BENCHTEST
BUSINESS SYSTEM

MONROE 8820

Peter Rodwell tests one of the latest upmarket business micros from the States.

Monroe is an American company which has for some years now been known for its desk-top calculators; I'm sure you know the sort — those large things the size of a portable typewriter which professional number-handlers who don't actually require computing power still buy because, unlike pocket calculators, they're built to take a real pounding.

However, people must move with the times and, even if you don't need programmability or data storage facilities, you're supposed to have a desk-top computer these days; calculators are definitely not what the up-and-coming accountant wants seen on his desk.

Perhaps sensing this trend, Monroe has taken the logical step of going into the personal computer market. It still makes those large calculators (presumably there are still people around who just won't let go of an old technology), although they're no longer on sale in this country.

Two basic models make up the Monroe range: the OC 88XX, a business (or 'occupational', as the Monroe brochure puts it) micro, and the EC 8800, a rather nice-looking educational model with colour graphics, not yet available over here. Two business models are made, the OC 8810, which has a single minifloppy disk and the OC 8820, which has twin disks. This Benchtest is of the OC 8820.

Hardware

The 8820 is an all-in-one computer with the electronics, keyboard, screen and disks housed in a well-made and quite heavy two-tone beige ABS casing measuring 53cm deep by 48cm wide by 29cm high. The whole unit is solidly constructed and has a feeling of real quality about it — you're left in no doubt that this is a serious piece of office equipment, built to take the pounding of day in, day out commercial life.

The keyboard has 93 keys, arranged in six groups. The main group is, of course, a full qwerty keyboard of typewriter pitch and with a nice solid feel, eminently suitable for touch-typing. To its right is a numeric pad with arithmetical operators, and beyond this is a column of cursor control keys topped with a clear/home key. A solitary red key labelled 'STOP' sits in the upper left corner; to its right is a row of two blocks of four programmable function keys such as editing and caps lock controls. All the keys auto-repeat when depressed for longer than a second or so.

The 22.5cm screen at first looks rather small but, in fact, when you're seated at the machine and using it, it's just fine. The screen provides the first surprise of the system; it has orange lettering on a grey background. I found this fairly weird at first, being used to green on black, but I soon got used to it and found it quite pleasant to work with. The 80 x 24 display is rock-steady and clear, the lettering having true descenders, although it was slightly blurred around the edges of the screen. The full ASCII character set can be displayed plus a range of 'chunky' graphics; the latter are displayed not directly from the keyboard but by sending the appropriate codes to the screen from a program. Characters can also be displayed in double height, double width, inverse video or dimmed. A brightness control is mounted just below the screen in the form of a thumbwheel protruding downwards from the bezel.

The twin minifloppy disks are mounted horizontally, one above the other, to the right of the screen. The disks, made by Micropolis, are single-sided, double density soft-sectored units holding 320k each. They are almost silent in operation, making only a slight 'clunk' when they turn off and on.

Inside, a 3MHz Z80 sits at the centre of things, with 128 kbytes of RAM to play with plus 4k of video RAM and a 2k bootstrap PROM.

Along the bottom edge at the back is a row of connectors with which the Monroe communicates with the outside world. Three of these are RS232 ports, labelled 'Printer', 'Communication I/O' and 'Auxiliary service I/O'; the latter is designed for service engineers — a troubleshooting ROM pack plugs into it to aid with system diagnostics. Two other, larger connectors are labelled 'Expansion bus' and 'External disk' — more on these later. The reset switch is mounted at the back, too, but the on/off switch lives on the left side, low down towards the back.

A final note on the hardware design: this is one of the quietest machines I have come across (among those which have disks, of course). Not only are the disk drives virtually silent, but so is the fairly powerful fan.

Operating system

The mention a few paragraphs earlier that the system has 128k of RAM will have already alerted you to the fact that the Monroe is somewhat different to the majority of 8-bit machines now on the market. This difference gets larger the deeper one probes into the machine.

Although CP/M is available as an option, the standard operating system which comes with the machine is Monroe's own operating system, which I'll be abbreviating here to MOS, and which bears no relation to CP/M whatsoever.

MOS is a single-user multi-tasking operating system which takes the form of 43 kbytes of 'core' software (ie, it sits in memory all the time) plus a number of utility programs which reside on disk and are called in as needed.

On power-up, or on hitting reset, the system first performs a memory check (which takes seven seconds, during which a message on the screen tells you what's happening) and then, to the

accompaniment of a 'waiting' message on the screen, searches both disk drives for a disk containing MOS. It will boot from either drive but drive 0, the lower drive, is the preferred one. The operating system, all 43k of it, takes 10 seconds to load, during which the screen message changes to 'Loading'; once it's running, a green LED next to the disk drives lights up to say that MOS is active.

CP/M contains certain intrinsic commands such as 'DIR', the directory command, which are executed directly by the operating system without the need to load and run a separate program resident on disk. MOS has no such intrinsic commands; it considers anything typed in at the keyboard to be the name of a program on disk and goes to look for it on the currently-active drive, giving a terse error message if it can't find it.

To describe MOS in full detail would require a separate review of considerable length. The following description is, then, more a brief run-through of the system's facilities than an in-depth study.

Each utility program is kept on disk as a task file (the equivalent to a CP/M 'com' file — ie, a directly-executable program).

Allocate: allocates space on a disk as a direct-access file, either binary or ASCII.

Bootgen: used when creating a new disk; it tells the bootstrap loader where to find the operating system on the disk.

Close: takes a specified device off-line; has to be used before you remove a disk from the system.

Command: similar to, but more versatile than, CP/M's 'Submit'. This allows you to execute a number of commands by creating a text file of the commands and handing this to Command for execution. We'll discuss this in depth later.

Copya: this transfers ASCII between two devices; not only can this be used for copying, but it can also function between, say, the keyboard and a disk, so that the text can be typed straight onto disk.

Copyi: this performs an image copy between devices and/or files. It would be used, for example, for making back-up copies of disks.

Copylib: a fast transfer copy program; it can also be used to delete programs and operates in either an interactive or automatic mode.

Copyt: used for copying task files; task files can be either absolute or relocatable and this utility will copy either.

Creindex: allocates and creates an ISAM (indexed sequential access method) file.

Delete files: does exactly that.

Diskcheck: It's necessary to close files before changing a disk or switching off. If you forget to do this, Disckcheck gives you the chance to repair the damage by closing any files left open on the disk.

Diskinit: initialises a disk after it has been formatted by giving it a volume name and bit map.

Format: formats a blank disk.

Lib: display a disk directory. Typing L displays just the file names (which can be up to 12 characters long) and the type of file: Tsk for task, Asc for ASCII, Bac for Basic, etc. Typing L, F displays this information plus a good deal more, such as file sizes and the date and time they were created and last used.

Open: brings a device on-line; if you're using drive 0 and you want to see what's on the disk in drive 1, you have to close drive 0 and open drive 1 first before typing L for Lib.

Option: each file has certain attributes, which can be changed with this utility. Thus a file can remain in memory after execution or be deleted from it; and a task can be aborted by another task or not.

Priority: as MOS is a multi-tasking operating system, tasks can be given priorities to determine their order of operation. That's what this does.

Rename: renames a file.

Space: tells you how much space is left on a disk, in *sectors*, which isn't as immediately useful as a kbytes read-out would be.

Set: enables the creation of an auto-start disk, ie, one which would immediately start an application program on boot-up without the user having to work through a sequence of operating system commands.

Sort: sorts out the contents of a file, including ASCII and numeric data files.

Time: sets the internal clock/

MONROE 8820

calendar. Unfortunately this is a software clock/calendar, so re-booting the operating system, such as is necessary when changing a disk, sets it to zero again. A hardware clock/calendar would have been more practical.

Vol: sets or changes the names of the system volume (ie, disk).

The above are all utilities associated with the system itself. An additional set of utilities is provided under the general classification of task handling utilities as follows:

Cancel: this cancels a task which is either current or in memory but dormant.

Continue: continues a task which has been paused.

Devices: displays information, including current status, of all the devices in the system.

Load: this loads one or more tasks into memory to allow multitasking to take place. The tasks are loaded in sequence and activated using the Start utility (see below).

Pause: this simply pauses a specified task.

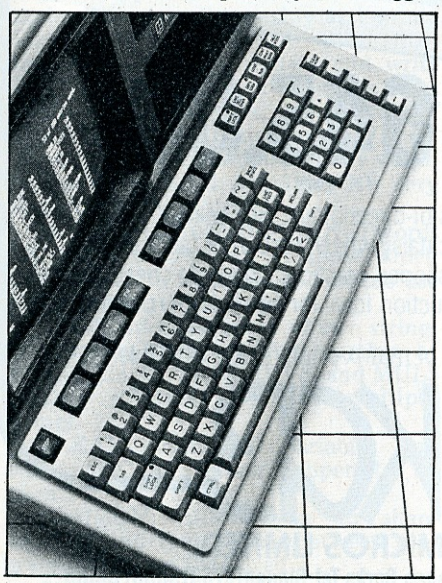
Run: this combines the Load and Start utilities; tasks are brought into memory and immediately executed.

Slice: during multitasking, work is scheduled according to each task's priority or according to time slices within that priority. If two tasks have an equal priority, the first in the queue will take all the processor's time until it's finished. Slice allows you to allow them to execute simultaneously by allocating time slices (in milliseconds).

Start: as mentioned earlier, this initiates tasks which have previously been Loaded into memory.

Task: lists the status of each task currently in memory, showing its priority, status, etc.

At first sight, MOS appears both complicated and unfriendly, particulary if, like me, you have become accustomed to CP/M and its many and varied idiosyncrasies. The profusion of copying utilities, for example, seems very confusing at first; and there's no easy way to find out what's on one disk if you're logged into the other — you have to type 'Close fpy0: open fpy1:' and then Lib, for example, if you're logged onto the lower drive and want to see the directory of the disk in the other drive. Compare this to the equivalent CP/M command, DIR B:.



Keyboard feels nice and solid

But a closer look at the utilities not only reveals that things are not as bad as they seem but shows the philosophy behind the Monroe System.

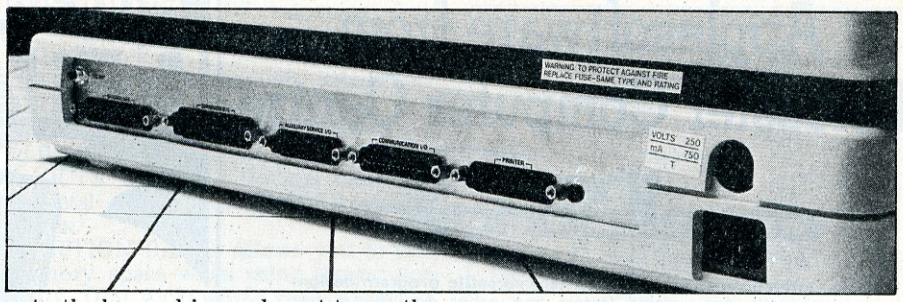
A microcomputer system will be used by two types of users — programmers and end users. In our Benchtests, we look at systems from the end user's point of view, mostly, although the programmer's angle obviously cannot be neglected. Looking at this machine as an end user, it appears awkward, unfriendly, overly complex and unnecessarily nitpicking in its syntax. In short, most end users, particularly the 'naive' (in computing terms) business user at whom the machine is aimed, would be frightened silly by it.

But it not intended that the end user should have to grapple with the utilities directly. The facilities exist for the programmer, using the very powerful utilities, to insulate the user from the operating system completely — the user would buy his Monroe with the software already arranged so that he would never actually have to come face to face with MOS; the auto start, run, load and other utilities can be combined to allow the user to carry out any operation necessary, including disk formatting, housekeeping and full multitasking without needing to learn anything of MOS at all. Although it is possible to insulate the user to some extent from CP/M, software houses rarely seem to bother and, in any case, CP/M simply doesn't allow the same degree of insulation.

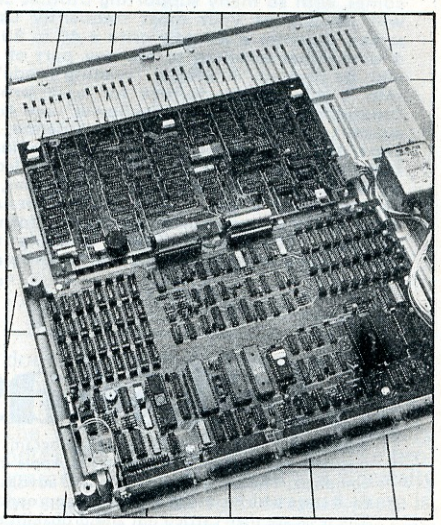
The above answers the question that immediately formed in my mind when I first looked at MOS: why did Monroe go to the trouble and expense of writing its own operating system when there are several off-the-shelf systems already around, such as MP/M-II and Unix? The answer, at least in part, must be that no existing operating system for micros allows the programmer to provide such a high degree of insulation. I feel that Monroe made the right decision in going the lone route. It has decided that it wants to sell microcomputers to businessmen and it has realised that these customers have neither the time not the inclination to learn the vocabulary, concepts and procedures of computing: to these users, the desk-top computer is another piece of office machinery and it needs to be as easy to use as a photocopier. Provided the programmer realises this and takes advantage of the machine's many powerful facilities, it should be easier to achieve this with the Monroe than with most other micros.

From the programmer's point of view, I feel the machine will prove an exciting and challenging beast. Programmers weaned on — or at least thoroughly accustomed to — CP/M will find it strange indeed, but those coming fresh from the mini/mainframe world will find it easy to live with. Which brings us to Monroe's other reason for developing its own operating system: although the official line is to deny it, there are strong and well-informed rumours that the company is developing a minicomputer; it seems reasonable to suggest, then, that MOS will have a good degree of compatibility with the mini's operating system and I'd go so far as to suggest that they were developed by the same team — MOS just has a mini feeling to it.

One minor gripe; Monroe has provided eight programmable function keys (actually 16 as they can be shifted), yet MOS makes absolutely no use of these at all. I understand a future release will, hopefully in the style of Hewlett-Packard's softkeys.



Rear view



Two neat boards house the major electronics.

Monroe Basic

Although Pascal is mentioned as an option in the literature, Monroe Basic is the system's main language. As 'MBasic' has already been used by somebody else, Monroe calls its Basic 'MEBasic' and it's certainly a very comprehensive and powerful implementation of the language.

Basic is called up by typing 'Basic' followed by the amount of memory you want to allocate to it; typing 'Basic,, 34000' gives you Basic (which takes up 34k) plus another 34k to play with. If you don't allocate any memory, the system gives you the default value of just 5 kbytes.

I found MEBasic one of the easiest and most pleasant to use of the various Basics knocking around these days. During programming and in command mode it accepts input in either upper

MONROE 8820

programming, these are converted and stored in upper case, so that 'a' and 'A', if used as variable names, reference the same variable and both appear as 'A' when the program is listed.

MEBasic checks your syntax dynamically as soon as you hit return at the end of the line. If you make a mistake, it consults an 8 kbyte disk file of error messages and tells you what's wrong; if the file isn't present on the current disk it gives you an error code instead. If you've made an error, the machine also bleeps at you and automatically enters its editing mode, with the cursor positioned at the end of the offending line – you can't continue until you've corrected the mistake, and editing keys such as insert and delete are provided to make this very easy. Thus, once you've typed in your program, you *know* that it's at least free of syntax errors before you RUN it.

On entering MEBasic, you are allowed variable names of only one letter plus (optionally) one digit, ie, A-Z and A0-Z9. Typing EXTEND (or including it in a program) allows the use of variable names of up to 32 characters, all of which are significant. I would have preferred the EXTEND mode to be the default mode, thus encouraging more readable programming; there's also a NO EXTEND command which cancels the EXTEND mode. You need to exercise a little caution when using EXTEND, though: in the NO EXTEND mode the system, is indifferent to spaces, so that INPUTA: PRINTA will be accepted. Using EXTEND, though, makes spaces critical as INPUTA would be interpreted as a variable name, and, in this context, cause an error message.

As can be seen from the list of Basic reserved words, MEBasic contains as its core the 'standard' facilities found in most Basics and takes a standard syntax for these, thus making it easy to use for programmers coming from, say, Microsoft Basic. In addition, there's a vast range of facilities to delight the serious programmer and to make the writing of business applications packages a cinch.

Of greatest interest among these are the ISAM file-handling capabilities. Regrettably, time did not allow me to explore these as thoroughly as I would have liked, but, basically, they allow you to open, read, update and write ISAM files which you create with a special Basic program, the listing of which is in the MEBasic manual. I know of no other microcomputer Basic which incorporates ISAM handling; it's very powerful and very useful.

Scanning through the reserved words list, here's a rundown of those which are peculiar to MEBasic and whose use isn't immediately obvious.

ADD$: adds the value of two strings of numbers to a specified number of decimal places. SUB$, DIV$, and MUL$ similarly subtract, divide and multiply strings.

CLEAR: clears all variables and closes any files which happen to be open.

COMMON: allows you to share variables and their values between programs which are CHAINed.

COMP%: compares two numeric strings, A$ and B$, returning -1 if A$ < B$, 0 if they're equal and 1 if A$ > B$.

CVT: converts numeric values into ASCII for an I/O file.

FLSH: causes the string following it to flash when printed on the screen.

OCT$: converts a decimal value into an octal one. Similarly, HEX$ converts to hex.

OPTION BASE: Allows you to save memory space when using arrays; normally, DIM A (5) would give an array of *six* elements A(0) to A(5); OPTION BASE 1 would eliminate the default A(0) element to give an array of five elements.

OPTION EUROPE: Not an EEC get-out but alters the respresentation of numbers from the British and American format of, say, 10,000.55 to the European format of 10.000,55. Nice to see Americans remembering these little touches.

PDL: returns the x or y coordinates of a joystick. As joysticks aren't provided with the OC 8820, I assume this is for use with the educational micro.

POSIT: Positions the file pointer to the specified number of bytes from the start of the file.

SET TIME: like the Time utility, this lets you set the internal calendar/clock.

SLEEP: Wins my 'Basic Reserved Word of the Year' award for 1982. It does just what it says by suspending the current program for a specified time (in seconds).

MEBasic has been designed not only for the OC 8820 but also for the educational computer with its high resolution colour graphics. Thus, in addition to the listed functions there is a whole range more dealing with colour graphics; as these obviously don't work on the OC 8820, I haven't included them in the list of reserved words.

As can be seen from this very quick look at MEBasic, it's a powerful language, thoughtfully designed and implemented and easy and friendly to use. Programmers will find it an efficient Basic as it not only encourages good documentation through the use of EXTENDed variable names but has many powerful built-in features, too, including the ability to CALL the operating system's supervisor calls directly. With the system came a large number of various Basic programs, some of them utilities and others just games and demonstrations; all are thoroughly documented and most are designed to show you how to use various features of both language and machine.

Documentation

The system came with three bulky manuals, one on MOS, one on the utility programs and a third, *very* thick one, on MEBasic. They were all described as programmer's reference manuals and a warning inside pointed out that they had been designed for experienced programmers, not as tutorials. Fair enough, they certainly made no attempt to teach basic concepts but they all explained every feature of operating system, utilities and Basic very thoroughly. I'd go so far as to say that they're among the best examples of documentation I've seen, being clearly written, thoroughly and accurately indexed and including heavily-commented examples of each feature, leaving the reader in no doubt whatsoever of how each facility should be used.

Basic reserved words

ABS	FNEND	PDL
ADD$	FOR. .NEXT. .STEP	PEEK
ASCII		PEEK2
ATN	GET	PI
AUTO	GOSUB	POKE
	GOTO	POSIT
BYE		PREPARE
	HEX$	PRINT
CALL		PRINT USING
CHAIN	IF. .THEN. .ELSE	PUT
CHR$	INP	
CLEAR	INPUT	RANDOMIZE
CLOSE	INPUT LINE	READ
COMMON	INSTR	REM
COMP%	INT	RENUMBER
CONTINUE	INTEGER	RESTORE
COS	ISAM OPEN	RESUME
CUR	ISAM READ	RETURN
CVT	ISAM UPDATE	RIGHT$
CVT$F	ISAM WRITE	RND
CVTF$		RUN
	KILL	
DATA		SAVE
DBLE	LET	SCR
DEF	LIST	SET TIME
DIM	LOAD	SGN
DIV$	LOG	SIN
DOUBLE	LOG10	SINGLE
		SLEEP
EDIT	MERGE	SOUND
END	MID$	SPACE$
ERASE	MOD	SQR
ERRCODE	MUL$	STDY
EXP		STOP
EXTEND	NAME	STRING$
	NEW	SUB$
FGCIRCLE	NO EXTEND	SVC
FGCOPY	NOTRACE	SWAP
FGCTL	NRML	SYS( )
FGDRAW	NUM$	
FGERASE		TAN
FGFILL	OCT$	TIME$
FGLINE	ON ERROR GOTO	TRACE
FGPAINT	ON. .GOSUB	
FGPOINT	ON. .GOTO	UNSAVE
FGPUT	ON. .RESTORE	
FGROT	ON. .RESUME	VAL
FGSCALE	OPEN	VAROOT
FIX	OPTION BASE	VARPTR
FLOAT	OPTION EUROPE	
FLSH	OUT	WHILE. .WEND
FN		

Notably absent was any manual for the user, apart from some preliminary notes put together by Fi-Cord. The programmer's manuals would not normally be supplied to an end user and there is, apparently, a user's manual on its way from the States which, if the programmer's manuals are a guide, should be of excellent quality.

No hardware manuals were supplied with the machine, the only hardware information being a single-sheet sales leaflet with a summary of the system's specifications.

Expansion

Although not officially announced yet, it seems probable that the memory will be expandable to beyond 128k, although by how much and whether this will be an internal expansion or an add-on box just isn't known yet.

Hard disks are on their way, either 5 or 10 Mbytes as an external unit with a built-in tape drive for back-up. Monroe feels this is preferable to the conventional approach of replacing one of the floppies with a winchester drive and backing up onto a series of floppies, although many users seem happy with the latter approach.

Monroe is shortly to announce another system, a visible ledger machine with either 32 or 64 kbytes of RAM and a 100k cassette tapes store which can hold either programs or data. This will link up to the OC 8820 through the latter's communications port for bi-directional data transfer and will have its own keyboard, 40-character strip diplay and integral printer which, apart from printing the ledger cards, will act as a general purpose printer too. It's an interesting idea – there are lots of people around who feel uneasy at trusting their business records entirely to disk or tape and who would like to combine computing with more traditional methods. Anticipated selling price of the visible ledger system is £2-2500.

I mentioned earlier that Monroe is thought to be working on a mini; if and when this appears, the OC 8820 should

GOTO page 187

BRAIN DUMP

Continued from page 73

and time-consuming, Forth is for me an ideal compromise between interactive, structured, high-level programming and fast-running, dirty-hands byte-bashing. Somehow the kick I get out of successfully adding a word to *my* language can only be rivalled by the one I got when my very first program in Basic ran without a syntax error. On a less sentimental note, the language seems to be genuinely all-purpose; it's hard to think of applications for which it isn't suitable if you are prepared to take new approaches guided by Forth's unorthodox structure. For instance, in the field of commercial software Forth suggests a different kind of user interface from that presented by Basic and Pascal. In those languages, a program runs, then it stops and prompts the user for an input, then it runs again. You can of course do it that way in Forth (you can hide Forth completely from the user if you want). But Forth suggests another way in which you have a vocabulary of words which do all the work themselves when you type them, so that 45 SPANNERS adds these to your stock ledger or ITEM FLANGE-REAMER defines a new type of stock item and sets up a record for it.

I have even learned to love 8080 assembler since discovering that the Forth Assembler is fully interactive; one can test assembled code immediately without any linking and such by calling the word containing your code. Even better, you can use structured looping within a code definition (the Forth name for an assembler program).

Perhaps a word of advice is in order for those about to take the plunge from Basic into Forth. The weakness of all the literature I've seen for the likes of us is that it underestimates the strangeness of the part-compiled, part-interpreted, virtual memory environment to those reared on interpreted Basic. The first things you need to know are that the dictionary lives in RAM, that all definitions entered directly from the keyboard go into the dictionary and can't be edited, that definitions you can edit live in source form in disk blocks. You need to know how to find an empty block and how to get it into RAM and how to write into it and how to load the block and how to list the dictionary to see that your definitions are there. All these things need to be told to you at the very beginning but in practice they tend to be told to you halfway through the book and in scattered places because to Forth people they are, presumably, not a problem. To a Basic person, not knowing them produces a curious feeling of insecurity rather like walking on quicksand; you're never quite sure that your program is there or where 'there' is.

But enough of these ramblings; my craving for part-digested programming languages demands satisfaction and there's still C, APL, Smalltalk, Logo, Fortran, Prolog, Wonderpop, Mumps, Shingles and F***ing Ada to go. And so little time. . .

Microtechnology Ltd is at 62 Mount Pleasant, Tunbridge Wells, Kent TN1 1RB. AIM Research is at 20 Montague Road, Cambridge CB4 1BX.

MONROE 8820

Continued from page 117

be able to communicate with it and communication with other OC 8820s will be possible — it's a software requirement not a hardware upgrade.

Conclusions

The Monroe OC 8820 is a well-built micro which makes no pretensions at being other than a no-nonsense business tool. It places the onus firmly on the programmer/software house to provide the user with a friendly, easy-to-use system tailored to his/her specific requirements, but all the facilities to do this are provided.

At as near as dammit £3000 for the basic system, it's definitely an upmarket micro these days, especially considering that it's an 8-bit machine at heart. But the power and flexibility of its operating system compensates for the price and its quality construction equals that of Hewlett-Packard's HP 125, although, in a very definite swings-and-roundabouts comparison, it hasn't some of the HP 125's attractive features.

I disliked the Monroe at first and I feel that others, especially the programmer used to CP/M, may feel the same. But on getting to know it, and on investigating the philosophy behind it, I found it a powerful and pleasant micro aimed at a specific niche in the market, for which it is well suited.

Ultimately, though, the machine's