERATION!!
02185 GOSUB 40000:REM SKIP SPACES
02190 IF C0$(;P,P)="+" THEN 6000 : MATRIX
ADDITION
02200 IF C0$(;P,P)="*" THEN 7000 : MATRIX
MULTIPLICATION
02210 IF C0$(;P,P)="." THEN 8000 : SCALAR
MULTIPLICATION
02220 GOTO 9600
03000 REM
03010 REM GET A MATRIX
03020 INPUT "Number of rows ? " R(A)
03030 IF R(A) < 1 OR R(A) > S THEN 3020
03040 REM GET NO. COLUMNS
03050 INPUT "Number of columns ? " C(A)
03060 IF C(A) < 1 OR C(A) > S THEN 3050
03070 REM NOW GET ARRAY
03080 FOR I=1 TO R(A)
03090 FOR J=1 TO C(A)
03100 INPUT A0(A,J,I); : PRINT ,
03110 NEXT J
03120 PRINT
03130 NEXT I
03140 GOTO 1000
04000 REM
04010 REM PRINT A MATRIX
04020 GOSUB [A] 40200
04030 GOTO 1000
05000 REM
05010 REM STRAIGHT ASSIGNMENT
05040 FOR I=1 TO R(B)
05050 FOR J=1 TO C(B)
05060 A0(A,J,I) = A0(B,J,I)
05070 NEXT J
05080 NEXT I
05090 R(A) = R(B)
05100 C(A) = C(B)
05110 GOTO 1000
06000 REM MATRIX ADDITION
06010 IF P=LEN(C0$) THEN 9300 : REM ERROR ***
MISSING ARG
06020 P=P+1
06030 GOSUB 40400 : REM GET THIRD ARG IN X
06040 IF X=0 THEN 9000
06050 REM NEED B AND X TO HAVE SAME
DIMENSIONS
06055 IF R(X)=0 THEN 9500
06060 IF R(X) <> R(B) OR C(X) <> C(B) THEN 9400
06070 FOR I=1 TO R(B)
06080 FOR J=1 TO C(B)
06090 A0(A,J,I) = A0(B,J,I) + A0(X,J,I)7+1-1
06100 NEXT J
06110 NEXT I
06120 R(A) = R(B)
06130 C(A) = C(B)
06140 GOTO 1000
07000 REM MATRIX MULTIPLICATION
07010 REM CHECK FOR THIRD ARGUMENT
07020 IF P=LEN(C0$) THEN 9300 : REM ERROR ***
MISSING ARG
07030 P=P+1
07040 GOSUB 40400 : REM GET THIRD ARG IN X
07050 IF X=0 THEN 9000 : REM *** ERROR BAD
VARIABLE NAME
07060 REM CHECK THAT ROWS/ COLUMNS WILL
WORK
07065 IF R(X)=0 THEN 9500 :REM *** UNASSIGNED
ERROR
07070 IF C(B) <> R(X) THEN 9700
07090 FOR I=1 TO R(B)
07100 FOR J=1 TO C(X)
07105 S0=0
07110 FOR K=1 TO R(X)
07120 S0=S0+A0(B,K,I)*A0(X,J,K)
07130 NEXT K
07140 R0(J,I)=S0
07150 NEXT J
07160 NEXT I
07200 REM ASSIGN RESULT TO DESTINATION A
07210 FOR I=1 TO R(B)
07220 FOR J=1 TO C(X)
07230 A0(A,J,I)=R0(J,I)+1-1
07240 NEXT J
07250 NEXT I
07260 REM ASSIGN CORRECT DIMENSIONS TO A
07270 R(A) = R(B)
07280 C(A) = C(X)
07400 GOTO 1000
08000 REM SCALAR MULTIPLICATION
08010 IF P=LEN(C0$) THEN 9300
08015 P=P+1
08020 N0$ = C0$(;P)
08030 N1 = VAL(N0$)
08050 FOR I = 1 TO R(B)
08060 FOR J = 1 TO C(B)
08070 A0(A,J,I) = N1*A0(B,J,I)
08080 NEXT J
08090 NEXT I
08100 R(A) = R(B)
08110 C(A) = C(B)
08120 GOTO 1000
08999 STOP
09000 PRINT C0$(;P,P); " is not a variable"
09010 GOTO 9900
09200 PRINT "Missing equals ..."
09210 GOTO 9900
09300 PRINT "Missing argument"
09310 GOTO 9900
09400 PRINT "Cannot add matrices of different
dimensions"
09410 GOTO 1000
09500 PRINT "Unassigned variable"
09510 P=P-1
09520 GOTO 9900
09600 PRINT "Not an operation"
09610 GOTO 9900
09700 PRINT "Cannot compose due to mismatched
codomain and domain"
09710 GOTO 1000
09900 PRINT C0$
09905 IF P=1 THEN GOTO 9950
09910 FOR I=1 TO P-1
09920 PRINT " ";
09930 NEXT I
09950 PRINT "^"
09960 GOTO 1000
40000 REM SUBROUTINE TO SKIP SPACES
40010 IF P = LEN(C0$) THEN 40090
40020 REM O.K. TO SEE IF THERE ARE SOME
SPACES
40030 IF C0$(;P,P) <> " " THEN 40090 : REM NOT A
SPACE
40040 P=P+1
40050 GOTO 40030
40090 RETURN
40095 REM
40200 REM
40210 REM PRINT MATRIX [X]
40220 VAR(X)
40230 IF R(X)=0 THEN PRINT "Unassigned" : RETURN
40240 REM
40250 FOR I=1 TO R(X)
40260 FOR J=1 TO C(X)
40270 PRINT A0(X,J,I),
40280 NEXT J
40290 PRINT
40300 NEXT I
40310 RETURN
40400 REM
40410 REM GET A VARIABLE NAME IN X FROM C0$,
40420 REM RETURNING 0 IF ERROR
40430 REM USES C0$, P, X
40440 GOSUB 40000 : REM SKIP SPACES

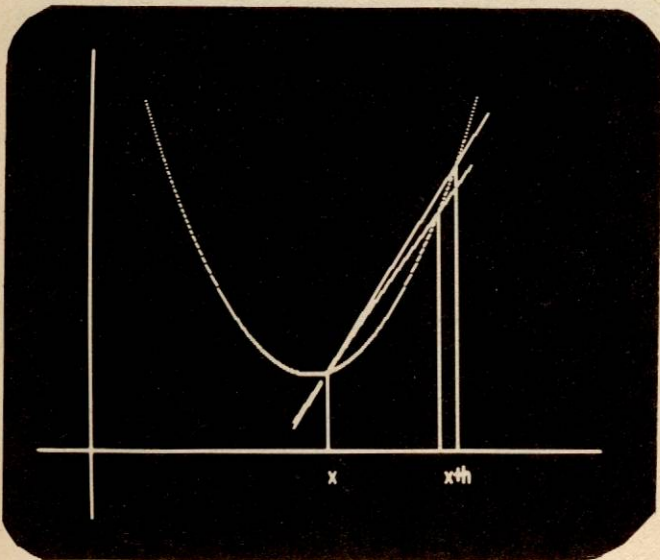
```

```

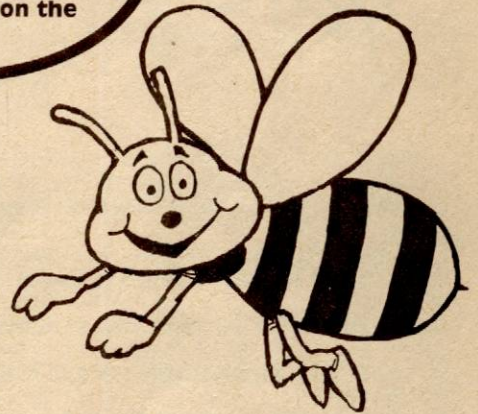
40450 IF C0$(;P,P) <"a" OR C0$(;P,P) >"z" THEN LET
X=0 : GOTO 40590
40460 X=ASC(C0$(;P,P)) - ASC("a") + 1
40470 IF X>N THEN LET X=0 : GOTO 40590
40480 P=P+1
40590 RETURN

```

DIFF is a program using HIRES graphics and the PLOT command to show the effect of a secant approaching a tangent to a parabola.



Illustrating Maths problems is no problem with MicroBee. You can combine words, numbers and graphics. The best educational programmes can be written on the MicroBee.



```

00100 REM This program illustrates the first principles
00110 REM of differentiation
00120 REM
00140 SD 14 : REM set up for high precision
00150 FN0 = 75+(-250)*(-250)/150
00160 FN1 = 75+(-250)*(-250)/150
00170 HIRES
00180 V = 4096*15 + 64*14
00190 L0=0 : REM set up for no erase after first draw
00200 HIRES
00210 REM draw axes
00220 PLOT 50,0 TO 50,255
00230 PLOT 0,35 TO 511,35
00240 REM draw parabola
00250 FOR X= 100 TO 400
00260 SET X, FN0(X)
00270 NEXT X
00280 P1 = 264 : Q1 = 380
00290 PLOT INT(P1),INT(FN1(P1)) TO INT(P1),35
00300 POKE V+INT(P1/8),ASC("x")
70310 POKE V+1024+INT(P1/8),0
00320 REM *** let h -> 0
00330 FOR Q1 = Q1 TO P1+1 STEP -16
00340 PLOT INT(Q1),INT(FN1(Q1)) TO INT(Q1),35
00350 GOSUB [P1,Q1] 490
00360 POKE V+INT(Q1/8)-1,ASC("x")
00370 POKE V+INT(Q1/8),ASC("+")
00380 POKE V+INT(Q1/8)+1,ASC("h")
00390 POKE V+INT(Q1/8)+2,ASC(" ")
00400 POKE V+INT(Q1/8)+3,ASC(" ")

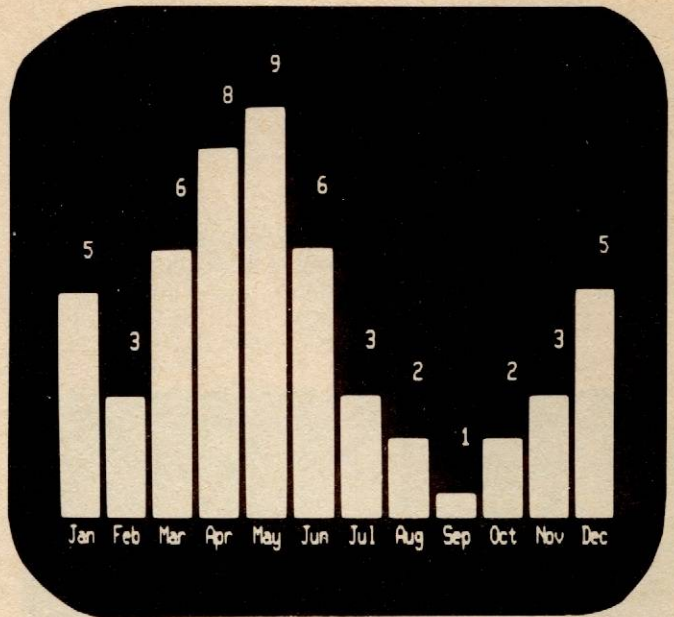
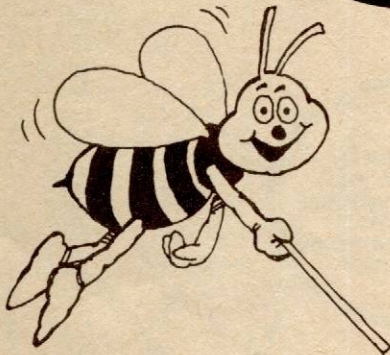
```

```

00410 FOR Z V+1023+INT(Q1/8) TO
V+1027+INT(Q1/8)
00420 POKE Z,0
00430 NEXT Z
00440 REM *** ERASE LAST SECANT OR LINE
00441 IF L0 THEN 460
00450 GOSUB [L0,L1] 560 : PLOT INT(L1),INT(FN1(L1))
TO INT(L1),35
00460 L0=P1 : L1 = Q1
00470 NEXT Q1
00480 GOTO 480
00490 REM DRAWS A SECANT THRU P1 OR Q1
00500 VAR(X1,X2)
00510 E0 = 30
00520 IF ABS(X1-X2)<30 THEN LET E0 = (70-ABS(X1-
X2))/2
00530 M0 = (FN1(X2)-FN1(X1))/(X2-X1)
00540 PLOT INT(X1-E0),INT(FN1(X1)-E0*M0) TO
INT(X2+E0),INT(FN1(X2)+E0*M0)
00550 RETURN
00560 REM CLEAR THE SECANT THRU P1 OR Q1
00570 VAR(X1,X2)
00580 E0 = 30
00590 IF ABS(X1-X2)<30 THEN LET E0 = (70-ABS(X1-
X2))/2
00600 M0 = (FN1(X2)-FN1(X1))/(X2-X1)
00610 PLOT INT(X1-E0),INT(FN1(X1)-E0*M0) TO
INT(X2+E0),INT(FN1(X2)+E0*M0)
00620 RETURN
00630 END

```

Run these great programmes and impress your friends! I'll make you an expert programmer in no time.



BARS is a typical example of the use of graphics to present information more effectively than just numbers alone.

```

00100 REM This is a LORES graphics demonstration
which
00110 REM accepts input data in numerical form, and
gives
00120 REM a representation of this in BAR CHART form
00130 REM
00140 DIM M0(12) : REM Holds the data for each month
00150 CLS
00160 PRINT "Enter your input data for each of the
twelve"
00170 PRINT "months just using integer values"
00175 PRINT "Use numbers less than 10000 please"
00180 PRINT
00190 REM Get the data for each month in turn from
operator
00200 REM and find the maximum at the same time
00210 M1=0 : REM set maximum to zero for start
00220 A=0 : REM index to maximum entry
00230 FOR I=1 TO 12
00240 PRINT "Data for month ";I;
00250 INPUT M0(I)
00260 IF M0(I)>M1 THEN LET M1=M0(I):A=I
00270 NEXT I
00280 REM now scale the graph so that the largest
entry,indexed
00290 REM to by A takes up the whole screen
00300 S0=42/M1 : REM use 42
00310 REM now print out the columns
00320 CLS:LORES
00330 FOR I=1 TO 12
00340 REM fill in one bar
00350 IF INT(S0*M0(I))=0 THEN CURS 5*I-2,15 : GOTO
410
00360 FOR J=5 TO 12
00370 PLOT 10*(I-1)+J,3 TO 10*(I-1)+J,INT(S0*M0(I))
00380 NEXT J
00390 REM print the value above the bar ...
00400 CURS 5*I-2,15-INT(S0*M0(I)/42*14+1)
00410 C=INT(M0(I))
00420 PRINT [I4 C];
00430 NEXT I
00440 CURS 1,16
00450 PRINT " Jan Feb Mar Apr May Jun Jul Aug
Sep Oct Nov Dec";
00455 CURS 0
00460 GOTO 460

```

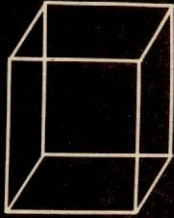
More great programmes like this one are available right now for your MicroBee. Your MicroBee comes with an extended BASIC manual, full of a huge variety of great programming ideas.



```

06000 REM This subroutine draws a square of lengths l1,l2
06005 REM with the bottom corner at a1,b1
06010 VAR(A1,B1,L1,L2)
06020 REM Draw left side, then top, then right, then bottom
06030 GOSUB [ A1,B1,A1,B1+L2 ] 4000
06040 GOSUB [ A1,B1+L2,A1+L1,B1+L2 ] 4000
06050 GOSUB [ A1+L1,B1+L2,A1+L1,B1 ] 4000
06060 GOSUB [ A1+L1,B1,A1,B1 ] 4000
06999 RETURN
65000 END

```



CUBE, a simple HIRESolution graphics program, shows the use of the SET routines and the passing of parameters to subroutines using GOSUB [...] and VAR commands.

Try it first, and then perhaps you might like to rewrite it using the PLOT command to see how much faster it can be made to run.

```

00005 CLS : HIRES
00006 L1=110 : L2=80
00007 P1=350 : P2=30
00008 Q1=P1+30 : Q2 = 60
00010 GOSUB [ P1,P2,L1,L2 ] 6000
00020 GOSUB [ Q1,Q2,L1,L2 ] 6000
00040 GOSUB [ P1,P2+L2, Q1,Q2+L2 ] 2000
00050 GOSUB [ P1+L1,P2+L2, Q1+L1,Q2+L2 ] 2000
00060 GOSUB [ P1+L1,P2, Q1+L1,Q2 ] 2000
00070 GOSUB [ P1,P2, Q1,Q2 ] 2000
01000 LIST 6000,
01999 GOTO 1999
02000 REM This program draws a line between 2 points
02010 REM with co-ordinates X1,Y1 and X2,Y2
02015 VAR(X1,Y1,X2,Y2)
02020 IF ABS(X2-X1)>ABS(Y2-Y1) THEN GOTO 2200
02030 REM draw y's by ones
02040 IF Y2-Y1 =0 THEN SET INT(X1),INT(Y1) :
RETURN
02050 S0 = SGN(Y2-Y1)
02060 S1 = (X2-X1)/ABS(Y2-Y1) : X3 = X1
02070 FOR Y3 = Y1 TO Y2 STEP S0
02080 X3 = X3 + S1
02090 SET INT(X3),INT(Y3)
02100 NEXT Y3
02110 RETURN
02200 REM draws by X's by ones
02210 S0 = SGN(X2-X1)
02220 S1 = (Y2-Y1)/ABS(X2-X1) : Y3 = Y1
02230 FOR X3 = X1 TO X2 STEP S0
02240 Y3 = Y3 + S1
02250 SET INT(X3),INT(Y3)
02260 NEXT X3
02270 RETURN
03000 REM This program clears a line between 2 points
03010 REM with co-ordinates X1,Y1 and X2,Y2
03015 VAR(X1,Y1,X2,Y2)
03020 IF ABS(X2-X1)>ABS(Y2-Y1) THEN GOTO 3200
03030 REM clear Y's by ones
03040 IF Y2-Y1 =0 THEN RESET INT(X1),INT(Y1) :
RETURN
03050 S0 = SGN(Y2-Y1)
03060 S1 = (X2-X1)/ABS(Y2-Y1) : X3 = X1
03070 FOR Y3 = Y1 TO Y2 STEP S0
03080 X3 = X3 + S1
03090 IF INT(Y3)<> FNO(INT(X3)) THEN RESET
INT(X3),INT(Y3)
03100 NEXT Y3

```

```

03110 RETURN
03200 REM Clears X's by ones
03210 S0 = SGN(X2-X1)
03220 S1 = (Y2-Y1)/ABS(X2-X1) : Y3 = Y1
03230 FOR X3 = X1 TO X2 STEP S0
03240 Y3 = Y3 + S1
03250 IF INT(Y3)<> FNO(INT(X3)) THEN RESET
INT(X3),INT(Y3)
03260 NEXT X3
03270 RETURN
04000 REM This routine draws a line, but only horizontal
and vertical
04010 VAR(X1,Y1,X2,Y2)
04020 IF X1=X2 THEN GOTO 4090
04025 Z = INT(Y1)
04030 FOR W = INT(X1) TO INT(X2) STEP INT(SGN(X2-
X1))
04040 SET W,Z
04050 NEXT W
04060 RETURN
04090 W = INT(X1)
04100 FOR Z = INT(Y1) TO INT(Y2) STEP INT(SGN(Y2-
Y1))
04110 SET W,Z
04120 NEXT Z
04130 RETURN
06000 REM This subroutine draws a square of lengths
l1,l2
06005 REM with the bottom corner at a1,b1
06010 VAR(A1,B1,L1,L2)
06020 REM Draw left side, then top, then right, then
bottom
06030 GOSUB [ A1,B1,A1,B1+L2 ] 4000
06040 GOSUB [ A1,B1+L2,A1+L1,B1+L2 ] 4000
06050 GOSUB [ A1+L1,B1+L2,A1+L1,B1 ] 4000
06060 GOSUB [ A1+L1,B1,A1,B1 ] 4000
06999 RETURN
65000 END

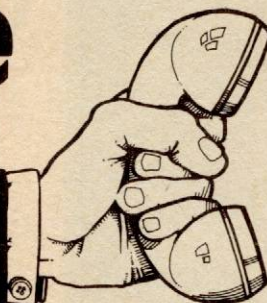
```

**Turn over the
page to find
out how to get
your MicroBee.**



microbee

How to order your MicroBee



Save time and order your MicroBee direct on our Hotline number. Just give your Bankcard Number and name. We check stocks and get the goods on the way.

(02)487 3798

MICROBEE HOTLINE



Applied Technology Pty Ltd

Send your order to:

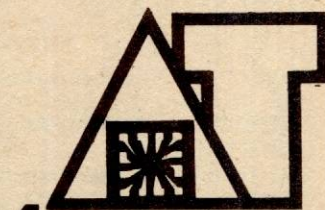
PO Box 311, Hornsby 2077

Showroom/Office at 1A Pattison Ave. Waitara NSW

Hours: 9-5 Monday to Saturday

Phone: (02) 487 2711 Telex: APPTec AA72767

**EASY
FIVE
STEP
MAIL
ORDER
FORM**



1 PLEASE DETAIL YOUR REQUIREMENTS BELOW

CAT. NO.	DESCRIPTION OF GOODS	QTY.	PRICE EA.	AMOUNT EXTENDED
	MicroBee kit including 16K RAM, sockets and full instructions.		\$399.00	
	12 inch Video Monitor		\$129.50	
	Cassette Recorder		\$35.00	
	Blank C10 computer cassettes		\$1.00	
	16K RAM expansion kit (includes powerdown RAM ICs, sockets and fitting instructions.		\$120.00	

2 POSTAGE CALCULATION

There is a standard charge of \$6.00 for postage and packing on the MicroBee. Heavy or easily damaged items such as monitors will be sent freight on via road freight. If you live near a pickup station, we can send items via overnight freightbag for \$7.50 - anywhere in Australia.

Handling/packing **\$6.00**

Postage calculation

Sub-total

Insurance add
\$1 per \$100

AMOUNT
ENCLOSED

Customers please note: Stocks of the MicroBee are expected to be available mid February. Please phone to check stocks before ordering.

OFFICE USE ONLY
TPN NUMBER

M

--	--	--	--	--	--

3 METHOD OF PAYMENT

Money order

Crossed cheque

Bankcard

Please debit my Bankcard:

Bankcard number

Cardholder's

signature

Name

Expires end /



4 DELIVERY ADDRESS

Name

Address

Postcode

5 CHECK STEPS 1, 2, 3 & 4