January-March 1987 Volume 1. Issue 1

Scorpio News

FOR OWNERS AND USERS OF:–

Gemini MultiBoard • Gemini M-F-B • Gemini Galaxy
80-BUS • Pluto • Quantum • 68K-BUS • Nascom
Gemini Challenger • Kenilworth • Gemini MultiNet

P.O. Box 286 · Aylesbury · Bucks · HP22 6PU

ARCTIC COMPUTERS LTD.

6 Church Street
Wetherby
West Yorkshire
LS22 4LP
Tel: (0937) 61644

ARCTIC are specialists in Gemini 80-BUS products with over six years experience.

We are suppliers to industry and education.

Io Research PLUTO graphics specialists for the North of England.

A full range of equipment is on demonstration including Designer, Autocad and the Gemini CAD-8 PCB design system.

PCB Design and Plotting services available.

Suppliers of Graphtec plotters & Staedtler drafting pens.

Demonstrations and callers by appointment only please.
Mail order service available (carriage charged at cost).
ACCESS and VISA accepted.

SPECIAL OFFER TO SCORPIO SUBSCRIBERS

★ EPSON LX86 incl. tractor feed ★
only £310.00 incl. VAT & carriage

JANUARY - MARCH 1987 SCORPIO NEWS VOLUME 1. ISSUE 1.

CONTENTS

EDITORIAL

Welcome to the first issue of Scorpio News. We have agreed with Gemini that for current 80-BUS News subscribers, this newsletter represents the final issue of their subscriptions. We hope that you like it and that you will now subscribe to the rest of the 1987 issues of Scorpio News.

You will find that this first issue has considerable similarity to previous 80-BUS News issues. This is mainly due to the material, much of which was already in hand and was thus inherited along with the 80-BUS subscriptions. We hope that future issues of Scorpio News will start to take a slightly different form.

For administrative simplicity we have decided that subscriptions will always run to the end of a calendar year. This way all subscriptions fall due in December, regardless of when they start, considerably easing the task of issuing renewal forms. Obviously the rate for joining part way through the year will be reduced compared to that for the full year.

We apologize for the somewhat higher subscription rate of Scorpio News compared to 80-BUS News. This is in fact to your benefit! 80-BUS News and its predecessors (INMC and INMC-80) were always directly supported by a manufacturer or dealer, varyingly Nascom, Interface Components and Gemini. Scorpio News is completely independent. There are three main effects of this.

One is that Editorial independence is, for the first time, totally guaranteed and the material in the newsletter will be unbiased towards any specific dealer or manufacturer - as in fact you should start to see in this issue.

The next effect is that time that should be being put into production of the newsletter will not be deflected into other efforts of the company providing the financial subsidies.

Finally, because of the lack of any subsidy, the newsletter has to be totally self-supporting. The circulation size of Scorpio News is extremely low when considered against most other publications - this is obvious because of its extremely specialist subject matter. Production costs are basically fixed regardless of the circulation size, and are therefore high per subscriber if there are few subscribers, while printing costs are high for small quantities, but drop dramatically with volume. And this brings us full circle to why the subscription rates are as they are.

So please, do your bit to help increase the number of subscribers. Tell your friends and colleagues about us. DON'T give them a photocopy of the newsletter - we know that this happened with 80-BUS News at a certain user group that meets in Windsor! - make them buy their own. This is not only contravening copyright laws, and no doubt costing various employers unnecessary photocopying charges, but it is also putting the on-going production of newsletters such as this at risk. To put it quite simply and bluntly - if we can't make ourselves a small profit from doing this then there's no point carrying on. Sorry about that, lecture over, but these "enthusiasts" doing a so called "favour" for their friends do irritate us. Support us and we'll support you.

WHAT'S NEW ?

And so now onto a little product news. There has been a fair amount of activity since the last 80-BUS News was published and so we will try and summarize some of that here.

Newburn
Newburn Electronics, based in Ireland, are becoming an increasing force in the 80-BUS market. They specialize in I/O boards, and now have 5 different ones available (see their advert in this issue). They are also the main distributors for memory upgrade kits for various 80-BUS boards. And finally, they have taken over manufacture of the Microcode 14-slot backplane, as Microcode have decided to drop out of the 80-BUS market to concentrate on some of their other projects.

Gemini
Most of Gemini's efforts of late have been concentrated on their 68000 based product range. The Challenger comes with a 12MHz 68000 processor, 512K RAM, two serial and one parallel ports, 1.2MByte floppy (formatted), battery-backed RTC etc, Winchesters of approx. 20, 30, 46 or 73 MByte (formatted) and optional 1/4" tape streamer. Operating Systems are extra and are currently CP/M-68K, p-system, multi-user BOS (MBOS) and multi-user, multi-tasking Mirage. A neat touch is that multiple operating systems can be installed on a single Challenger Winchester, although only one may be run at a time. Expansion boards available are 4-port serial, 1MByte and 2MByte RAM, and various graphics boards. Available shortly will be an IEEE488 board and an input/output board with various D/A A/D and digital facilities.

On the 80-BUS front there should be a tape streamer available shortly, using audio cassette size digital tapes. There is an Intel 80186 based board in the pipeline that should give "reasonable" IBM compatibility, but availability is unknown. Similarly a 2MByte RAM-DISK is also known to be on its way.

EV Computing
EV are known to be responsible for the 80-BUS tape streamer software, with Gemini being responsible for the hardware. They have also announced a number of new products - see their advert in this issue.

Io Research
A new Pluto 1 is rumoured, similar to the old Pluto 1 but complete with palette. Most of Io's efforts of late have been in IBM compatible Plutos, and they have some very impressive software for these, called "Designer". Arctic Computers of Wetherby can demonstrate most of the Pluto range.

The David Hunt Pages

Farewell 80-BUS News - Hello Scorpio News

May 1986

So it's the end of an era. 80-BUS News is coming to an end. That's about seven years of continuous (if erratic) publication under different guises; a far better record than some computer mags that I could name.

As has been remarked to me by more than one 80-BUS News subscriber, "Is this the end of Gemini's support for the home user?". Well I fear that it might appear so, although the facts could be different. But then again, I don't know! My hesitation could be that I'm feeling distinctly uninformed and more than a bit miffed by the way the decision to pull the plug was taken. Certainly I wasn't consulted, and the first I heard of it was by earwigging to a conversation over lunch at a recent Gemini dealer meeting. Sketchy details were then announced to the dealers at that meeting in the afternoon. It wasn't until I started receiving phone calls from subscribers that I learned about the letter (which of course no-one had thought to send me), announcing the demise, after the publication of volume 4.

So what course now, well certainly I shall subscribe to the CP/M Users' Group, and if asked, may be pen the odd word or two (knowing me, the odd word or 6000), for them. As to keeping informed on Gemini products, updates, bugs and other interesting snippets, I've no doubt that as a dealer I shall find out what's going on anyway. But I suspect that for the general reader, their proposed Gemini Newsletter will not have the same irresistible tendency to remain stuck in ones hand until every word has been read (twice) and then lovingly kept for future reference. A brief scan and then chucked in the bin will, I guess, be more like it.

As for support of the existing user base for Gemini and Nascom products, Gemini insist that the existing 80-BUS product range will continue well into the 1990's with a vigorous process of development of new hardware, some being new products, some being redesigns of earlier products to take advantage of newer technology. However, the trend has been progressively up market of late, and although the pockets of the more serious 'computerphile' are pretty lengthy, and provided the product is the right product then price is of little consequence, there must come a time when the bottom of the most lengthy pockets has been reached. I for one consider the current Gemini developments to be in the wrong direction, profitable for Gemini possibly, interesting for me - I doubt it.....

This file was found unfinished on my 80-BUS disk, and today, the 12th December, I've decided to finish it.

December 1986

Six months on and how things have changed. I have been in a new and different job for five months, still loosely attached to Gemini by merit of a small Gemini dealership in the Amersham area, although that isn't my main concern. My new job has widened my perspective of the 'serious' computer world and what is more they pay me more money!!

Then a few days ago a green leaflet dropped through the letter box together with a request for any unpublished 80-BUS material I might have. So I've blown the dust off the incomplete 80-BUS material I've got lying around and have come up with this amongst other things. As Scorpio sounded much like that other well known sign of the Zodiac, I decided to investigate. A bit of digging soon revealed its independence and that they have gone out and bought a blue pencil of their very own. Just to test their independence I'll write something really contentious for the next issue and see if they print it - Watch this space. I hasten to add that I have nothing to do with Scorpio except in the respect that I hope to become a contributor. Perhaps after my forthcoming effort they won't print anything from me ever again.

As you can see from the earlier paragraphs, I was a 'bit miffed' at the turn of events last May, since then I haven't had the time to worry about such things, I've been too busy learning and revising my opinions. That doesn't mean I've deserted my trusty Nascom/Gemini/DH machine at home, but it does mean that I've spent a considerable amount of time investigating the performance (or in most instances, the lack of) of various IBM micros and their clones. Because that's what sells (in quantity) whilst Gemini's don't (in quantity).

The recent experience with other machines has also opened my eyes to the good points and failings of the more popular competition, but even more so to the lost opportunities and ostrich like behaviour of many UK companies to adapt to the wind of change, however unsavioury that wind might be. Unfortunately the open architecture philosophy of the IBM type machine is with us, and with us for a good many years to come; despite, I suspect, that IBM have only recently realized what a can of worms they have let loose on the world; and despite the fact that following that path could well turn out to be a dead end. None the less, it took someone like IBM to establish some sort of standard, in disk formats, in screen addressing modes, in operating system behaviour, in software portability, et al. And that can't be a bad thing from the user's point of view however much you might disagree with the standards set.

And so to lost opportunities, many entrepreneurial types are importing Taiwanese and Hong Kong made IBM clone cards. They are making money (perhaps not as much as they would like, because there are too many of them) by satisfying an increasing business market demand. Not so much for the main computer cards but for bus cards. Now many small UK manufacturers have always been clever at designing cards for existing systems, yet I can only name one UK designed and built card for the IBM, the I/O Research Pennant graphics card (a sort of different shaped Super Pluto card), there may be more, but I don't know of them. Almost all the imported cards are cheap copies of already established and obsolete American designs, and yet no new cards seem to be forthcoming. Here's a hint to Gemini, the GM870 modem card is a beaut, it's well behaved, and it's understandable. Come on John (Marshall - MD of Gemini), get a hacksaw to it, change it's shape, code convert the Gemini modem software and sell it cheap!!! It doesn't have to be Hayes compatible (nice if it were) but no-one out there is selling a half decent plug-in modem card for the IBM and the market must be enormous. Commit to making enough of them and you could happily sell it cheap enough. Alright, so my suggesting to John to make IBM cards isn't going to help us 8-bit Gemini types directly, but in the long term it could.

For instance, Gemini promised an 8088 or 8086 or 80186 'IBM compatible' adaptor card for the Gemini kit, that is, it's got to run MS-DOS properly. This is no great secret and the idea has been rattling about for a couple of years or more. Where is it? I'm not talking about the Costgold card, I don't know them, and I've never seen a review which convinces me of its worth. Now if Gemini had made that card a couple of years ago, I, and I am sure many others, would have bought it then, if the price had been half reasonable. Instead I'm faced with the prospect of forking up a couple of grand for an IBM AT clone (the only IBM offering which approaches the speed I've taken as being normal for the last three or four years) because the software I write for work has to be 'IBM compatible' these days. True I've got IBM machines at work, but if I want to write things over the weekend I have to lug one of these home with me and they certainly don't qualify as portable machines. Well not unless you have a particular liking for wearing a truss.

What I'm saying is that I have been forced into a position where IBM is beautiful (well not so much beautiful as tolerable) and I'm sure I'm not alone. I would far rather convert my existing kit into a Nascom/Gemini/DH/IBM hybrid than to invest in new kit. Such a conversion might be difficult but by no means impossible, and at least I would know what made it tick. (That's an indictment of IBM's documentation, where quantity is everything but quality and information is non-existent.) Come on John, it's almost too late, I've got to buy this machine by February, and I'd far rather give my hard earned loot to someone I'm charitable enough to think cares, than to some inscrutable chinese represented in this country by a bunch of know-nothing box pedlars!

Dealer Profile - Marston Spurrier

[Ed. - This is the first of our Dealer Profiles, which we hope will be a regular feature, this time taking a look at Marston Spurrier.]

Marston Spurrier consists of Nick Spurrier plus occasional contract programmers and, as he says, "half a wife and three eighths of a dog". So it is one of the smaller Gemini dealerships, is located in sunny Battersea and specialises in the Challenger and 68K-BUS with a particular orientation towards the Mirage Operating System.

How Nick got into the business is interesting because it illustrates a point which we are all trying to put across - that a computer will usually be invaluable in business - and also shows that once the virus which infects anyone who does serious programming takes hold it does not let go.

The time is early 1977. Nick is Chief Executive of Argyle Securities Limited, a public company in property development, investment, retailing, housebuilding, fertiliser manufacturing, and small-time banking, operating through about 132 companies in the UK, France, Belgium, the Netherlands, Switzerland and the USA. And he is going quietly mad, wading monthly through mounds of financial paper in order to keep a grip on precisely what is going on in each company.

Argyle is not a big company being worth just over £20 million (say £80 million in today's money) but the £20 million is represented by £140 million of assets and £120 million of debt; most of the assets are property and the property crash is by no means over. Nick needs a computer. The problem is that there aren't many around and there is certainly no software to buy off the peg. So he buys one of the first Apples which as he says "was pretty useless but a stop-gap" and starts writing programs in BASIC to consolidate financial information into an understandable form.

A few months later on a trip to the USA Nick buys a "proper" micro - a Vector MZ - based on the S100 bus and running CP/M. With some considerable effort, knowing initially nothing about hardware and little about programming, he has a home-made database manager and a cash flow analyser up and running. These provide the basic tools for him to analyse the strengths and weaknesses of his operating companies and to embark on a programme of disinvestment to stabilise the business and produce information to keep his bankers as happy as they could be, when similar companies were crashing to the ground at regular intervals. This is a fairy story (but true) because having access to more "instant" information he is able to sell off companies which have inherent dangers for his group and monitor the sale of properties and debt repayment schedules so that by 1979, Argyle is turned into a "cash shell" and sold.

Coming to the Gemini connection, in 1982 Nick (now a director of Allied Suppliers Ltd, then the 4th largest food retailer in the UK) replaces the Vector with one of the first Galaxys. This is in order to implement a project to finalise the rationalisation of the property assets of Allied, which consisted of 1,250 operating stores and over 800 "investment properties", which he had begun in 1980. He wrote the software for this in Pascal and assembler and so began a "love affair" with low-level programming and the Pascal language. He also became a Gemini fan because of the reliability of the hardware (plug, plug) and soon replaced the Galaxy 1 with a Galaxy 3 (Winchester based system). It is a sad reflection on corporate life that his employers would only permit IBM computers on the premises so he had to buy the Galaxys himself and conceal them as Word Processors. This was also a fairy story (of the Brothers Grimm variety) because Allied was taken over by Argyll Foods, Nick's project was satisfactorily completed, and there was nowhere for him to go but out with a silver-gilt handshake. So in 1985 he started Marston Spurrier.

With a background in larger businesses, and knowledge of the problems facing organisations wishing to implement Financial Control and Information Systems properly, Nick remained steadfastly unaffected by the lure of IBM XT/AT systems and their wealth of software, believing that the future lay in true multi-user

"supermicros". He had been partially involved in the development of the Challenger and just after its launch began working, with Sahara Software, on a port of the Mirage operating system to a prototype machine.

For those of you who don't know Mirage, it is a real-time, multi-user, multi-tasking networking OS specifically developed for the 68000 CPU and, like the Challenger, it is British. The first Mirage-based Challengers were shipped early in 1986 and Marston Spurrier became the Distributor of Mirage language processors and software for Gemini Challenger dealers. As refinements have been made both to Mirage (with a new version due for release as Scorpio News hits the streets) and the Challenger when running Mirage, the combination is (Nick believes) formidable and there is no competition as to price/performance.

Marston Spurrier's business is now wholly based on Challenger systems sales and development of new software and utilities for Mirage. For example, the latest product is a Mirage software driver to read Gemini CP/M (80, 86, 68k) disks. Nick is also a consultant to Sahara Software (the Mirage sales organisation) assisting in porting of existing applications to the Mirage environment. One of the projects he has been involved in will result in the publication in January 1987 of an integrated Word-Processor, Spreadsheet with colour graphics, document formatter with indexing and table of contents generator, software printer font generator and print package to drive laser printers with software fonts rather than the inordinately expensive ROM fonts. This is fully multi-user and will retail at £595 - true multi-user software at a PC price.

All Scorpio News readers, including 80-BUS users!, are welcome at Marston Spurrier's offices at 10 Ransome's Dock, Parkgate Road, London SW11 - just south of both Battersea and Albert Bridges and the first road joining the two Bridge Roads - to look at the Challenger/Mirage combination.

A Review of the Newburn Opto-Input Board by M. Black

For our particular application, we needed to interface a variety of different voltages to the 80-BUS. We saw an advert from Newburn Electronics for their NE871 opto-input board. It seemed just what we needed as the board is self-contained, and unlike the Maas board does not require the Gemini GM816 I/O board to drive it (therefore less BUS slots are required). We ordered a board with plug-in modules for 8 of each 12V, 24V, 50V and 110V sensitivity. We also ordered the industrial Klippon termination.

We received the board within a few days. On inspection, the board seemed well made with a comprehensive manual. It's nice to see that 80-BUS manufacturers now include board-ejectors as standard, this saves many grazed knuckles.

We set the board address to Hex $40 using the on-board Dil-switch and plugged the board into the rack. We used 40 way ribbon cable to connect the board to the Klippon block. The Klippon block mounted easily on the rear of the rack using standard Klippon rail.

Connecting a suitable voltage to the correct Klippon terminals, lit the corresponding LED on the board. We found it very handy being able to see the state of all inputs using the LEDs.

Programming of the board was carried out using Turbo Pascal. This was very simple as all inputs are read from 4 ports, in this case Hex $40 to Hex $43. Each bit of the port corresponds to one input.

In conclusion, we found the board ideal for our application. Gone are the days of birds nest wiring and resistive dividers hanging from the rear of connectors etc. Real inputs can be easily connected using the Klippon termination rail. This gave us the professional finish we required.

An Introduction to FORTRAN by P.D. Coker

(In this article, F80 refers to Microsoft FORTRAN-80, ProFortran to the 8-bit version of FORTRAN from Prospero Software. NFortran refers to Ellis Computing's Nevada Fortran and FORTRAN refers to other versions such as FORTRAN IV or FORTRAN 77.)

WHY USE FORTRAN?

FORTRAN is, as computer languages go, very old, having first seen the light of day about 30 years ago as a scientific and technical high level language for early IBM mainframes - it's older than ALGOL and BASIC (which owes a lot of its style to FORTRAN II) and, by comparison, PASCAL and C are infants!

Most users are aware of the drawbacks of interpreted languages - particularly in terms of execution time. FORTRAN is a compiled language in which the source code is written using a text editor and then compiled into a run-time package which will execute much more rapidly than is the case with an interpreted language. There are two disadvantages to this, however. One is that any amendments which need to be made to the program involve editing and recompiling the source code which can be tedious. The other problem is one of cost. Virtually all compiled languages are expensive (£150 - 350), although HiSoft PASCAL (a very satisfactory version) costs quite a lot less. There is a very good FORTRAN compiler, sold by Grey Matter of Ashburton, South Devon which is incredibly cheap. For £30 you get Nevada Fortran and a useful Guide - admittedly, the implementation of FORTRAN is not complete - but few will miss the bits the author has left out!

So, if you are fed-up with your BASIC and want to try a compiled language, then it really is worthwhile trying NFortran. True, FORTRAN is a little less flexible and efficient than PASCAL but the conversion of programs from BASIC to FORTRAN is relatively simple and straightforward. I must admit to being professionally biased in favour of FORTRAN but some versions of PASCAL are very finicky about syntax.

PROGRAM STRUCTURE

In BASIC, program entry is very straightforward with line numbers and statements typed in directly but in FORTRAN, the program statements must first be entered using a text editor such as WordStar (in non-document mode) or PEN before calling the compiler. All versions of FORTRAN use up to 80 columns per line of input, organised as follows:

Cols	1 - 5	Optional statement number (1 - 99999)
Col	6	'Continuation' character field (used only if the number of columns used by the statement exceeds 72)
Cols	7 - 72	FORTRAN statement
Cols	73 - 80	(can be used for identification, rude words or whatever).

COMMENTS

In BASIC, REM statements (or comments preceded by a single quote (')) can be used to highlight information about programs. In FORTRAN, the use of a letter C in column 1 performs the same function; lines so marked are ignored by the compiler.

STATEMENT NUMBERS

In most BASICs, it is necessary to give each line of the program an identifying line number in ascending sequence since lines are interpreted on an incremental basis. FORTRAN differs since only certain statements are required to have numbers and program execution does not depend upon a numerical sequence, but on the way in which statements are arranged - it starts at the first line and finishes at the last! Statement numbers are usually only required during loop operations, data I/O or where control has to be passed to another part of the program.

CONTINUATION LINES

There are occasions when text headings or complicated expressions or layouts are used in a program where the space available for the statement (cols 7 - 72) is insufficient. A continuation line (or lines) needs to be used to accommodate the additional material; in such a case, a character must be typed in column 6 and columns 1 - 5 left blank. Usually the first continuation line is labelled with an 'A', the second with a 'B' and so on; the figures 0 - 9 could also be used if there are a lot of continuation lines and F80 and ProFortran place no restriction, unlike some mainframes where a limit of 10 - 20 is quite common. It is allowable, but not good practice to use the same symbol for succeeding lines. The normal use of only 72 columns is a hangover from the days of teletypes and was maintained when the majority of input was in the form of punched cards.

PROGRAM ENTRY

It is perfectly possible to write small FORTRAN programs on the back of an envelope before typing them into your micro - but a great deal easier if you use a proper coding form for the purpose in which the various columns and fields are shown; a typical example is shown in fig. 1. A flow chart is useful when constructing a new program; note that control characters which the text editor may insert MUST BE REMOVED - the compiler tends to get uppity if it receives ^U or ^G. The program must not be formatted in any way and I would not recommend the use of DR's ED which is only for the masochists - PEN, WordStar or HiSoft's ED are much better.

The statement must start in or after column 7 of each line, with a statement label if needed in columns 1 - 5; the 'tab' facility could be used if one has a dislike of using the space bar although this will usually start the statement in column 9. If a continuation line is required, column 6 must be used as noted earlier; program statements must finish in or before column 72 unless a continuation line is to be used.

DATA NAMES AND TYPES

FORTRAN defines data specifically by name and, unlike most forms of BASIC, by type.

There are four names for data; constants (such as an explicitly stated numbers or pieces of data); variables - which are symbolically identified pieces of data; arrays which are ordered sets of data in 1,2 or 3 dimensions and array elements which refer to one member of the set of data in an array.

Data types may be Integer, Real, Double Precision, Logical or Hollerith.

INTEGER types are precise representations of whole numbers, positive, negative or zero in the range -32768 to +32767 and have a precision of 5 digits.

REAL types are approximations of real numbers, to 7+ significant digits which may have a magnitude lying between 10^-38 and 10^38 (or 2^-127 and 2^127).

DOUBLE PRECISION types are real data with an accuracy of greater than 16 significant digits with the same range as real data.

LOGICAL types are single byte representations of .TRUE. or .FALSE. with .FALSE. equivalent to zero and .TRUE. has a value of -1; in practice, any non-zero value will be treated as .TRUE. in a logical IF statement.

HOLLERITH types (named in honour of the inventor of the punched card tabulator) consist of a string of any number of characters from the computer's character set. They are sometimes known as 'TEXT' or 'LITERAL' types.

F80 does not support COMPLEX data types, but ProFortran does. A COMPLEX variable has two components - a real part and an imaginary part - each of type REAL.

FORTRAN constants are of types INTEGER, REAL, DOUBLE PRECISION, LOGICAL, HOLLERITH and, additionally in F80, HEXADECIMAL.

The INTEGER type has from 1 - 5 decimal digits together with a '-' sign if appropriate. e.g 521, 2, -45, 00012

REAL types have 7 digit precision and are represented either in floating point (F) or exponent (E) format. For example, 3.21456 is a floating point REAL constant and 0.321456E 1 is the exponent equivalent - as in BASIC the use of the 'E' indicates '10 to the power' and the figure that follows is the index of the power - in this case, 1. REAL numbers can be positive or negative and must have a decimal point. They will be truncated if the number of figures exceeds the stated level of precision.

DOUBLE PRECISION type constants have a higher level of precision and obey the rules for REAL numbers - but with more significant figures or the use of 'D' in the exponent format - thus 4.172345860098 is a double precision number as is - .786D 4

LOGICAL constants are either .TRUE. or .FALSE. and generate FF (hex) and 0 (hex)

LITERAL constants are strings of any character except the single quote ('), enclosed by single quotes. 'QWERTYUIOP' is one such constant.

HEXADECIMAL constants are found in F80 but rarely in other FORTRAN versions. The constant consists of up to 4 hexadecimal digits enclosed in single quotes and preceded by a Z or X. Z'FA12' and X'2C' are examples.

FORTRAN variables are identified by symbolic names which are strings of 1 to 6 alphanumeric characters which are unique to the program in which they are used. The first character must be a letter (apart from the dollar sign - which is reserved for system variables and runtime subprogram names. The restriction of variable names to a maximum of 6 characters is a consequence of the IBM origin of the language.

Variables are (by default) of types denoted by the initial letter and the convention is that integer variables have names beginning with any letter from I to N and real variables begin with A - H and O - Z. To complicate matters slightly, a variable may explicitly be assigned to a different type by declaration at the beginning of a program or subroutine. Thus START is a real variable and ISTART is an integer variable by default, but by means of an explicit declaration statement:

```
INTEGER START          REAL INPUT
```

as described, the program will treat variable START as an integer and INPUT as a real. Double precision and logical are declared in the same way.

ASSIGNMENTS

This is the correct way of describing statements which contain an 'equals' sign. The use of the '=' sign in FORTRAN is exactly the same as in BASIC and PASCAL.

ARITHMETIC OPERATORS

Again, this is the correct way of describing the +, -, *, /, and ** operators. The first four are familiar enough in BASIC but the fifth replaces the 'up-arrow' used to indicate exponentiation. The order in which the operators work is as follows - from left to right in three sweeps; these deal respectively with any exponentiation, followed by multiplication or division and finally the addition or subtraction. There is no precedence between addition and subtraction or division and multiplication and the use of parentheses (brackets) can be used to emphasize or change the natural order which has just been described. Again, there is no difference between BASIC and FORTRAN.

ARRAYS

Arrays are characterised by having 1, 2 or 3 dimensions (or more in some versions of FORTRAN!). The size of an array must therefore be declared explicitly at the beginning of a program or subroutine, by using the DIMENSION statement in which the array name (1 - 6 characters, the first of which is a letter) and its dimensions are stated. The process is broadly similar to BASIC and the relevant programs line appears thus:

```
DIMENSION SQUARE(10,10),CUBE(5,4,3),LINE(20)
```

gives the dimensions of two REAL arrays and one INTEGER array; SQUARE has two dimensions with a maximum of 100 elements, CUBE has three dimensions and 60 elements and LINE is a single dimension linear array (vector) which can store up to 20 INTEGER values only, just as CUBE and SQUARE can only store REAL numbers. If the DIMENSION statement is preceded by an explicit variable declaration - such as making CUBE an integer variable, the array CUBE becomes capable of holding only integer data. The values in brackets after the array name are referred to as subscripts and identify the array elements; on the first occasion that an array is declared, the dimensions must be numerical but if the array is required in a subroutine (the FORTRAN 'equivalent' of the GOSUB - RETURN or the PROCedure in other versions of BASIC), these numerical values can be replaced by letters or other variable names provided these have been given a value before the subroutine is called. The value of the subscript should be adequate to deal with the anticipated amount of data so that if 105 values are input into array SQUARE, there will be an overflow and program execution will cease. One dimension and two dimension arrays are fairly straightforward and are widely used in BASIC, and the method of accessing a particular array element is identical in FORTRAN. The three dimension array is simply a series of two-dimension arrays arranged in 'layers'. In the example given above, CUBE consists of 3 'layers', each with 5 rows and 4 columns; so that CUBE(4,3,2) = 10.5 indicates that the array element in the fourth row of the third column in the second layer has the value 10.5 assigned to it and CUBE(5,4,3) = -1.5 indicates that the very last element (row 5, column 4, layer 3) has the value -1.5 assigned to it.

It is possible to reduce storage requirements by three methods; one relies upon the different amount of storage required by each data type. One REAL data value occupies 4 bytes, DOUBLE PRECISION takes 8 bytes, INTEGER takes 2 and LOGICAL, 1 byte. Space can be saved by making sure that data is given an appropriate type - serial numbers, for example can be stored as integers rather than real numbers. Another method uses the EQUIVALENCE statement in which arrays can share the same storage area - preferably if they are of the same type (integer, or real). The statement has the general form:

EQUIVALENCE (X1),(X2)....(Xn)

- where X1..Xn are sequences of two or more variables or array elements, separated by commas. Thus,

EQUIVALENCE (GROT(6),THISGROT(3,2))

- allows the two arrays to share space happily.

Variables can also be EQUIVALENCED -

EQUIVALENCE (A,D)

- is allowed.

If EQUIVALENCE is used to save program space, then make sure that you do not fall into the trap of letting one of the equivalenced variables or arrays overwrite the other(s). The EQUIVALENCE statement is very useful if a program contains intermediate variables or arrays which are not required in later stages of a program.

The final method of space saving is to use the COMMON block statement in which arrays and variables which appear in more than one of the program subroutines are declared as COMMON at the beginning of each:

COMMON /X1/ A(20,40),NASTY(100,2)

- are put in each subroutine to save a modicum of space.

In a large program this is very handy! In the above example, arrays A and NASTY are stored in a COMMON area called X1. This area is shared by all subroutines in which this COMMON statement appears.

EXTERNAL statement

Many functions are provided in the system library and can be called up (as in BASIC) by including their name - for example, SIN, ABS or PEEK (in F80 and ProFortran). But others are not available and have to be provided from another subroutine or external function. The EXTERNAL statement does this by allowing the programmer to specify the name of the external subroutine that does the required job - thus if you wanted to calculate the volume of a cylinder, a subroutine which provided the area of the top could be called in the EXTERNAL statement as follows:

```
C       POTTY PROGRAM
        EXTERNAL CIRCAR
C   THIS IS AN EXTERNALLY SUPPLIED FUNCTION WHICH GIVES THE AREA
C   OF A CIRCLE
        READ(5,10)HEIGHT
10      FORMAT (F6.3)
        VOL=HEIGHT*CIRCAR
        WRITE(5,100) VOL
100     FORMAT('CYL. VOL IS ',F6.3,'CUBIC METRES')
C   THIS IS FINE AS LONG AS ALL CYLINDERS HAVE THE SAME SIZE TOP!!!
        END
```

Note that if the EXTERNAL statement had been omitted, CIRCAR would have been incorrectly treated as an ordinary REAL variable by the computer.

DATA statement

This is not the same as the DATA statement in a BASIC program. The function in FORTRAN is to allow the programmer to set up initial values for variables before program execution occurs. For example, the values of frequently used constants could be initialised in this way:

```
DATA  PI/3.142/,BEANS/5.0/.....
```

In each case the variable name is followed by a solidus (/), the value of the variable, another solidus and separated from the next name by a comma.

STATEMENT ORDERING

It is most important, particularly with type declarations and array declarations, to ensure that these are put in in the correct sequence before ANY executable program statements.

A suitable order for most FORTRAN versions is:

```
REAL
INTEGER
DOUBLE PRECISION
LOGICAL
DIMENSION
COMMON
EXTERNAL
EQUIVALENCE
DATA
.
. (executable statements)
.
END
```

This applies to both main program and all subroutines, but not all of these will be required for most programs (thank goodness!).

SUBROUTINES and FUNCTIONS

These are program sub-units which are called by the main program as required. They do not occur in precisely the same form in most versions of BASIC (where the GOSUB...RETURN statements have an approximately similar operation) - BBC BASIC uses PROCedures which are a closer approximation. There is an essential difference between SUBROUTINEs and FUNCTIONs in the way the main program uses them; the SUBROUTINE is CALLed by name using the CALL statement and when the program statements contained in it have been processed, control is passed back to the main program by means of a RETURN statement. Typical uses for a subroutine are where a series of tables have to be set up or where graphical data has to be displayed or a complicated expression has to be evaluated. FUNCTIONs are used differently. The function name appears as one of the variables in an assignment statement and the value of the function (as calculated) is used in the evaluation of the assignment. For example, if the value of pi is required for some calculations involving the area or circumference of a circle and the version of FORTRAN you are using does not supply this as one of its intrinsic functions, then a FUNCTION subprogram could be written to provide it (only a dodo would actually do this but it is a good illustration!):

```
      FUNCTION PI
C  PROVIDES THE VALUE OF PI AS REQUIRED
      PI=3.14159
      RETURN
      END
```

The form of the FUNCTION statement is as follows:

FUNCTION name (dummy arguments - if needed)

or, more explicitly,

type FUNCTION name (dummy arguments)

The reason for the second form is that the type can be explicitly declared as INTEGER, REAL, DOUBLE PRECISION, LOGICAL or (in ProFortran) COMPLEX. Thus, if the function required was called LENGTH and it was a REAL number rather than an INTEGER, the correct form for the FUNCTION subprogram which gave the single value for LENGTH would be:

REAL FUNCTION LENGTH

The same rule applies if the function name is one which by default would be a REAL if an INTEGER value was required. For example:

INTEGER FUNCTION AREA

INTRINSIC FUNCTIONS

In all high-level languages, there is a range of particular functions for which the necessary code is supplied - such as INT or ABS in BASIC. These are termed Intrinsic functions and are called by name to produce a particular value. There are rather more intrinsic functions in FORTRAN than will be found in most BASICs since some are available in integer, real and double precision forms. There is always a list of these in the documentation associated with the version of FORTRAN you are using. Some old favourites like CHR$ and LEFT$ are never found in FORTRAN, others are spelt differently - SIGN rather than SGN - but most are readily interpreted.

EXTERNAL FUNCTIONS

Again, these have their counterparts in BASIC although many are available in double precision or even complex number forms in addition to the conventional real number form - typical of these are the trigonometric functions, SIN, COS or TAN, and SQRT - spelt differently to the BASIC way (SQR). There is log base e (ALOG) and log base 10 (ALOG10) and a random number generator - variously spelt RND or RAND. A list of these will also be found in the documentation, together with a list of restrictions on the type of variable with which they can be used - thus if you want the square root of a variable it must be of type real or double precision and if it is an integer, this must be converted to real by the use of a cunning little intrinsic function called FLOAT. Failure to do this first will lead to an execution error and the program grinding to an unprofitable halt. An external function called IFIX does just that to REAL numbers, converting them to INTEGER type.

DATA INPUT/OUTPUT

On first acquaintance, this appears to be rather more complicated than in BASIC, but it is a great deal more flexible. Data to be read into or results output from a program can be in almost any desired form provided the machine is properly instructed by means of a FORMAT statement. Data is read in from either keyboard or disk file using a READ statement and written to the screen, disk file or printer by means of a WRITE statement. In their basic form these are as follows:

READ (u,f) k or WRITE (u,f) k

where u is the physical or logical unit number, f is the statement label of the associated FORMAT statement and k is a list of variable names, separated by commas although it is optionally possible to use the form:

```
READ (u,f,ERR=L1,END=L2) k    or    WRITE (u,f,ERR=L1) k
```

where L1 is the label of the statement to which control is passed in the event of an I/O ERROR and L2 the statement label to which control is passed if an END OF FILE is encountered. These are extensions of data I/O not found in the 1966 standard.

The Logical Unit Numbers referred to above are normally assigned as follows:

Units 1, 3, 4 and 5 are assigned to the console/VDU; Unit 2 is assigned to the printer. Units 6 - 10 are assigned to disk files. Units 11 - 255 may be assigned as the user wishes (and units 1 - 10 may also be re-assigned if you really want to!)

For most purposes the default assignments will be found adequate but for further information, the appropriate reference manual must be consulted.

A typical READ statement will look like:

```
      READ(5,35) K,L
   35 FORMAT(I3,2X,I4)
```

in which two data entries, K and L are read in from the keyboard with field widths of 3 and 4 respectively, separated by a blank field of 2 spaces. The same holds for a WRITE statement since, if it is in the same program subroutine,

```
      WRITE(2,35) K,L
```

will cause the values of K and L to be output on the printer, starting in column 1 and separated by 2 spaces. Statement numbers must not be duplicated in the same subroutine but more than one READ/WRITE statement may reference a given FORMAT statement.

Alphanumeric text material (letters, symbols and figures) may be used as program I/O by means of a FORMAT statement in one of two ways which employ Hollerith field descriptors; the first (and less convenient) method involves the programmer in counting the exact number of characters and spaces to be input or output - tedious for long titles, while the other simply involves placing the entire text between single quotes ('). In both cases, the program statement must not run beyond column 72; if a longer text string is needed, then a continuation line is used. The statements:

```
      WRITE(6,100) L
  100 FORMAT(20HDATA FROM EXPERIMENT,I3)

      and

      WRITE(6,100) L
  100 FORMAT('DATA FROM EXPERIMENT',I3)
```

are treated in the same way by the compiler but the first uses the Hollerith (H) field width descriptor - 20 units wide in this case and the same as the text including spaces while the second uses single quotes as text delimiters - a lot easier to use! The expressions mean 'write to a disk file the text followed by the current value of the variable L'.

On both input and output, text can be split between lines by using the solidus (/). As far as input is concerned, this applies to disk input data files as well as to punched cards or paper tape - both of which are unlikely input media for the average micro user. The first example given above could be rewritten as:

```
100 FORMAT(9HDATA FROM,/,10HEXPERIMENT,I3)
```

which would print out as:

```
DATA FROM
EXPERIMENT  1   (assuming that L had a value of 1)
```

F80 and ProFortran recognise ten different types of I/O field descriptor:

	A	Alphanumeric [A - Z, 0 - 9 etc.]
	D	Double precision numeric
	E	Exponent form [e.g. 1.1E04 = 11,000]
	F	Floating point [e.g. 12.34]
	G	(can be used for floating point or exponent)
	H	Hollerith (string)
(or)	'	alternative to Hollerith
	I	Integer data [e.g. 12, 560]
	L	Logical [T(rue) or F(alse)] - not often used.
	P	optional scaling descriptor used with D,E,F and G conversions either as a multiple or a fraction
	X	blank space

These are fairly standard for most FORTRANs but NFortran does not have Double Precision or the scaling descriptor P. It does have T (tabulation), K (hexadecimal) and Z (inhibit <cr,lf> on output) which might be useful.

Most users will only be concerned with A,E,F,H,I and X descriptors; apart from the H and X descriptors, which have the general forms nH or nX where n is the number of columns each descriptor covers, the rest have the operational form rZy where r is the number of repetitions of the field descriptor if this is grater than 1, Z is the descriptor type and y gives information about the total width of field required for each repetition.

Some examples would not come amiss here -

10A4	indicates 10 alphanumeric fields each 4 columns wide
E8.2	indicates 1 exponent field 8 columns wide, where there are two figures after the decimal point. Thus 12345.67 is represented as 0.12E_05 (where the '_' represents a blank column in which nothing is printed)
E12.5	is a single exponent field of 12 columns in which 5 figures follow the decimal point and a figure such as -12.345678 would be represented as -0.12345E_02. This implies that the number is truncated (shortened) and loses some accuracy.
F3.0	shows that a floating point number occupying 3 columns is involved which has no significant numbers to the right of the decimal point. 7.0 or 2.0 would be shown in this way.
F8.4	shows that the floating point (real) number occupies 8 columns and has 4 digits to the right of the decimal point. 123.4567 is a suitable example; a real number such as -34.89088 would be shown as -34.8909, since the field only permits a maximum of 4 digits after the decimal point. Overflow can occur if the number of digits to the left of the decimal point cannot be fitted into the space available - thus 1234.56 would be printed as *.56 - the asterisk shows that the field width was too small.
4I3	indicates that there are four 3 digit integer fields involved; thus 123, 245, 778 and 200 are represented as: 123245778200 while -30, 2, 45 and 559 are represented by -30_2_45559. In the latter case, the '-' sign occupies 1 of the columns in its field and the numbers with less than 3 digits are right justified.
4X	this shows that 4 columns are to be skipped (not read or printed)

Field descriptors can be mixed in a format statement if a particular mode of I/O is required. Thus if a record such as a batch identifier and number, followed by various parameters is to be read or written, the associated format statement could look like this:

```
FORMAT(A4,2X,I3,2X,3F6.3)
```

This looks complicated but means that a 4 letter batch identifier and 3 digit batch number are followed by details of 3 parameters expressed as floating point numbers; the batch i/d and serial numbers are separated by 2 spaces as are the serial number and parameters. The output would appear as:

```
¯LOT¯¯123¯¯23.34517.27011.555
```

This looks a bit messy and the parameters should be spaced out; this can be done simply by amending part of the format description to include spaces:

either - (A4,2X,I3,2X,F6.3,2X,F6.3,2X,F6.3)

which works but is untidy

or - (A4,2X,I3,3(2X,F6.3))

which also works but is better programming practice.

The effect of the change is as follows:

```
¯LOT¯¯123¯¯23.345¯¯17.270¯¯11.555
```

The overall effect is to carry out the reading or printing of 3 sets of similar combined fields and it should be noted that combinations of field descriptors used in this way should be enclosed in round brackets.

CONTROL STATEMENTS

These guide the flow of the program and are of ten types (some of which are identical to their BASIC equivalents)

1. GO TO	2. IF	3. STOP	4. ASSIGN	5. DO
6. CONTINUE	7. CALL	8. PAUSE	9. RETURN	10. END

GO TO statements

These may be unconditional (e.g. GOTO 100 which transfers control to another part of the program just as in BASIC); computed [e.g. GOTO (80,90,100,123) K, where K is an integer variable such that if K has a value of 3, then control is passed to the statement label which is third in the list - in this case, 100]. Note that if the value of K is 0 or greater than the number of statement numbers in the list (4 in this case), then control passes to the next logical statement in the program sequence.

Assigned GO TO statements are not often used and the ASSIGNed statement label(s) must be declared before they are used in this way - thus:

```
ASSIGN 100 TO MILK    - ensures that in an assigned GOTO of the form
GOTO MILK             - control is only passed to statement number 100
```

More than one ASSIGN statement can be used so that if:

```
ASSIGN 212 TO MILK    - the next assigned GOTO which mentions the variable
                        MILK will look like:
GOTO MILK,(100,212)   - and control will be passed to whichever statement
                        number corresponds with the current value of the
                        variable MILK.
```

IF Statement

Two types of IF statement are used in FORTRAN - the arithmetic form and the logical form. The arithmetic IF takes the form:

IF(expression) L1,L2,L3 [L1,L2 and L3 are statement numbers]

If the value of the expression is less than 0, control passes to statement no. 1, if it is equal to 0 then to statement no. 2 and if it is greater than 0, control passes to the last named statement -

IF(N-1) 3,5,7 - if the value of N-1 is 0 then control passes to the statement label no. 5.

The logical IF statement has an almost exact parallel in BASIC and it takes the form:

IF(logical expr.) statement

The logical expression is evaluated as .TRUE. or .FALSE.; if true, control passes to the statement immediately following otherwise control passes to the next program line. The logical expression can contain both logical and relational operators, which are defined as follows:

Logical operators

.NOT. .AND. .OR. .XOR.

Relational operators

.LT.	less than.	.LE.	less than or equal to
.EQ.	equal to	.NE.	not equal to
.GT.	greater than	.GE.	greater than or equal to

Relational operators are self explanatory but logical operators may need a little clarification.

If A and B are logical expressions, then:

.NOT.A	is the logical opposite of A (0 bits become 1 and vice versa)
A.AND.B	the value of this expression is the product (logical) of A and B. This implies that the value is .TRUE. if A and B are .TRUE. and .FALSE. otherwise
A.OR.B	this produces the logical sum of A and B. In this case the value is .FALSE. if A and B are .FALSE. and .TRUE. otherwise.
A.XOR.B	the value of this expression is the exclusive OR of A and B. Here, the value is .TRUE. if only one of A and B is .TRUE., otherwise it is .FALSE..

STOP statement

This is self explanatory. It causes program execution to cease at that point; if the form:

STOP (string of 1 - 6 characters)

is used, then the characters in the string will be printed out on the monitor and program execution then finishes.

PAUSE statement

This command also stops the execution of the program. It either takes the form:

PAUSE or PAUSE (string of 1 - 6 characters)

The characters, if present are displayed as before and program execution can be resumed if needed by typing any character other than 'T' - which terminates the program at that point. This statement should be used with care since not all FORTRAN compilers support it in precisely this fashion.

CALL statement

This statement transfers control into subroutines and provides parameters for use by the subroutine. It has the general form:

CALL subroutine name (dummy arguments)

Dummy arguments are values of expression which need to be transferred from the CALLing program to the subroutine so that it can work properly. A more detailed explanation is given in Alcock (pp 68-69) or in the Microsoft F-80, NFortran or ProFortran documentation.

RETURN statements

These are always used when control is to be passed back from the subroutine to the CALLing program.

DO statements

These provide a means for executing a series of statements repetitively, just as in the FOR - NEXT loop in BASIC. The DO statement takes the following form:

```
DO J K=M1,M2,M3     - or, where M3 is 1, it may be omitted
                    [J is a statement label, K is an integer or variable
                    and M1, M2 and M3 are integer constants or integer
                    or variables]
```

A typical example might be:

```
      DO 14 I=1,5
       .
       . {a series of executable statements}
       .
   14 PROD=PROD + A(I)
      WRITE(5,100) PROD
  100 FORMAT(F6.3)
```

in which the program executes the DO loop five times (as I goes from 1 to 5) before writing out the result. The statement label J must actually occur and it must not be a STOP, PAUSE, RETURN, arithmetic IF, GOTO or another DO. The controlling index K must be an integer and always positive; M3 is the increment thus if the first line of the DO loop was replaced by:

DO 14 I=1,9,2

successive loops would increment the value of I by 2 (1,3,5,7,9) rather than singly; this is analogous to the STEP statement in BASIC and some versions of PASCAL. DO loops can be nested so that one or more may exist within the range of a larger loop but these nested loops must terminate inside the main loop. Other features of DO loops are noted in the Microsoft or Prospero reference manuals.

CONTINUE statement

This is normally used as the terminator of a DO loop. Although it is classified as an executable statement it really does nothing. It is very useful if the normal terminator of a DO loop is one of the forbidden statements (GOTO, STOP etc.)since it allows the successful completion of the loop before control is logically transferred to the next line of the program. In the above example, statement label 14 could be altered to:

```
      PROD=PROD + A(I)
   14 CONTINUE
      IF(PROD)  24,56,71
   71 WRITE(5,100) PROD
            .
            .
      etc.
```

In this case control is passed to 71 and the value of PROD is printed if it is greater than 0.0; otherwise control is passed to another part of the program.

END statement

This is always the last statement in a FORTRAN program and its execution causes control to be passed to the system exit routine and thence to CP/M; it has the form:

```
      END
```

and may have, optionally, a statement number.

CONCLUSIONS

This article was not intended to teach prospective users how to use FORTRAN, but to give intending users some idea of the way in which it can be used. A couple of books are worth mentioning which will give a more comprehensive coverage of the language. Wainwright and Grant's contribution is quite handy since it treats BASIC and FORTRAN in parallel (there is a similar book for BASIC and PASCAL) but the book is poorly produced by Babani and includes a simple FORTRAN 'interpreter' written in Spectrum BASIC - a nasty bit of work with many errors. Alcock's book is in the same mould as his original book on BASIC and is well worth getting if you like his style. I recommend it to all my students since it is infinitely more readable than the majority of so-called text books on FORTRAN.

References

Alcock, D. (1983) Understanding Fortran. Cambridge Univ. Press.

Wainwright, S.J. and Grant, A. (1984) BASIC and FORTRAN in parallel. Babani.

The Computer User's Dictionary by D.R. Hunt

To err is human, but to make a total cockup you need a computer.
....S. Power 1986

ASCII n. A table of numbers and letters designed to make communications between computers standardized, unless it is called EBCDIC.

assembly v. The act of converting a fully working and debugged source program into a non-working object program.

backup n. A rarely used copy of important data, usually two generations out-of-date or corrupt when required.

backup v. The act of copying out-of-date data over the only existing copy of up-to-date data. The regularity with which this is performed is in inverse proportion to the importance of the data.

bug n. A hidden and undocumented feature within a program designed to perplex the average user of a program by producing unpredictable results from a given input.

bus n. An incompatible method of connecting various computer parts together.

comms program n. A program specifically designed to allow two computers to communicate with each other until actually used.

compiler n. A program, the purpose of which is to convert flawless source files into programs containing hidden features specially introduced at the whim of the compiler writer.

compile time a. The time taken for a compiler to complete its task. Usually measured in units of time known as cigarettes. Compiling small programs with fast compilers usually takes less than one cigarette, in the case of some compilers and large programs, the programmer has usually expired of lung cancer before the task is complete.

computer n. An electronic machine designed by the initiate for use by business offices and other places of work to dull the senses, to increase the work load and to complicate the decision making process.

computer op. n. One whose job it is to sit in front of a terminal and watch the lights blink.

computer purchaser n. A person responsible for the purchase of a computer system based on the size of the lunch offered by the computer salesman.

computer salesman n. An enthusiastic person who talks a lot, thinks a little and knows nothing.

computerization v. To create disorder out of order by using a computer.

corrupt data n. A strange form of computer dyslexia. Data which should normally be used for preparing bills or invoices, also for name and address lists. Data which has been entered correctly and yet contains peculiar errors.

data n. Information which would normally be stored on paper, but has been committed to computer in error.

database n. A disk file which contains data in a form which is impossible to reconstruct when corrupt. The likelihood of a corrupt database is in direct proportion to its length.

data prep. v. The human act of committing faulty information to computer using the wrong program incorrectly.

debugging v. The act of removing bugs from a program. The number of bugs found is always one less than the number of bugs present.

disk n. A flat rectangular object bearing no relation to a disk, used to store data and programs.

disk drive n. A machine for wearing out disks and for overwriting important data.

EBCDIC n. A table of numbers and letters designed to make communications between computers standardized unless it is called ASCII.

format n. A conspiracy between manufacturers to ensure disk incompatibility.

format v. The act of instantly destroying any required data. Particularly where backup copies do not exist.

hard disk n. A more expensive and intricate version of a disk drive, usually justified where speed and data size are not a prime consideration.

head crash n. A particularly useful way of explaining loss of data or justifying large service charges. The deliberate act of an inanimate object, the hard disk, to behave in a perverse animate way in loading the disk drive head physically on to the hard disk whilst it is in motion.

head crash v. Banging of the head on the terminal to alleviate the anguish caused by the use of a computer.

IBM n. A large faceless international conspiracy of non-political computer salesmen dedicated to complete world domination and to propagating the Great American Dream.

language n. A syntactical grammar used by the programmer to write a program. Simple languages are usually easier to learn than Serbo-Croat whilst some complex languages look more like Mandarin Chinese.

mainframe n. A large and complex machine designed to maintain the mystique and mystery surrounding systems analysts and programmers.

microcomputer n. A smaller less complex one operator version of the mainframe designed to produce symptoms of acute frustration in operator.

manual n. A door stop or table support. A printed list of instructions for using a program; a manual must be written in an incomprehensible language and have little relation to the program in question. The size of the manual is in *inverse proportion* to the complexity of the program.

modem n. A display box to advertise either a red triangle sticker or a green circle sticker. Green circle stickers are officially preferred, red circle stickers are cheaper and more useful.

network n. A method of connecting a number of computers together in such a fashion that the degree of frustration caused by using the network increases with the square of the number of computers connected.

object n. The string-of-number instructions to a computer produced by assembling a fully understandable source program into totally incomprehensible garbage.

operating system n. An interface program between the computer and the program to ensure that program interchange is impossible.

peripheral device n. An adjunct to a computer the purpose of which is to translate the results of computation into such a form that can be understood by the average moron or another computer.

printer n. A mechanical paper shredding machine.

program n. A list of instructions to a computer which rarely does what is required, or when it does, contains hidden untested features which preclude its proper use.

program v. The writing of a perverse list of instructions to a computer usually written in a language incomprehensible to the layman and programmer alike.

program crash n. The result of any working program which stops for no reason whilst performing a large or unrepeatable task. This could be a special feature, see bug.

programmer n. One who attempts to write a program, the logic of which has been ill understood from the outset and is beyond the mental capacity of the person writing it.

protocol n. A method of specifying the order in which things are to be done, usually refers to communications between computers where identical protocols are certain not to talk to each other.

RAM n. Random Access Memory. The area within a computer for storing or manipulating programs and data. Parkinson's Law applies in that programs and data always expand to fill the available RAM.

realtime n. Some feature of a program which acts automatically and is too fast for the intervention of the operator.

serial interface n. A method of connecting a computer to a peripheral device which either too fast or too slow or has the wrong protocol for the device in question.

service company n. An insurance company which knows nothing about computers.

Sinclair n. (proper) The given name of the Creator, the Great God of computers. Now known to have been a false God in the light of the Revelations of Sugar.

Sinclair v. To advertise for sale before the availability of a product to assess market demand. To make money out of Scotch Mist.

Sinclair C5 n. An animated bathchair produced in error by the the Computer God as a new portable computer. Its main instruction was to go forth and multiply the money invested by the Gods, an act it manifestly failed to do. A more conventional design and provision of a keyboard might have proved more worthwhile.

source n. A list of readable instructions partially understood by the programmer which forms the basis of a bug-ridden program.

systems analyst n. One whose job it is to decide, wrongly, what your business is, and to translate those decisions in to a form bearing little relation to the original job definition.

terminal n. A television like device whose capabilities are usually slightly less or too slow to play the latest realtime version of Space Invaders.

write protect tab n. A small piece of opaque sticky tape normally fitted across a notch in the disk to ensure that a disk can not be written to when required or by omission, to ensure that a disk may be written to when not required. Standardization dictates that 8" disks work in the presence of the write protect tab, 5.25" ones fail when the tab is fitted.

Disk Formats and CP/M Disk Routines by M.W.T. Waters

[Ed. - this article runs to some twenty plus pages in total. I have therefore split it up, and this is the first part, concentrating on the format that data is actually written onto disks in.]

Have you ever wondered about how CP/M stores away its data, how the directory records relate to the files on disk, what determines the minimum and maximum file sizes or what determines disk size and the number of directory entries available?

There are several good books available about CP/M but none of them appear to cater for the dabbler in operating system software or for the type of 'hacker' best described as an 80-BUS user. The Digital Research handbooks contain all of the data required for a manufacturer to implement CP/M on a microcomputer but rarely explain WHY a particular disk parameter, say, is given a specific value or HOW it fits into the great scheme of things.

I should mention here that I was introduced to computers when hackers were (mainly) electronics enthusiasts who built computer systems from scratch and taught themselves programming by sheer hard work combined with more than a little trial and error. These days, hacking seems to apply to juveniles (generally) who illegally enter other peoples computer systems and create a bad name for home computer users.

Most of the information contained in this article is available to the average 80-BUS user who is armed with a disassembler, the CP/M manuals and lots of time and patience (or as someone else put it, "Stupidity and sheer bloody mindedness").

Disks and disk formats

Before proceeding with an in-depth breakdown of CP/M, I shall describe, briefly, floppy disks and floppy disk formats and then go on to examine the data physically written to the disk surface during formatting. The examples given will be oriented towards the current Gemini 5.25" disk formats but will be equally applicable to any IBM 3740 or IBM 34 type disk format (eg: Nascom).

A floppy disk consists of a disc of thin mylar (a flexible plastic) that has been coated on both sides with ferrous oxide; the same material that is used to coat magnetic recording tape. The disk surface itself is contained in and protected by a covering of some sort with cut-outs to allow access for the read/write head(s) of the disk drive. Disks are available in four sizes; 8", 5.25", 3.5" and 3" with 5.25" being the most commonly used by microcomputer manufacturers. In the case of 8" and 5.25" disks, the covering (known as the envelope) is made of cardboard while the other two sizes are protected by a rigid plastic case.

In use, the disk is rotated, usually, at a uniform speed (the Sirius microcomputer being an exception) and data is written to or read from the disk by one or two read/write heads similar to those used in tape recorders. Floppy disks may be either single or double sided. Disk drives that are designed for single sided operation only have one head while those designed for double sided operation have one head for each side of the disk. However, with early 3" disk drives, to access the second side, it is necessary to remove it from the drive and physically turn it over. For most disk operating systems (DOS's) this latter type of disk appears to be two separate disks joined back to back.

From this point on, I shall only refer to 5.25" disks but the principles involved are very similar to the other types. The disk surface is divided up into a number of tracks at the time of formatting; the number of tracks depending upon the disk size and the physical characteristics of the disk drive. The read/write head is permitted to access each track by stepping the head in or out under software control. The disk is given a reference point in terms of

rotation by an index hole which marks the start of all of the tracks on the disk. This index hole passes over an optical sensor, once every revolution of the disk, thus indicating to the disk drive that the head is about to find the beginning of the currently selected track.

Two standards have evolved for the number of tracks on a 5.25" disk. These are 48 tracks per inch (tpi) and 96 tpi although drives are now available which will achieve 192 tpi. The useful recording area of a 5.25" disk is just under a one inch band and so 40 or 80 track drives are usual although we all know of the Gemini and Superbrain 35 track 48 tpi formats, when the earliest drives had only that number of tracks. It is fairly obvious that in order to increase the number of tracks on a disk, the read/write head must write narrower strips to the disk.

Each track on the disk is divided up into a number of sectors. These sectors are generally one of 128, 256, 512 or 1024 bytes long depending upon the disk format chosen by the manufacturer. The Gemini SDDS format uses 128 byte sectors while the DDDS, QDSS and QDDS formats use 512 byte sectors. On most disk systems, the sectors are written to the disk by the format program and disks formatted in this way are known as soft sectored disks. Some systems (Apple for instance) use hard sectored disks where the sectors are physically marked on the disk by small holes similar to the index hole.

Consider, for a moment, the Gemini QDDS disk format. Disks in this format have two sides, 80 tracks per side with ten 512 byte sectors per track. The sides are numbered 0 and 1, the tracks are numbered 0 to 79 and the sectors are numbered 0 to 9. On some systems, the sector numbering starts at sector 1 so that the sectors would be numbered from 1 to 10. Some computer manufacturers start with other numbers but 0 and 1 are the most common values. This side, track and sector information is physically written to the disk during formatting so that the disk controller chip can identify the current side, track and sector by reading the disk. How this information is written to the disk will be looked at shortly but suffice to say that because the sector numbers are held on the disk as a prefix to the data held in those sectors, the disk controller can find a sector and read its data irrespective of the order of the sectors on the track. If the sectors on a track are not held in numerical order (ie: 0, 1, 2,..., 8, 9) then they are said to have been skewed.

Sector skew and its companion sector translation are used to improve access times when reading from or writing to a disk. Imagine that a disk has its sectors numbered sequentially from say 0 to 9 and that we wish to read sectors 0 and 1 in that order. Having found and read sector 0, there will probably be a delay while the processor is deciding that the next sector it wants is sector 1. Meanwhile the disk will still be turning and by the time that a request is sent to the disk controller to read sector 1, that sector will have probably passed under the disk head and the controller will have to wait until sector 1 comes around again. To overcome this problem, some manufacturers including Gemini allow the disks sectors to be physically skewed during formatting. If we look at a track on the disk, the sector numbers may look like this:

0 7 4 1 8 5 2 9 6 3

In the example given, if we now read sector 0, two sectors will be allowed to pass under the disk head before sector 1 comes round and the processor should now have plenty of time to make its mind up. Obviously, the amount of skew employed depends very much on the speed of the processor and too much skew is as bad as too little when it comes to slowing down disk access.

Sector translation uses a similar principle to sector skew except that it is a software measure to achieve the same result. With sector translation, a table of sector numbers is held in the computer memory. This table may look like that given below:

0 3 6 9 2 5 8 1 4 7

What happens now is that although the physical sectors will be in sequential order, the data is read from or written to the sector pointed to by the

translation table (known as logical sectors). In other words, using the table, when sector 0 is required we find that the data will be read from sector 0 but when sector 1 is required, we actually read from sector 3. If you compare the results of the skew table and the translation table you will see that they have an identical effect with one exception. A disk that has been written using sector translation must be read using the same translation table, otherwise rubbish will result as the sectors will be read in the wrong order. If, however, a skewed disk is read on a machine that would normally use a different skew factor (ie: perhaps the skew was set up for a 2MHz processor and we are reading on a 4MHz processor), the only penalty would be one of speed of access. It is worth noting perhaps that Gemini skew the system track of the disk differently from the data tracks. On the system track, the sectors are recorded in the order shown below while the sectors on the data tracks are skewed by 0, 1, 2 or 3 as chosen by the user. The example sector skew shown a little earlier had a skew factor of 2.

Example of sector skew on the system track:

0 2 4 6 8 1 3 5 7 9

Getting back to the physical disk format, let us now look at what is physically written to the disk during formatting. Most disk controllers format the disk by writing a complete track at a time. To do this, the host microcomputer will have assembled a memory image of the track which it will then transfer to the FDC (floppy disk controller) chip. The data consists of gaps, index and address marks, track, side and sector numbers, CRC bytes and of course the areas for saving the data.

Let us break the track up into its component parts and examine them in detail starting with the track, side and sector information and the area for the data.

Each disk sector is preceded by an identification block containing information about that sector. This block is six bytes in length and is identified to the FDC chip by an ID address mark immediately before the block. The data in the block is as follows:

1 byte - Track Number
1 byte - Side Number
1 byte - Sector Number
1 byte - Sector Length
2 bytes - CRC bytes

The track and sector numbers on the disk may lie in the range 0 to 255 although this will obviously be limited to the number of tracks that the disk drive is capable of accessing and the number of sectors that will fit on one track of the disk. The side number takes the value 0 or 1. The sector length byte (with the FD1797) may fall in the range 0 to 3 and the values correspond to sector sizes of 128, 256, 512 or 1024 bytes respectively. The two CRC bytes are automatically computed by the FDC chip and will be written to the disk when instructed to do so by the host computer.

When the FDC is instructed to read a sector, it first waits for an ID address mark and then reads the ID block. If the side, track and sector information matches the values given by the host computer, and if no CRC error has occurred, the FDC will transfer data from the data area following the ID block. The data area will be of the length indicated by the sector length byte and is preceded by a Data Address mark on the disk. Initially, the data area contains the value 0E5H for every byte. This value is set during formatting but in fact any value could have been used. 0E5H is used, by convention, because IBM used this value in their original floppy disk formats (back in the dim and distant past when 8" floppies were used with mainframe computers and micros had not yet been invented). Finally, the data area is terminated with 2 CRC bytes for error checking.

Writing a sector is similar to reading but after the correct ID block has been found, a data address mark is written to the disk followed by the number of data

bytes indicated by the sector length byte. Finally, the two CRC bytes are written to the disk followed by one byte of ones (0FFH).

In the Gemini DDDS, QDSS and QDDS formats, there are ten sectors per track and consequently the pattern of ID blocks and data areas will be repeated ten times on the track but with differing sector numbers.

Normally, the memory image of the track to be written during formatting would look something like that given below; the values given conform to the IBM System 34 format. When using the FD1797 FDC, the index mark is not required and so Gemini have left it out together with the pre-index gap (gap 4) and the 0F6H bytes. In the example memory images given, the centre column contains the data sent to the FDC while the right hand column shows the data written to the disk surface.

IBM System 34 Format

	Number of bytes	Hex value of byte sent to FDC	to disk	
	80	4E	4E	
	12	00	00	
	3	F6	C2	
	1	FC	FC	; Index mark
	50	4E	4E	
*	12	00	00	
*	3	F5	A1	; Resets CRC generator
*	1	FE	FE	; ID Address mark
*	1	Track No	Track No	
*	1	Side No	Side No	
*	1	Sector No	Sector No	
*	1	01	01	; Sector Length (256 bytes)
*	1	F7	CRC1	
*			CRC2	
*	22	4E	4E	
*	12	00	00	
*	3	F5	A1	; Resets CRC generator
*	1	FB	FB	; Data Address mark
*	256	E5	E5	; Empty Data area
*	1	F7	CRC1	
*			CRC2	
*	54	4E	4E	
**	598	4E	4E	

* Write this field 26 times.
** Continue writing 4EH until next index pulse received (Physical end of track).

By contrast, the track format for the Gemini QDDS format is given below:

Gemini QDDS Format

	Number of bytes	Hex value of byte sent to FDC	to disk	
	32	4E	4E	
*	12	00	00	
*	3	F5	A1	; Resets CRC generator
*	1	FE	FE	; ID Address mark
*	1	Track No	Track No	
*	1	Side No	Side No	

```
*      1          Sector No     Sector No
*      1            02            02       ; Sector Length (512 bytes)
*      1            F7           CRC1
*                                CRC2
*     22            4E            4E
*     12            00            00
*      3            F5            A1       ; Resets CRC generator
*      1            FB            FB       ; Data Address mark
*    512            E5            E5       ; Empty Data area
*      1            F7           CRC1
*                                CRC2
*     30            4E            4E

**   512            4E            4E
```

* Write this field 10 times.
** Continue writing 4E until next index pulse received (Physical end of track).

As can be seen from the examples, there are gaps containing zero and/or 4EH bytes between the ID blocks and data areas, before and after the index mark (if present) and at the end of the track. These gaps are there to allow the FDC to synchronize with the disk to read the data and ID blocks.

The gaps before the index mark and at the end of the track are known as Gap 4. The gap after the index mark and before the first ID block is known as Gap 1. Gap 2 separates each ID block from its associated data area while Gap 3 separates the data area from the next ID block on the disk.

The values of the bytes used to create the gaps are different for single density and double density modes as are the number of bytes used. Below is an extract from the manufacturers data sheet for the FD1797 showing the byte values and byte counts for the gaps. The values shown for the byte counts are minimum except where shown.

	Single Density	Double Density
Gap 1	16 bytes FF	32 bytes 4E
Gap 2	11 bytes FF	22 bytes 4E
*	6 bytes 00	12 bytes 00 3 bytes A1
Gap 3	10 bytes FF	24 bytes 4E
**	4 bytes 00	8 bytes 00 3 bytes A1
Gap 4	16 bytes FF	16 bytes 4E

* Byte counts must be exact.
** Byte counts are minimum except exactly 3 bytes of A1 must be written.

The physical data is held on the disk surface as a serial data stream. In double density mode a 250nS pulse is sent to the disk drive for each flux transition. This would imply that a pulse is sent to the drive each time the data changed from 0 to 1 or 1 to 0. In addition to the data, a clock is recorded on the disk surface. This clock is picked off when reading the disk to synchronize the data being read. The presence of a clock on the disk is used to good effect by the FD1797 when sending control bytes to the disk.

As seen in the example track formats, byte values of 0F5H and 0F6H generate the values 0A1H and 0C2H on the disk surface. However, so that these bytes can be distinguished from data bytes of the same values, clock pulses are deliberately

missed out by the FDC. In fact, when sending the 0A1H byte, there is no clock pulse sent between bits 4 and 5. Similarly, when sending the 0C2H byte, the clock pulse between bits 3 and 4 is missed.

In single density, various other clock pulses are missed when sending control bytes to the disk. In the table below, when sending a data byte, if all 8 clock pulses associated with that byte are present then the clock can be considered as having the value 0FFH. A missing clock in any position may be represented by a zero bit in the clock byte. The table also shows the values written to the disk for the values sent to the FDC.

Byte sent	Single Density	Double Density
00 to F4	Write 00 to F4 with Clk=FF	Write 00 to F4
F5	Not allowed	* Write A1, preset CRC
F6	Not allowed	** Write C2
F7	Generate 2 CRC bytes	Generate 2 CRC bytes
F8 to FB	Write F8 to FB, Clk=C7, preset CRC	Write F8 to FB
FC	Write FC with Clk=D7	Write FC
FD	Write FD with Clk=FF	Write FD
FE	Write FE, Clk=C7, preset CRC	Write FE
FF	Write FF with Clk=FF	Write FF

* Missing clock transition between bits 4 and 5.
** Missing clock transition between bits 3 and 4.

File sizes, disk sizes and directories

Having well and truly taken a disk to pieces, we can at last return to CP/M and the questions asked at the beginning of this article. (Can anyone remember what they were?) Well, you'll have to wait until the next episode.

A Look at MultiNet 2 by P.A. Greenhalgh

MultiNet Design Philosophy

The aim of Gemini's MultiNet networking system is to provide computing facilities to a number of people for the minimum possible cost. As a very significant proportion of the cost of any system is in the mass storage and hard copy devices, the overall cost of a multiple system installation can be dramatically reduced by allowing a number of users to share these facilities. 'Share' in this instance means the ability for any user to be able to make use of the mass storage device, but not, in general, the sharing of the stored data.

CP/M Compatibility

As Gemini MultiBoard systems are all capable of running the CP/M operating system, and given the amount of applications software available for that operating system, MultiNet is designed to provide a CP/M 'compatible' environment, the major difference being that the user need not have physical disk drives present at his Workstation, but the MultiNet software refers all disk requests to a Fileserver.

To achieve this, software has to be written that looks to the applications program as though it is CP/M, but in reality this software contains no disk driver or file handling routines, but instead refers these to the Fileserver that is controlling the mass storage. To achieve this the relevant CP/M documentation is used to write software that meets the given specifications as closely as possible, given the major premise that there are no physical drives present. Unfortunately, in practice, it is found that certain programs make use of certain 'quirks' or 'undocumented features' of CP/M, and so the emulation software has to be modified in order to provide as identical an environment as possible. It is thus extremely difficult, if not impossible, to achieve 100 per cent compatibility.

MultiNet 1 to MultiNet 2
The original MultiNet 1 software achieved an extremely high level of CP/M compatibility, but over the 2 years that it was available certain anomalies were brought to light. In addition a number of possible enhancements became apparent, and so MultiNet 2 was introduced to improve the CP/M compatibility of MultiNet and to provide additional facilities. It is available from Gemini dealers.

SERVER Changes

Auto-serving With MultiNet 2, the Server will automatically start-up network operation, unless within the first few seconds the user depresses a key on the Server, in which case it becomes available for maintenance purposes or stand-alone use.

64K system If Server load is deliberately aborted then the user is presented with a conventional Gemini 64K CP/M environment, as opposed to the special buffered system that the network software runs under.

Larger Winchesters The maximum possible number of MultiNet users is increased to 94 to allow the use of 3 logical drives (i.e. Winchesters up to 3 x 8MBytes = 24 MBytes).

Faster screen response All Server screen updates are now done using direct screen addressing to speed system operation.

Free space displayed The amount of free space on the Server drives may optionally be displayed on the Server display.

Spooling changes The Spool queue length (i.e. number of files awaiting printing) is always displayed and is updated whenever the length of the queue changes.

Additional Server features
An operator is now permitted to perform certain functions on the Server while it is in operation. These are all initiated by typing <ESC> followed by a Control character.

<ESC>	Ctrl B	Broadcast Message
<ESC>	Ctrl C	Initiate system closedown
<ESC>	Ctrl D	Display Directory space

Broadcast Message
The operator may enter a message of up to 50 characters to be broadcast to selected (or all) stations that are currently logged on. Once the message is set up, the operator is prompted to enter the numbers of the Stations that are to receive the Broadcast. If the character 'A' is encountered, it is understood as ALL, and all logged in Stations will receive the broadcast. An asterisk '*' is placed on the Server Logon display between the Station no. and User no. to indicate that the system is waiting to broadcast to that destination. At this time the screen display is altered to indicate that broadcast is in progress. When the broadcast has been accepted by an individual Station, the '*' on the Logon display is changed back to a '.' as normal. When all specified Stations have accepted the broadcast, the bottom portion of the screen is cleared and the Server is ready to accept further keyboard commands.

It is important to note that as the user stations cannot accept 'unsolicited' network messages, the broadcast can only be sent when the Station next sends a request to the Server. Thus if a Station is left logged on but unused, it will not receive the broadcast. This is the reason for putting the '*' into the Station logon display.

System Closedown
It is now possible to control the shutdown of the network. The following options are available:
1) - Shut-down immediately. In this instance all Server buffers are flushed, and Server operation is aborted. A confirmatory 'Y' is requested.

2) - Shut-down after last log-off. No new logon is allowed, and shut-down occurs when the last user logs-off. If a print spool file is in progress this will be aborted and spooling of that file will commence again, from the beginning, the next time the network is run.
3) - Shut-down after last log-off and end of spool file. No new log-on is allowed. Spooling continues. When the last user has logged off the file that is being spooled at that time is completed, and then Server operation is aborted.
4) - Shut-down after last log-off and last spool-file. No new log-on is allowed. Spooling continues. When the last user has logged off ALL spool files are printed, and then the Server operation is aborted.
5) - Ignore this shut-down request, and return to normal Server operation.

As soon as the first <ESC> Ctrl C is hit, all NEW logons will be refused, but NDOS Function 41 (User Logon) will be honoured to allow user area changes, providing the station is already logged on.

Once a Closedown option 2-4 has been selected, an automatic Broadcast is set up for all logged in Stations, informing them that the system is about to close. The absence of a '*' on the logon display indicates that the Station has received the broadcast. This automatic message may be re-sent by entering '<ESC> Ctrl C' to restore the Closedown menu, and then reselecting the required option.

Display Directory Space
This shows the number of directory entries free on the Server drives.

NDOS Calls
New function calls have been added for use from Workstations and Superstations:

Function 51 - Return Free Space on Server Drives
Function 52 - Return Count of Free Directory Entries on Server Drives
Function 53 - Return MultiNet Station Logon Vector.
Function 54 - Return Spool Queue Length
Function 55 - is Reserved for internal use by NDOS
Function 56 - is Reserved for internal use by NDOS
Function 57 - Return Current User Logon Number

NETWORK BROADCAST
Both Superstations and Workstations may receive Broadcast messages. These will appear in the top three lines of the screen display, and all operation will be halted until the message is acknowledged by pressing the <ESC> key twice.

PASSWORDS
When a user responds to a request to enter a Password, the entry that the user makes is no longer displayed on the screen.

New Utilities

SETBOOT
This is equivalent to the standard Gemini disk based CONFIG program, and it may be used to set various user options in the Workstation boot file, such as default LIST device, serial port parameters (baud rate, data bits etc), auto-execute file etc.

CU
This utility may be used to display various information on the current status of the network. Options include: Space remaining on Server drives. Number of directory entries available. Current users on the system. Number of files in the print spool queue. Current and original user number and station number.

MAIL
This utility allows messages or files to be sent by any user to a 'postbox' for receipt by another specified user. It also allows that user to receive any items that are in the 'postbox' addressed to him.

Letters to the Editor

MIRAGE PROJECT

Scorpio News readers may be interested in a current Mirage-driven project. This is called PPPP. It is not a sign of incontinence but stands for Pascal Program Porting Project and is aimed at those who have written marketable programs in extended Pascals (SVS, Turbo, MT+ or UCSD for example) and who would like a larger market by porting to Mirage via its excellent Swifte-Pascal compiler.

Straightforward code presents no porting problem to a multi-user environment and only the file-handling routines need to be re-written to use either Mirage's Locks Manager or the TRAP ISAM file processor. Ported programs may either be marketed by their authors or by Sahara Software together with Marston Spurrier.

Projects currently underway include the port of the Omnis 3 database manager from the IBM AT environment and two CAD packages. It would be very interesting to port some 80-BUS software which used the Pluto boards to the Challenger using the new range of 68K-BUS graphics boards. Anyway, I would consider most things and London-based programmers could use a terminal on my Gemini Challenger system for development work.

Nick Spurrier, Marston Spurrier, 10 Ransome's Dock, Parkgate Road, London SW11.

TINGE OF SADNESS

[Ed. - This letter was received by 80-BUS News after its imminent demise had been announced. It was passed to David Hunt to write a reply. At the time David was running the Computer Section of Henry's Radio, a London based Gemini and Nascom dealer. Both the original letter and reply are now published here.]

19th April 1986

Shirley
Solihull
West Midlands

Dear 80-BUS News,

With my final, one-issue subscription, there is a tinge of sadness about your demise - a sadness which also accompanied the end of the uP Nascom Newsletter. Looking back at the years between Nascom 1 and this funereal occasion, it is possible to state that all the 80-BUS adherents must have achieved a great leap forward in their knowledge and experience as the result of our contact, and that the nation as a whole must have benefited, both economically and educationally.

Where these benefits to the individual and the system have been greatest is in having encouraged exposure to machine code and electronics, with the result that many enthusiasts have gone on to develop these interests on other computers, from 'micros' right up to mainframes.

Nascom originally stood in the almost unique position in offering a fairly sophisticated and eminently communicable board to the unskilled public; the pity is that its original high cost never came down with the reduction in chip prices, which encouraged other board producers to 'hike-up' their prices, and also that the original board nor its add-ons ever advanced into boards organised through ULAs, or with the additional lines needed for extended memory addressing.

Having complained previously, in several publications about the urgent need for a simple board with 80 column display but otherwise software compatible (with the Nascom?? - Ed), I would like to say again (and obviously for the last time): our [Nascom customers'] investment in Nascom and Gemini products now stands at somewhere around £4,000,000 (original boards, add-ons, disk drives, software and firmware); it is a tragedy that much of this investment will go to

waste. A new board is still a viable proposition, provided that it is a bare board, and given some cooperation from existing copyright holders to provide (or give permission for) software modifications which allow existing users to carry forward most of their existing software and firmware investment. How about it Gemini and Lucas?

My final appeal is that the final issues of 80-BUS News should be spent on as much hardware/software information as can be contained between their covers, since this is the last opportunity, leaving no spare room for humorous anecdotes - the world's a funny enough place already.

Yours faithfully, Bert Martin.

P.S. How about asking all subscribers if they would like a full list of names and addresses?

DAVID HUNT'S REPLY

May 1986

Dear Mr. Martin,

Thank you for your letter, it's been passed to me by Paul, probably because he thinks I may be able to give a more objective answer, drawing on my wider experience of the home computer business, covering scenes from behind the shop counter, playing at design engineer (what I was originally trained as) and as supporter of the home computer cause. I have not confined my discussion of your letter to the above but have also reread your letter published in 80-BUS vol 3. iss 3. page 6.

Overall, I feel in general agreement with your first two paragraphs, as I for one have most certainly benefited from the original conception of the Nascom 1, through the Nascom 2 and then through the ever growing number of Gemini components. As these machines have grown in power and complexity, so I have learned to come to terms with them, both in design and software implications. This, to me, has been something of an uphill struggle and has involved the personal investment of many thousands of hours, not to say money in acquiring the machines in the first place. Ok, I did end up with a free Nascom 2 from the Company, but most of the rest has been purchased over the years at only a little more advantageous price than that paid by most other people.

This investment in time and money is at last paying off, as I shall be starting a new job in June which I can honestly say is directly attributable to the knowledge gained by close and diligent learning from these machines.

In 1977 I knew nothing about computers, they only taught me a smattering about digital logic when I gained my HNC in the mid-sixties, and most of this study was more applicable to building power stations than computers. Nascoms were the start, followed by Geminis, and now (dare I say it), IBMs, Vax's and 68000 machines like the Gemini Challenger and Prime. It has to be recognised that in the field of computing I am almost totally self taught, with a little (but invaluable) help from friends I have made along the way. By the time technical colleges woke up to the fact that people wanted to learn about micro-computers, I was sufficiently ahead of the field to be approached from the point of view of being a night class tutor rather than a student. (I have turned those offers down and continue to teach the RAE at Paddington College, an altogether more sane occupation.)

So, yes I can agree that the early (and more recent machines) have been of considerable help to me, and I'm sure to many others. I feel it's a bit strong to claim these benefits on behalf of the country as a whole, but to me, yes, there have been benefits.

I'm sorry, Mr. Martin, but it's about here that our opinions start to differ. I don't know your profession, but I would guess it's not involved with either electronic manufacture or retailing.

Agreed, the Nascom was unique, being in advance of its time and aimed at a totally (in this country) untried market. Its virtues were simplicity (relative term) and the fact it was supplied as a kit. But was not its success due to more to its uniqueness in the marketplace rather than the fact that it was communicable and supplied as a kit? In the beginning there was Nascom and no other choice except either a much simpler machine (the original the Cambridge ??? (you know, the thing before the ZX80)) [Ed. MK14??] or the machines being published as serials in the electronics mags at the time which were almost totally diy.

The original high cost was brought about by the need to amortize the original development cost and not the cost of the chips employed at the time. The fall in chip prices was reflected by a price reduction in the Nascom 1 a year or so after its introduction. Also I can think of no examples of board manufacturers 'hiking up' prices in line with the Nascom. All manufacturers are faced with the same problem, the need to recoup the cost of a design. The material cost of the finished product usually has little relation to the final selling price. Most other computers started life priced fairly highly and prices fell as the costs were recouped (ignoring the dumping of product which did not sell at all).

Next, Sir, you imply the lack of progressive design and by so doing contradict one of the main virtues of the Nascom. One of the main reasons that the Nascom was so adaptable and of such value in (self) education was the very fact that design was NOT stuffed into a ULA. Apart from the fact that ULAs were only just becoming available at the time the Nascom 1 was designed. ULAs are marvelous things from a mass production point of view. But you can hardly get inside one with a soldering iron and bend the original designers' ideas towards your own. Secondly, ULAs are extremely costly to develop and to gain benefit from their use, a market must exist in hundreds of thousands or even millions. The Nascom sold in thousands, but never enough to make the use of ULAs viable.

Gemini have compromised a bit, they tend towards specialized PROMs and PALs in their designs. This saves a lot of chippery and by so doing simplifies board layout. Again Gemini boards sell in quantity, but not enough to make ULAs viable. Also, Gemini I think must know that they are a specialized market, and even the introduction of PROMs places restraints upon the flexibility of the boards. I wonder if you have looked inside an Amstrad PCW8256, it's only got half a dozen chips in it (excluding eight RAMs) all centered around one postage stamp sized 99 legged beast in the centre. Marvelous from a manufacturing point of view, it even keeps the accountants happy. But you've got to aim at making 1,000,000 of the machines to make it viable, and heaven help anyone who wants to personalize an Amstrad!

And so to extended addressing. There is a craze at the moment to endow the Z80 and the 6502 with 128K, 256K, 512K or even a 1M of addressing capability. But what can you do with it? No application software for micros is ever written that big, and no individual in a Nascom/Gemini context is ever going to write anything bigger than 60K because the market is too small and even smaller for the few Geminis fitted with 256K RAM cards actually used as RAM. It's the numbers game! I've got something bigger than you - never mind what you do with it! The only sensible thing to do with this amount of RAM is to use it a virtual disk. Gemini have done both with the GM813 with its extended addressing and their 256K RAM card and the 512K silicon disk. But I often wonder if it was worth the effort. Don't forget, to use this RAM, you must have application software to run in it and loading 128K from tape will take all day, and disk software has to be applicable to machines which don't necessarily have this address range, so what's the point? Ok my machine has a 512K virtual disk but I hardly ever use it, and something like the Amstrad only uses it as virtual disk because it's only got one drive and it was cheaper than fitting a second.

No, I'm sorry Mr. Martin, but I don't agree that the development of the designs was hampered by the lack of use of ULAs or that prices were particularly high, also extended addressing is available on more recent cards - for what it's worth. You forget that the Nascom was a specialist machine manufactured in relatively small volume for a very special market. Its initial success was probably not due to its design but because it was available. Its decline can be directly related to the advent of the poorly communicable, un-tailorable and boring plastic boxes (Sinclair, Acorn, Dragon, et al) which did little for the appreciation of the hardware but were in the main originally intended to sell very clever games software to an unsophisticated market. These machines are all less tractable than the Nascom and Gemini, but then, they are manufactured to satisfy a superficial need or whim (if you will) for gimickry in our present society and not really to engender an understanding of what actually makes them tick. I wouldn't mind betting that 90% of home computers sold over the last four years are now collecting dust in some forgotten corner as the owners have no idea what to do with them apart from playing Space Invaders.

And so on to the MOCSAN, your plea for an unpopulated board for a single board computer to use the original peripherals for Nascom and Gemini and if possible to use the original chip components. 80 column display but otherwise software compatible with the Nascom.

Firstly you speak as if, with the demise of the 80-BUS magazine, all components and future expansion for the Nascom and Gemini through the 80-BUS, will cease to exist. THIS IS MOST DEFINITELY NOT THE CASE. I know full well that you have not attended Gemini dealer meetings, where various people, most notably John Marshall, the MD, have repeated ad nauseam that there is no intention of phasing out the 80-BUS components until well into the nineteen nineties, although work on future products, most notably the development of the 68000 based Challenger series will reduce the design effort devoted to 80-BUS. In fact, it has been hinted, there are some surprising developments forthcoming in 80-BUS products in the near future. (I have no idea what these are, as I consider that all that needs to be available for 80-BUS is already available.)

I would argue that a new board is probably not viable as my close association with the current market suggests entirely contrary conclusions to those you cite for the viability of the board.

Firstly, the kit market (for anything, be it computers or single transistor audio pre-amps) is rapidly declining. It is now cheaper to buy a fully working, built item, than to build it yourself. Ok, the built item may not be entirely what you want, but the electronic buying public as a whole are not prepared to pay the money for a kit which does exactly what they want. I know, because my Company and many others have tried to revitalize this market, and it's just not there. Go and ask any retail components supplier. This decline in the kit market obviously affects the price of discreet components, forcing the price spiral upwards whilst depressing the market it is supposed to generate.

Next to the need for a single board computer itself, there have been several attempts to introduce new single board computers on the home market. Some even fulfilling the major parts of your specifications, and some very reasonably priced. Do you remember the 'Big Board' or the Multitech MPF-1? No, the demand for a board computer has declined as the majority no longer want a single board computer without a box. This leaves the board market to the specialist manufacturers who supply to a special market. That market is no longer the home user, it is more likely to be system designers making modest quantities of dedicated machines for special purposes, things like credit card embossing machines, automatic packaging machinery, lamp bulb making machines, etc. The large majority of boards now go in that direction.

Let's look at the profitability of producing a bare board to a new design. The costs are enormous. The board is unlikely to attract buyers at more than £20.00, yet to design a new machine to supply a potential market of a few hundred such boards, the individual board cost is more likely to be three times that. The £20.00 would only just cover the manufacture of the board yet alone the cost of development, drafting, board manufacture set up costs before you

start. And who is going to pay for component procurement, and a 'get you going' service after it's built? No there's nothing in it for a potential manufacturer - believe me, if there were, they'd have done it long ago.

So the final answer is to 'do it oneself', where the cost of the time taken to achieve an end is not considered as a material part of the costs. If I considered the cost of the time which I have spent in designing one off projects for my own use I would never have started them in the first place. Certainly if someone offered me one of my one off projects at a realistic price I would have certainly done without and not bought it.

I don't consider myself as an unusually mercenary individual but I do like some (token) reward for my efforts, and in that I don't consider myself unusual. Many things are undertaken for the pleasure in doing them (I don't write for this mag for the money, I'll never get rich that way) and the rewards need not be monetary. But I personally think that your MOCSAN appears to represent a very considerable amount of hard work and if it is to contain ULAs, then also prohibitively expensive, certainly not something to be undertaken for fun. If it isn't fun, then someone (or many) will expect to get paid, and the projected price will go out of the window. No sorry Sir, it's a nice idea, but to me, it's not on. Perhaps I am a minority, I'd welcome other views.

D. R. Hunt.
(Reply written May 1986, re-edited Dec 1986)

A Review of the 2MB Upgrade for the GM833 by D. Greenfield

The Gemini GM833 Virtual disk board has proved very useful for over two years. When this board was first used, it seemed that it would never become full. Could it be that software is more sophisticated, or is it just less efficient ? Anyhow, "Disk Full" or "Write Error Drive M" started to rear its ugly head.

It seems that a virtual disk drive must have at least the capacity of a floppy disc on the system. Preferably it should have a lot more, at least 50% more as a lot of temporary files are produced during working.

The PBM upgrade for the Gemini GM833 board was fitted to the GM833 in about an hour. This upgrades the GM833 to 2MBytes, 4 times its original capacity. The Kit comes as a small pcb, wire, 41256 memory ICs, instructions and double sided sticky tape to hold the small Pcb to the main board. Everything went well on the modification, the instructions were reasonable, although an actual photo of the board would make component location easier. No tracks are cut, thank goodness. 8 wires are used to connect the pcb to the main board at various points. Pin 1 of all the memory ICs were connected together. This provides the extra address line required.

Everything seemed fine, on startup, the M> prompt displayed as usual. Running "STAT" displayed "2024K Bytes Free". However, on loading more than 512K, the system failed. It seemed that the same 512K was "reflected" 4 times on the board. A phone call to PBM systems threw light on the problem, two of the solder pins had bridged with solder under the board. Removing this fault resulted in correct operation. The board has been running correctly since.

To conclude, the board represents good value. With care of soldering, the mod is easy to implement.

Doctor Dark's Diary - Episode 24

Just as I was about to start my own Nascom fanzine, duplicated badly, and printed on grotty paper, the leaflet advertising Scorpio News arrived. I was delighted to see it, and posted my subscription the same afternoon. After all, I suspect publishing is jolly hard work, and I bet this gets printed on decent quality paper as well. Besides, I would have needed the address list of the 80-BUS subscribers, and something to print...

THE STORY SO FAR

For the benefit of the many who have never heard of me, and have not read the previous 23 episodes of this column, now appearing in its third magazine, I am an enthusiastic amateur computer owner. I am enthusiastic because the machines we all have in common are so good, compared with the other stuff on the market. And I am an amateur because there don't seem to be any computer firms with the sense to offer me a job in this neck of the woods.

I started out with a Nascom 1 kit, which amazed me by working after I had built it. I learned to program from the Nascom 1 software manual, which is no mean feat, as this document was designed to prevent people from wanting to use their Nascom 1s. By the time I had finished my machine code biorhythm program, and typed in the hex using the monitor program, I was hopelessly hooked. As the various expensive new boards appeared, I expanded the machine, changing to a Nascom 2 when the Nascom 1 refused to run at 4MHz (I know, lots do. Mine wouldn't.) The system this is being written on consists of the following: Nascom 2, Nascom 8 amp power supply, Gemini G809 disc drive controller, two Pertec FD250 drives, a Pluto graphics board, a Gemini SVC, two MAP-80 256K RAM boards, a Gemini GM870 modem board, a Belectra floating point board, and a Nascom IO board. There are three screens, one for the Pluto, one for the SVC, and one for the Nascom 2 screen.

Languages in use are Z80 assembler whenever I want things to go particularly fast, BASIC when they can go slowly, and nobody is going to know, Pascal when I am writing something large, with fancy data structures, and C when I am in the mood for some suffering and can stand the ugliness of it. For instance, my current project, some Prestel software, is being written in assembler, in the hope that it will go fast enough not to need fancy queues of data or horrid interrupt driven software. I was originally going to write it in Hisoft Pascal, but changed my mind because I thought it might not go quite fast enough, and because I would have had to plan everything properly. I decided against C for the simple reason that I don't feel I know it fully yet. (Has anyone seen a book that explains anonymous data types properly?) The reason I didn't us BASIC, is that it would never have run properly...

Now all that must sound (I hope) very competent and educated. Fooled you! We *sixth form dropouts sometimes recover from thinking we can not learn new tricks*, you see. In 1978 I knew none of these things, but am now half way to an Open University degree, mostly in maths. One of these days, with the help of the escape committee, I hope to escape from my boring job and have some fun doing something interesting with micros. And that is quite enough about me. Now read on...

HARDWARE UNPLEASANTNESS STRIKES!

As is well known, the Gemini GM870 MODEM board and the Belectra HSA-88B floating point board both use port addresses £80 and £81, so I can not use them both at once. In theory, it is possible to move a link on the HSA-88B, and have it work as other ports, but (again) mine won't. I am planning to build some sort of decoder to select between the two boards, if there is no other cure, but not unless I have to.

The main reason I described interrupt routines as unpleasant above is that the last one I wrote, several times, refuses to work. It tried to use the CTC chip on my Nascom I/O board to interrupt the system regularly, in order to read the MODEM's input port, and queue up any input. However, the program just will not

run, even though similar programs written when the system was much smaller worked well. Even more interestingly (I probably mean disgustingly), the system refuses to work at all without the I/O board. I suspect that either the motherboard is just too long for all these boards, or that there is a fault on the motherboard affecting the I/O control signals or the address bus. I meant to replace the mother board with a decent one ages ago, as it is just Vero board, but never got round to it due to the cost of the good ones. Any dealers got a Microcode 14 slot board left? [Ed. - see the Newburn advert. They have now taken over production of this board.]

I have read and re-read all the various manuals, and am fairly sure all the boards are set up correctly with regard to 80-BUS signals such as NASIO and DBDR, which is an exercise I recommend for anyone who wants to test their patience and comprehension. My decision, until such time as I decide to pull the system to bits for maintenance, is to ignore all these problems, and program round them. Hence the need for assembly language speed in the Prestel software, in order that all the routines will be completed before incoming data has been lost. And if the Pluto board keeps the system waiting, I suppose I will just have to get the double speed processor option fitted, even if it is a silly price.

HISOFT C, DOES IT HURT?

No, it doesn't hurt your pocket, just your head. For very little money you get a good manual, a good screen editor and a good compiler. But the language itself rivals Forth for unreadability. (I know, you can write Forth so that other people can read it, but who actually does?)

The manual contains a quite helpful tutorial section, and the editor is excellent for programming use, as it has all the handy facilities like auto indentation. It will not do justification though, or word-wrap, like your favourite word processor. The really clever bit comes when the compiler finds a mistake in your C program. It re-enters the editor at or near the error, for you to fix it. Exit from the editor, and the compiler has another try. I like this, it saves a lot of the time usually spent giving commands to the compiler and editor on other systems! (I believe the newest version of the Hisoft Pascal compiler also does this, and will let you know when I get mine updated.)

The version of C implemented has no floating point numbers at all, just integers, which is a pity. It is a surprise too, given the good floating point support in the Pascal compiler. I just hope they will be doing a version for the Belectra board, as well as an ordinary Z80 version, when they do add floats. The system as supplied, and the programs it compiles, certainly work, and I am finding them helpful in my attempts to learn yet another language, but it is not a pretty language!

PRESTEL ON PLUTO

As mentioned above, I am writing a Prestel program to put a display on the Pluto board, instead of using the useful "Pretzel II" I bought with the MODEM which uses the SVC (or an IVC) for its display. When finished, this will have quite a few fancy facilities, which would normally make it well worth trying to market the program. However, how many people have compatible hardware? If either of them is interested in a copy, they can contact me to arrange a swap for some Pluto program of their own writing. Anything pretty, like Mandelbrot set stuff, or flight simulators, would be most acceptable...

STOP BIT

Correspondence about any of the above can be sent either to the Editor, if you want him to do something about it, or to me, Chris Blackmore, at 27 Laburnum Street, Taunton, Somerset, TA1 1LB. Sometimes I fail to answer letters. Nobody is perfect. Although I do have a telephone, I actually dislike using the things to speak to people, and would prefer not to be telephoned unless it is to offer me an amazing free gift or something!

A Beginners' Guide to 1/2" Magnetic Tapes by Timeclaim

[Ed. - As well as using their Gemini Multi-Format (M-F-B) Systems for transferring files between many different disk sizes and formats, a growing number of users are also using them for transferring data to and from 1/2" magnetic tapes. This article is an extract from the manual for Timeclaim's 1/2" magnetic tape sub-system for the M-F-B and describes some of the basics of 1/2" tapes. Our thanks to Timeclaim for allowing us to reproduce this documentation, which is copyright (c) Timeclaim.]

Introduction

1/2" Computer tape comes on plastic reels which are usually kept in a plastic box or with a strip of plastic round the edge for protection. The diameters of the reel are 7" (holding either 300' or 600' of tape), 10.5" (holding 1200' of tape) or 12" (holding 2400' of tape). The centres of these reels are the same so that they all fit the same tape drives. (History will reveal some exceptions but these are almost never seen today).

Tracks

Tapes are said to have 9 tracks because the recording head consists of 9 sections one above the other. All tracks are recorded at the same time unlike tape cartridges which usually are recorded one track at a time. There are some old 7 track tape machines about but these are rarely encountered.

Density

Progress has meant that the density of data recorded on tape has been increased over the years. The densities seen nowadays are as follows:

800 bpi	NRZI	(Non Return to Zero Inverted)
1600 bpi	PE	(Phase Encoded)
3200 bpi	PE	" "
6250 bpi	GCR	(Group Coded Recording)

The density refers to the number of bits per inch on each track. As one complete byte is written at a time (with the 9 head sections) the number of bytes per inch is equal to the number of bits per inch. In the above table it will be seen that there are some further letters. These describe the method by which data is recorded on the tape. This information really needs to be known only by the tape deck engineers.

800 bpi tape are almost obsolete in the data processing world where there is a need to store a great deal of data on a tape. However, many CAD/Engineering/Scientific users still use 800 bpi as they do not usually need to store so much data. Often they will only use the first few feet of tape on a 7" reel.

1600 bpi has become the data exchange standard. Most tape decks can read and write at 1600 bpi so this is how data is usually transported from one system to another.

3200 bpi. This is a double density version of 1600 bpi. This density is usually quite a cheap add on for 1600 bpi tape decks but is almost never used for data interchange. Systems using this are fairly unusual.

6250 bpi. This is the latest development in 1/2" open reel tape and is coming into use at many mainframe computer sites. Most 6250 bpi drives will also read and write at 1600 bpi. 6250 bpi drives are much more expensive than 1600 bpi and are not available in some countries.

Data Layout on 1/2" Tape

Data is written onto tape in one complete 8 bit byte at a time. The 9th track is used for a parity bit which is dealt with automatically and is not seen by the programmer. A number of bytes on a tape are grouped together into a block. Blocks are separated by gaps in which no data is recorded. These gaps are from 0.6" for Phase Encoded tapes to 0.75" for NRZI tapes. Blocks can be any length but blocks shorter than 11 to 14 bytes are not allowed. The actual minimum

block size depends on the individual computer system. The maximum length of a block is limited only by the availability of memory to hold the block in the computer. To identify a block on Phase Encoded and Group Coded Recording there is a preamble before the block and a postamble after the block. These are recorded so that the machine knows where a block starts and where a block finishes. Blocks are not numbered nor is there any record of their length. The system works out the length of a block by counting the number of bytes read between the preamble and the postamble.

BOT/EOT

The beginning and end of the data area of the tape are identified by a piece of reflective foil stuck to the tape. This is seen by photoelectric sensors when the marks pass. They are known as the BOT (Beginning Of Tape) and the EOT (End Of Tape) markers. When the tape is first loaded (on the tape drives) the tape is automatically moved so that the BOT mark is in front of the photoelectric sensor. The first data block is located a short distance after the BOT mark. Blocks are written one after the other separated by gaps. The purpose of the gap is to separate blocks so that the blocks may be used one at a time.

File Marks or Tape Marks

In order to divide the tape into files a special type of block is recorded - this is called a file mark or tape mark. The file mark does not contain any useful information. It is just a place marker. When a file mark is read a special signal is sent from the tape deck to the host computer. This usually causes reading to cease. The file mark is used to separate files on a tape. A further use of the file mark is at the end of the data on a tape. Usually the end of the recorded data on a tape is well before the reflective EOT marker. To signify that there is no more valid data on the tape two or more file mark blocks are written next to each other. If one file mark is encountered and the system is asked to read the next file and another file mark is immediately found, with no data block being read, the system knows that it has reached the end of valid data. Some systems write several file marks at the end of data for good measure.

Adding Files to a Tape

When a file is to be added to a tape, the tape is positioned to the file mark just after the last file. When recording restarts all of the extra file marks are overwritten. At the end of the new file multiple file marks are written.

Headers, Volume Labels and Trailers

As can be seen from the above, there is no directory system on a tape. It merely consists of files recorded one after the other. This is quite satisfactory for some applications when it is sufficient to stick a paper label on the tape reel to identify the contents. Most scientific and engineering applications find this adequate.

In commercial data processing departments where many more tapes are handled and they are kept in big libraries a more formal scheme is required. A system of Headers and Trailers has been developed so that each file in fact consists of 3 tape files. The first is the Header, the second the data file itself, and the third the trailer or "End Of File header".

Each reel of tape is known as a Volume - just as with books. To identify the volume the first header file on the tape has an additional record in the first block and this is known as the Volume Label or Volume Header.

Another type of label that is encountered is the End of Volume. This is used when it is necessary for a file to continue on to another volume of tape. Instead of an end of file header at the end of data, the end of volume label is used to signify that the file carries on to another tape volume.

The exact content of these headers varies from system to system. Standards have been produced by ANSI, ECMA, ISO and IBM. These are all similar in principle but differ in detail. It will be found for instance that IBM use the EBCDIC character representation instead of the ASCII used by the others.

Review of the MAP-80 Video/Floppy Controller by P.D. Coker

Over the past few years I have been using an 80-BUS system in which the MAP VFC card provides the video and floppy disk control normally provided by the Gemini IVC/SVC and FDC cards. The card was one of the original versions and was supplied as a kit which was well documented and presented, and worked satisfactorily after a simple setting up of the FDC with an oscilloscope. The VFC is supplied as a kit or ready-built, either as a video controller or floppy disk controller only, or with both facilities implemented; a keyboard interface and video switch are optional for the video-only version but are included in the complete VFC.

WHAT IT CONSISTS OF

The VFC card can be used in 3 ways; as a combined video and floppy disk controller, as a video controller only or as a floppy disk controller (for 5.25" or smaller drives only).

The video section provides a memory-mapped 80 column by 25 row display with a highly readable alphanumeric character set; there is a facility to switch an external video signal into the same monitor as the VFC, and the video output level can be regulated with an on-board potentiometer. A 7 or 8 bit ASCII keyboard can be accommodated by the VFC and there are both 'normal' and 'inverted' character sets, half of which are left for expansion (for example, as a games set) but still leaving 512 alphanumeric and other characters for immediate use.

The floppy disk controller will cope with any make of drive, apart from 8". Some criticism has been levelled at MAP Systems for this, but few users will find it a limitation. Also included is 4k of RAM, 2K of which is used for the display and the rest by the operating software. It uses a total of 16 ports which are link-selectable, enabling the user to choose any group of adjacent ports from C0 to EF; the default port allocations are from E0 to EF but alternatives are suggested.

The board contains a standard 6845 CRT controller and a 2797 FDC; unlike the Gemini IVC/SVC, there is no on-board Z80 and the 2797 is a more sophisticated version of the 1797 disk controller used on the Gemini GM829 FDC/SASI board, and is from the same family as the 2793 disk controller used on the more recent Gemini GM849 FDC/SCSI controller. All this is contained on a single 8" square board which plugs directly into the usual 80-BUS connector. Some care is needed if the kit is constructed and an oscilloscope is needed to set up the 2797 - MAP will do this for you for a small fee, but the construction manual is very good and the average person would find little difficulty in producing a working version.

There is a significant design difference between the two types of video controller; the Gemini IVC and SVC have a Z80 incorporated on the board (Z80A in the case of the IVC and a Z80B which runs at 6MHz in the SVC.) Effectively, these are single-board, highly specialized micro-computers, whereas the VFC has no microprocessor included and thus forgoes the benefits of parallel-processing. For many applications this will not be a great disadvantage since the VFC software is quite sophisticated and carries out the task of interfacing between the VFC and CPU very efficiently so that most users would not be aware of any difference in the performance of the VFC compared with the IVC. The SVC, running at 6MHz, with faster screen handling, would appear to be different, as Dave Hunt's brief test (80-BUS News vol 3 issue 1) shows! The SVC has a lot more on-board memory as well and a generally very impressive performance. It also uses a high quality version of the 6845 (its performance is degraded if the standard version is used).

DOCUMENTATION

This is adequate and consists of a detailed explanation of the port allocations and their functions, program examples for the insertion of support for the VFC into a CP/M BIOS or interfacing the VFC with NAS-SYS, and details of the control codes which the VFC recognizes. In view of the large number of possible combinations of VFC and Nascom 2/Gemini/MAP CPU or FDC cards, the implementation notes are quite extensive. They are also easy to follow which makes a pleasant change from some systems; unfortunately, no details are included for the Nascom 1 and I gather that there are difficulties in using the VFC with it. In the full version, 15 or so links are involved but not all of these have to be used. Details are also given of modifications which need to be carried out if RAM boards (Gemini and Nascom) are to be used. No modifications are needed if the MAP RAM card is used. A final section gives details of pin assignments on the VFC and a useful note on TEAC, Pertec and Micropolis drive pin assignments.

If the fully implemented VFC or the FDC-only system is purchased, MAP-80 Systems can supply a fully licensed version of CP/M 2.2 or 3.0 with a customised BIOS. This is supplied with a short instruction manual which explains how linkages are made or modified on the various types of CPU cards, gives details of disk error messages as well as how to customise the BIOS for other features. Both parallel and serial printer output and screen editing using ^@ are supported. A special boot EPROM is supplied which replaces the RP/M boot EPROM if the CPU card is a GM813.

In addition to the usual CP/M files on the disk, and the customised MOVCPM.COM or CPM3.SYS, a multi-utility program is supplied, which allows users to format, verify or copy disks. MAP will also customise existing users' CP/M on request at a reasonable cost.

THE VFC IN USE

The fully implemented card has two 3 pin connectors for the video input and output, which are incompatible with the Gemini GM812 IVC video output which is a standard jack socket requiring an enormous plug. Why Gemini use this gargantuan means of taking out the video puzzles me since Nascom found some small coaxial connectors for their AVC which are really neat. I'm not all that keen on MAP-80's connector either and I'm sure that there must be all sorts of impedance mismatches since neither type is likely to approach 75 ohms or whatever a monitor is supposed to need to produce a good display. The keyboard and disk drives are plugged in to the board - using non-latched ID connectors which are a little difficult to unplug when needed.

On booting up, a system message is displayed together with the CP/M prompt; commands may then be entered. The display is 80 columns wide and because of the alternative character set, both normal or inverse (black on white) characters can be displayed, or an alternative character set which can be programmed into a 2716 or 2732 EPROM and plugged into the second ROM socket.

The memory-mapped display appears to be quite fast and free from visible interference as a consequence of CPU and CRT controller interaction; as has already been mentioned, the Gemini video controllers have Z80 CPUs on board and under certain circumstances on the IVC some interference can take place which produces a faint diagonal patterning on the screen - which can be irritating under some screen display conditions. The cause possibly lies in the layout of the IVC board where one or two long-ish tracks radiated noise which was picked up by other parts of the system. This problem appears to have been resolved with the Gemini SVC which has a much more legible screen display than its predecessor.

I experienced no problems with the floppy disk controller when using Pertec (48 TPI) and TEAC, Canon or Mitsubishi (96 TPI) drives - which are plug-compatible. The Mitsubishi drives were quietest in operation. Some problems were experienced when using 96 TPI Micropolis drives - I'm not sure why - perhaps they objected to the fast track stepping rate which suits the TEAC and

compatible drives. The TEAC drives were rather noisy in operation, compared with the other 96 TPI drives. They are even noisier when used with either of Gemini's FDCs. [Ed. - this is probably so with the early Gemini GM809 FDC board, but with the GM829 and GM849 boards the drives may be stepped at their maximum rate (3mS). (With the VFC and GM809 only 6mS is possible, which results in the noise.) The GM829 and GM849 give virtually silent operation with Teacs. You must either have the software set up incorrectly, or not have the correct software (Gemini BIOS 3.2 or later).]

Unfortunately, some software is not completely compatible with the VFC - which is a nuisance; when I tried to use an IVC version of DISKPEN, I found this out the hard way and had to resort to WordStar - which is (initially) a lot less easy to use. A Sargon chess program with rather nice graphics which worked on the IVC (with a nasty frame wobble) failed to display any graphics at all on the VFC - all one had was a record of the moves!

A version of DISKPEN is available now (not surprisingly called MAP PEN) which gets over one of my problems and some software hacking might possibly overcome the other problem - but it probably has something to do with the switching of the IVC into its 48 character Nascom-lookalike mode which the VFC cannot do. I suspect that the Gemini SVC would also not like my version of Sargon either since it now has a 40 by 25 display option. I don't play chess very often and when I do, the machine beats me so the modifications needed are right at the bottom of my list of priorities. Like most of the 99% of video controller users mentioned by Dave Hunt in his article, I probably don't use the undoubted potential of either the VFC or the SVC to their full advantage since most of my work is concerned with number-crunching and word-processing, rather than graphics. The instructions given are probably as comprehensible as those supplied with the IVC and SVC, thus implementation would pose no problem for the addict.

CONCLUSIONS

As far as most software is concerned, the use of the VFC poses no problems. In my view, the VFC represents a good, reasonably low-cost attempt to bring the undoubted advantages of disk operation to Nascom 2 owners; for someone starting up with CP/M, the purchase of a Gemini GM813 or MAP CPU card and the VFC represents a considerable cost saving on the minimum 'all Gemini' system where 3 boards are required. Some useful features available with Gemini's current video and floppy disk controllers are not found on the VFC but both the extent of one's purse and the type of application may well be the deciding factors.

The ready-built full version costs £214 + VAT: other versions are about half this price. In kit form, the full version is £175 + VAT and would take a couple of evenings to build; the video-only and floppy-only versions cost £99 + VAT. MAP 80 Systems are now in Egham (0784 37674).

Incidentally, if you are contemplating purchasing disk drives, it is possible to buy TEAC or Mitsubishi drives which are advertised for the BBC Micro by a number of firms. You need the double sided 80 track versions with 400k (BBC mode) capacity which will give a formatted capacity of 784k on an 80-BUS system. Several suppliers offer good bargains - borrow a Beeb-orientated magazine and check the ads.

Private Advert

FOR SALE
Due to upgrade we have the following for sale. Galaxy 1 with keyboard and screen. Also Galaxy 4 Fileserver 10MB with keyboard and screen. Will throw in accounts package, Wordstar Multiplan plus Mailmerge F.O.C. Open to all reasonable offers. RING 0522-38525 (DAY) 0522-751769 (EVENINGS)

Putting on the Style by P.D. Coker

One of the more grisly aspects of computing is to read some of the glossy magazines sold for the profit of the publishers and the edification(!) of the users of plastic box computers. Apart from the ads., the program listings are often so dreadfully or densely printed in minuscule type that it is a major business trying either to read them or (worse) type them in. Multi-statement lines up to 250 or so characters in length do little for one's understanding of program logic!

Quite a number of commercially published programs appear to have been written in such a way that only the most dedicated 'hacker' will try to disentangle the program logic - the reason being to discourage the phantom fiddler; such techniques are to be deplored as are programs which are inadequately documented or commented.

There are four basic properties that any computer program should have, regardless of whether it is to be offered for sale, placed in the public domain or used by the originator for his own purposes.

1. It should be logically constructed and portable (i.e. able to run without a great deal of modification on many machines).

2. It should be easy to follow and well documented.

3. It should work properly, giving correct answers or behaving in the manner indicated by the author.

4. It should run without excessive demands upon memory or CPU time.

A good program should be constructed so that it is user friendly at all stages; regrettably, few are. Brown and Sampson (1973) compare a good program with an amiable, large dog - not easily ruffled, slow to take offense and difficult to divert from its chosen course (they don't mention the large appetite of large dogs - excessive processing time or memory usage, perhaps?). A little unfairly, they go on to liken most peoples' efforts at programming to poodles (very finnicky about their food, demanding only the very best and tastiest titbits, very quick-tempered, easily upset and generally more trouble than they are worth). All this is possibly unfair to poodles but does apply to most software that I have seen (and to a lot I have written!). The worst offenders seem to be authors of programs in BASIC, closely followed by FORTRAN and PASCAL practitioners.

Good Design

Many of us, when faced with a problem which needs the attention of a computer, tend to jump in feet first with a rash of statements in whatever language we think we are most proficient at using. This isn't the best way except for the most trivial applications.

Define the problem - what it is and the best way to go about it.

Is a good program already available which will do what you want (or which can be easily amended for this purpose)? Do you know what you want to do, and how the computer can help you do it - there is no point in trying to write a database program if you don't know what a database is. Do you really need to use a computer or would a few minutes with a calculator serve just as well? If you can use another's program, would your data be in an appropriate form or does it need prior processing?

Having defined the problem, one should then outline the program, specifying its purpose, the types of data input and output, the variables to be included and the mathematics which may be needed. At this stage, it should also be possibly

to envisage any exceptional i/o conditions and what should be done to cater for them - such as numbers out of range, or data in the wrong format. Once this has been done, one can then look into the development of an algorithm which, amounts to a well defined and complete series of operations, and will produce the results which you expect. This isn't the sort of exercise that can be carried out on the back of an envelope!

Some folk like to use flow charts to help in the production of the algorithm but the production of a decent flow-chart is not an easy task - most people tend to become bogged down in program flow to the virtual exclusion of the program's function, particularly when dealing with large or complicated programs. It is difficult to confine some types of flow charts to one piece of paper - no matter where one starts on a piece of paper, the @?*! chart seems to spread onto several overspill sheets! An additional problem in flow-charting occurs when the user is unsure about the amount of detail needed - so that some sections of a program are dealt with in minute detail while other sections are less adequately covered. It is perhaps best to use flow-charts to help sort out problems associated with parts of programs where difficulty is experienced or where the logic may be complex but in general, one should lose no sleep if one cannot draw one!

Program design is important and these preliminary steps should not be carried out at the keyboard - one should, initially, concentrate on producing a plan for a modular program in which each unit contains a manageable number of lines (not more than, say 30 or 40). It is a good idea to place, say, output routines into subroutines (such as GOSUBs) rather than in the main part of the program unless they are used once only. These subroutines may themselves use other subroutines, so the importance of a well-structured program becomes apparent.

The best way to achieve this is by a 'top-down' approach in which the main part of the program is coded and then checked for errors, and the outlines of the subroutines established. The next stage consists of coding and checking the subroutines which the main program calls - and so on until the program is complete. This is the ideal situation and progress is often aided by tabulating the stages within the program; programmers who start with the subroutines and end up with the main program may run into difficulties. Some work through from top to bottom in a linear sequence and deserve the problems which they frequently encounter. Most of us tend to start with the coding of the algorithm or whatever comprises the heart of the program and to work down to the output section then up to the inout section. There are fewer snags to this approach but it is much less efficient than a well-planned top-down approach. It is always a good plan to keep a copy of the relationship of subroutines to the main program so that (if necessary) additional subroutines can be included without the bother of working out the program hierarchy all over again.

Portability

This probably causes more problems than any other aspect of programming in any language. If you are certain that your program will only be run on a particular type of machine which uses the same dialect of whatever language you are using - well, feel free to employ any machine or language implementation-dependencies that grab you, but don't expect your program to enjoy wider use unless you stick to 'basic' BASIC or 'portable' FORTRAN or PASCAL where non-standard language extensions, hence non-portability, do not apply.

This is true of programs published in this journal - those of us fortunate enough to have disks and a copy of disk MBASIC are inclined to forget that Nascom ROM BASIC is less well endowed with goodies (it's one-third the size). The problem becomes most acute when graphics are involved - or when data i/o is required from particular ports which may have different addresses according to the manufacturers' whims. This point is well illustrated by the problems involved in documenting the port allocations used by 80BUS boards from various manufacturers - some of whom were not very cooperative (the saga of which occupied the Editor of 80-Bus News in 1983).

A useful list of basic BASIC statements and commands is given in Monro (1978). I don't think that either of the usual PASCALs (Hi-Soft and COMPAS) give any indication of which of their statements and commands are non-standard as far as the ISO definition is concerned, but Prospero's Pro-Pascal and ProFortran do. It is always a good idea to include in a REM or COMMENT statement, the version of the language which you have used if it is not the 'portable' type.

Presentation

This is one bit of program development which is so often skimped. A fabulous program which has all the bells and whistles one could ever imagine and which everyone will want to use isn't going to inspire much confidence if the listing is untidy, disorganized or poorly commented. An incompletely commented program might be all right if it works all the time but what happens if you want to modify it later and you've forgotten why you put in a particular bit of code. Are your variable names sensible - it is helpful to use T for the sum of a series of numbers and N for the total number of observations - rather than Q or Z for example? Screen or printer output should be helpful - if you want a response to a prompt on the screen, put in a few words to request the input - such as 'Number of observations'. Similarly, results, whether on the screen or printer should have some explanation - such as headings for columns.

'Prettyprinting' is a rather 'twee' way of expressing the advantage of (for example) indenting parts of a program listing - it was first used by Nagin and Ledgard in 1978 as a means of pointing out the advantages (for following program logic) of various levels of indentation caused by typing spaces before the statements - thus nested FOR...NEXT loops could be traced very easily if the second and subsequent loops were indented by 2 or 3 spaces. COMAL does this automatically. A tidy screen or printer display of results is more easily understood and some 'prettyprinting' here is achieved by spaces or the use of tabs or particular print field descriptors. The readability of a listing is improved by blank comment lines in appropriate places.

Documentation

For many people, particularly those not familiar with a program, adequate documentation is essential, so its provision is a major and often disliked part of program development. If a program is for your own use, why bother? The trouble is that one's memory is not faultless and a lot of time and temper can be wasted. A good example of documentation is that provided with the various PEN programs - well set out and comprehensible to the average dodo. We all know of bad examples!

Integrity

In an ideal world, our carefully written, well-documented program would produce the right results from whatever data we stuffed into it - or it would do in other ways, what it was designed for. Unfortunately, this is rarely the case and even after extensive debugging, it may still refuse to function properly. A program should, once it has been found to be error-free, both in terms of syntax (confusion of 1 and I, 2 and Z or 0 and O, for example - or the wrong number of brackets) and the results obtained, be 'error-trapped' so that an incorrect or out-of-range input does not throw it into a state of utter confusion so that the machine crashes or an incorrect result is output. This takes a little time to organise but is well worthwhile. Test data may work perfectly but real data may produce odd results so one's test data should, where possible, include values lying at the extremes which are likely to be encountered. Commercial packages are variable in this respect - some 'throw a wobbly' if bad data are encountered but the majority are designed to cope with this eventuality and allow some user-intervention to correct the situation after the package has produced an error message.

The accuracy of the results obtained with a computer depends critically upon the level of precision to which the language implementation does its calculations - so an engineering program (or any in which the accuracy of the result had to be guaranteed to a large number of significant figures) would need to be structured so that the largest possible number of significant figures were employed and no 'rounding off' was applied. This can be achieved by the use of double precision arithmetic but there is a time and memory penalty for its use. On the other hand, a simple program to calculate the areas of triangles or to work out interest payments needs only single precision arithmetic - the default on most systems - and rounding off may be permissible. The use of built-in 'trace' or debugging facilities can allow intermediate results to be examined if final results are not as would be expected. One should not accept the computer's results uncritically.

Resource efficiency

Two major constraints on most microcomputer users are the amount of directly addressable memory and the execution time of a program; a large adventure program may use most of what is available - this is not usually a problem but tends to slow things down a bit unless the code is efficiently written. Multi-statement lines can help economize on memory usage but this should be employed with care if the legibility of the program is not to suffer. Some maths programs in which large arrays of data are processed may use lots of memory and take a long time to do it. In both cases, the use of memory and time can be optimized by cutting out unnecessary steps and optimizing where possible. A few examples may help illustrate this point.

It is quicker to use addition and subtraction rather than multiplication so X=Y+Y is faster than X=2*Y. Multiplication is faster than division so:

```
T=1/Z                                   U=X/Z
U=X*T          is faster than           V=Y/Z
V=Y*T                                   W=A/Z
W=A*T
```

Exponentiation is very slow, particularly when the 'power' used is a non-integer. It is a lot quicker to multiply:

T=X*X*X is quicker than T=X^3 (or T=X**3) and a lot quicker than T=X**3.0 which implies that a non-integer power is used (this does not apply to BASIC as far as I am aware).

Always use the supplied functions such as SQR or SQRT; the results are usually more accurate and more quickly obtained than if you devise your own version or use a fractional exponent! If you need to write your own functions (such as TAN *which isn't found in most PASCALs or FORTRANs) do check your version using* suitable values and paper and pencil.

Inefficient programmers often repeat calculations within the same statement or loop:

```
A=B+C-X+4*(B+C)/Y could run faster as two statements:

P=B+C
A=P-X+4*P/Y
```

Further savings in time can be gained by examining the way in which arrays are accessed. If an array element is used several times in a series of calculations, the value of the array element can be assigned to a variable and this variable used in the calculations which follow:

```
Z=X(I,J)                                A=B*X(I,J)
A=B*Z              is better than       L=C*X(I,J)
L=C*Z
```

FOR...NEXT or their FORTRAN or PASCAL equivalent DO loops work faster if unnecessary arithmetic is reduced - so constants can be placed outside the loop. It is also quicker to dispense with loops altogether in some cases where, for example, an array has to be initialised.

```
FOR I=1 TO 3
X(I)=0
NEXT I
```

if it is 'unrolled' to give the direct assignments:

```
X(1)=0
X(2)=0
X(3)=0
```

since this avoids the work/time overhead associated with loops.

The worst offenders in almost all respects are programs originally written for mainframe computers in which memory restraints are almost unknown and the clock speed and cycle time are such that even inefficiently-written programs run extremely quickly. Good programs, regardless of their function or the language in which they are written, should take into account most, if not all of the points raised in this article during their development. The extra time and care involved will certainly not be wasted!

References

Brown, A.R. and Sampson, W.A. (1973) Program debugging: the prevention and cure of program errors. Macdonald.

Monro, D.M. (1978) Basic BASIC. Edward Arnold.

LISTING FOR "MAKING CP/M MORE USER FRIENDLY - By C. Bowden"

```
;
; Bios workspace additional location
;
cursor: defw 0                  ; Saved cursor address
;
;
;************************
;*       BLINK IKBD     *
;************************
;
; Make the character 'get' part into a subroutime as it is used twice.
;
blinka: LD A,ESC                ; Get character from IVC/SVC keyboard
        CALL CRTA
        LD A,"K"
        CALL CRTA
        CALL VDUIC              ; Get the returned character
        ret
;
BLINK:  call blinka
;
; Hardware support from keyboard: ^T lead in Key.
;
        cp ct                   ; ^T Lead in char. ?
        jr nz,blink2            ; Return char. to CP/M if not.

;
        call savcur             ; Save Current Cursor Location
        ld hl,0030h             ; Top Row, Near R.H.S.
        call setcur             ; Cursor placing routine
;
        ld hl,topmsg            ; Print reminder of available commands
        call pmsg               ; Use the BIOS routine.
;
        call blinka             ; Get second char.
        push af                 ; Temporarily save it
        ld hl,0030h             ; Blank Top Line reminder prompt.
        call setcur
        ld a,esc
        call crta
        ld a,"*"                ; Blank to end of line
        call crta
;
        ld hl,(cursor)          ; Restore cursor to orig. location
        call setcur
;
        pop af                  ; Get char. back
;
        cp ct                   ; Second ^T ?
        jr z,blink2             ; Send it to CP/M
```

```
;
        if clock                ; Make conditional on clock hardware
;
        cp cd                   ; ^D for Date/Time update
        jr nz,trynxt            ; Skip if not ^D
        call aclock             ; Update the clock
        jr blink1               ; Exit with CR
;
        endif

trynxt: cp 10h                  ; ^P for page printer
        jr nz,trynx1
;
pageit: ld a,(plpag)            ; Page printer and reset line count
        ld (lcnt),a             ; Reset count
        ld c,cs                 ; Form Feed
        call parall             ; Send it to Printer
        jr blink1
;
; Change the codes sent to suit the Printer. Use C register to send.
;
trynx1: cp cc                   ; ^C for Compressed print
        jr nz,trynx2
        ld c,ct                 ; Cancel Enlarged if set (14h)
        call parall
        ld c,0fh                ; ^O for compressed
        call parall
        jr blink1
;
trynx2: cp csd                  ; ^N for normal print (0eh)
        jr nz,trynx3
        ld c,conr               ; Cancel condensed (12h)
        call parall
        ld c,ct                 ; Cancel enlarged (14h)
        call parall
        jr blink1
;
trynx3: cp cons                 ; ^S to enable screen paging
        jr nz,trynx4
        ld hl,scnt
        xor a
        ld (hl),a               ; Reset count to 0
        jr blink1
;
trynx4: cp conr                 ; ^R to reset IVC/SVC
        jr nz,blink1
        ld a,esc                ; Send ESC,"1"
        call crta
        ld a,"1"
        call crta
;
blink1: ld a,cr                 ; Return a Carriage Return.
blink2: RET
;

;
; Message for topline ( and sound the bell)
;
topmsg: DEFB "Enter ^T,"

        if clock
        defb "^D,"              ; Show ^D only if clock supported
        endif

        defb "^P,^R,^N,^C or ^S -",7,0
;
```

The next two subroutines were added to the MAP BIOS and are also used by the clock software.

```
;
;Enter with hl containing desired cursor position. h=row, l=column.
;
setcur: ld a,esc                ; Set cursor to (hl)
        call crta
        ld a,"="
        call crta
        ld a,h
        add a,20h
        call crta
        ld a,l
        add a,20h
        call crta
        ret
;

;
;Save current cursor location at address `cursor' in workspace
;
savcur: ld a,esc
        call crta
        ld a,"?"                ; IVC/SVC command for return
        call crta               ; of cursor co-ordinates.
        call vduic              ; Get first location (Row)
        ld (cursor+1),a         ; Save it
        call vduic              ; Get second (Col)
        ld (cursor),a           ; Save it.
        ret
;
```

****

Making CP/M More User Friendly by C. Bowden

This article briefly discusses the software available to improve CP/M and suggests some simple CBIOS modifications that the user can carry out, that will give additional system flexibility. These modifications apply to CP/M Version 2.

Only a day or two ago an acquaintance who is still using NASDOS and POLYDOS rang me and said 'You use CP/M don't you - I have heard that its as friendly as [expletive deleted] so I don't know whether to upgrade'. My reply was to the effect that since he was a Nasbus/80-BUS user, he could progress to what is probably one of the best implementations of CP/M available.

Certainly, to the average user of microcomputers, the mouse and window approach as exemplified in the Mackintosh is very attractive and easy to use. However I think that most readers of this newsletter are probably more aware of the inner workings of the machine than the average user, and would find the relative inaccessibility of the modern machine extremely frustrating.

The NASBUS/80BUS system is geared to the engineer, scientist or enthusiast who needs to be able to alter his hardware systems and associated software. The system may look a little old fashioned when compared with the sleek plastic machines around now, but it lives on whilst many others fall by the wayside. The enormous range of available CP/M software (and our investment in it), and the flexibility of systems like ours make it worthwhile to stay in the 8 bit world.

Of course standard CP/M is rather unfriendly. It evolved in a world where TTY terminals, Tape readers and Punches were still common. This is still reflected in software like ED.COM and MBASIC.COM, where the line editing features are truly as unfriendly as a hungry wolf. Fortunately, it is not necessary to remain locked in the embraces of standard CP/M. There is a lot of software around, much of it in the public domain, that can transform CP/M into a much more sophisticated system, and the various CBIOS's available on Nasbus/80-BUS systems make system extremely friendly when compared to many other systems. Above all, it is the 'ON SCREEN' EDIT feature, starting back in the good old NASCOM days, that is so useful.

CP/M consists of three modules, namely the CCP, the BDOS and the BIOS. These modules are much more fully described in the CP/M manuals, various books and some of the references in Appendix 2.

The CCP (Console Command Processor) is the part of CP/M that sits showing the A> prompt and waits for your command. It is 2k bytes long.

The BDOS (Basic Disk Operating System) is the interface between the CCP or currently running program and the BIOS. It is 3.5k bytes long.

The BIOS or CBIOS (Customized Basic Input Output System) holds the software that actually controls the system hardware. The CBIOS therefore varies from machine to machine. It can be any length up to about 4k Bytes long. Obviously the longer the CBIOS, the better the system (ought to be) in terms of hardware support and user friendliness.

It is not intended to describe in any detail the implementation or modification of advanced modules, which is usually well described in any accompanying documentation, but merely to describe what is available, and to indicate the main features. Readers who wish to obtain a more detailed understanding of CP/M or modifying it may find some assistance in articles referred to in Appendix 2

Improving the CCP - CCPZ/ZCPR2

A number of very useful improvements can be made to the CCP. The standard CCP provides six built in commands (DIR, ERA, REN, SAVE, USER and TYPE). There is no screen paging support, no 'PATH', and precious little else.

There are two alternative software replacements for the CCP. The simplest approach to system improvement is to replace the standard CCP with CCPZ or earlier versions of ZCPR. These programs are virtually the same and are based on ZCPR in the SIG/M or CP/M User Group Library. ZCPR literally means Z(80) CCP Replacement. Since Z80 code is usually more compact than 8080 code, it is possible to include more in the available 2k. The resulting new CCP is compatible with the standard CCP, and provides all of the standard facilities. It also provides the following extra features which are more fully described in the .DOC files provided from the library.

a) Displays the names of ALL files removed by an ERA command.
This can save potentially fatal errors.

b) Provides improved Directory display and SYS/DIR File options.
A)ll and S)ys options for DIR+SYS or SYS file display. Better directory formats may be selected for assembly.

c) Optionally displays the current USER number in CP/M prompt.
This is an assembly option.

d) Optionally provides screen paging support for screen output.
May be assembled to default to on or off. `P' parameter will toggle this option. (Normally off on this system as BIOS will provide better paging.)

e) Allows default USER area to be altered, by DFU command. eg: DFU 3
This would mean that the `path' would search USER 3 and not 0. The USER number may be given in Decimal or Hex. eg: DFU FH.

f) Provides a GET command to load a file to memory.
Eg: GET 8000 MYFILE.COM would load MYFILE.COM in memory at 8000H.

g) Provides JUMP and GO commands to operate a TPA resident program.
JUMP will `call' a subroutine. eg: JUMP E000H.
GO will call the subroutine at 100H. eg: Restart MBASIC etc. GO is the same as JUMP 100H.

h) The SAVE command allows HEX or DECIMAL, PAGES or SECTORS.
eg: SAVE 18 TEST.COM or SAVE 12H TEST.COM to save 18 pages. You may also give Number of sectors with `S' parameter. eg: - SAVE 10H ANOTHER.ONE S or SAVE 16 ANOTHER.ONE S for 8 Pages. N.B. 256 byte `page' or 128 byte `sector'.

i) More flexible handling of SUBMITS and CCP buffer default Cold Boot Commands.

j) Provides a LIST command to send a file to the printer.
eg: LIST THISFILE.TXT. The CBIOS will page the printer.

k) Provides a search path for required .COM files.
This is so useful that it merits a fuller description.

With a normal CCP if you issue a command like `STAT' then CP/M will look for STAT.COM on the current drive and USER area. If the file is not found, CP/M will give up with a query `STAT?'.

With CCPZ/ZCPR a three level search is performed. The CCP will look in the current user area of the default drive. If the file is not found, user area 0 of the default disk is searched. If this fails, USER 0 of drive A: is searched. If the file is still not found, CP/M will give up with the usual query. This

feature is extremely useful even on a two floppy system, but is invaluable in systems with Virtual or Winchester discs. (If the default USER is changed by the DFU command, the new default USER will be searched instead of USER 0 in the above example.)

NOTE that CCPZ has been standard with all Gemini CP/Ms for some time now.

Improving the CCP - ZCPR3

A second method of CCP improvement is possible by the use of the more recent versions of ZCPR. In addition to replacing the CCP, additional memory is used above CP/M and a considerable system enhancement is obtained. App. 2 Ref. 10 describes the system more fully. I am hoping to try ZCPR3 in the near future, but I have not used it yet so I can only indicate a few of the features :-

Environment Descriptor - Holds information on the ZCPR3 system.
Named Directories - Allows Drives and USER areas to be named.
Commands -
Resident Command CCP - GO, SAVE, GET, JUMP.
R.C.P. Segment - CP, ERA, TYPE, LIST, PEEK, POKE, PROT, REN.

Flow Control - Allows conditional testing of CCP processing.
Input/Output Package - Routes I/O and redirects data.
Terminal Descriptor. - Describes the Terminal and its commands.
Five level search PATH. (Easily redefinable)
Wheel Byte - Improves system security.
Drive and USER access via a C6: type command.
Support is provided for security via Passwords.
Large number of support utilities like HELP, MENU, SHOW, UNERASE, CONFIG, CLEANDIR etc;

One penalty of ZCPR3 is that some additional system memory is needed which reduces the available TPA by about 4k. If this were a problem with certain programs, a more standard operating system could be loaded with such programs.

Improving the BDOS - BDOSZ

This area of CP/M is the least amenable to modification. However a BDOS replacement known as BDOSZ is available, and improvements have been incorporated by using Z80 code to provide extra room for refinements in the 3.5k of space available. BDOSZ provides the user with much better recovery from errors than the standard BDOS. A summary of features that are provided by BDOSZ :-

a) When a 'SELECT ERROR' occurs you may enter a valid drive No. and continue.

b) When a Disk is 'R/O' you will be asked 'DO IT ANYWAY'. If you answer 'Y' BDOSZ will reset the disk and proceed.

c) If a 'R/O' file is found during ERA, the screen will display the file name and the same query will be printed. If you answer 'Y' the file R/O flag will be reset, and the ERA carried out. This applies to REN and SAVE activities as well.

d) If a bad sector is found, a 'READ' or 'WRITE' error message will be displayed. ^C will cause a Warm Boot, but any other key will cause the error to be ignored, so that it is possible to recover partly bad files.

e) DISK or DIRECTORY FULL errors. BDOSZ will display a 'Change Disk Y/N/^C' option and allows the Disk to be changed if desired.

BDOSZ is a worthwhile improvement to CP/M. (N.B. I have found a Bug - see note in App 1.)

Customizing the CBIOS

The CBIOS is the area where most can be done to make CP/M more user friendly. I recently bought an Alphatronic PC - A Z80 CP/M machine that was very well made and selling at a bargain price. I had thought that it would be useful as a second machine. I kept it for about six months, and then sold it to a friend who had previously been using a PET. He thinks that it is fabulous, so why did I sell it ? (I did put CCPZ onto it.)

Well, I found that I just could not put up with a machine that had no Type Ahead, no screen dump, no screen editing, limited paging (from CCPZ), limited function key support, no backspace key, kept changing 'case', and had partly ROM based BIOS. The supporting utilities for copying and formatting were not very good either - pretty screen displays, but no verification, no information on progress, and a bug or two. At least when I sold it it had CCPZ and some public domain utilities to help. It made me realize just what some people have to put up with.

A number of BIOS's are available for the Nasbus/80-BUS machines. I have used BIOS's by Gemini, Richard Beal (SYS) and MAP80 Systems. I have no knowledge of Nascom CP/M BIOS's so I cannot comment on them. The three BIOS's mentioned are all very good and provide features that are rare, if not unique, on 8 Bit CP/M machines. Fortunately, MAP and SYS BIOS's come/came with source code. (Unfortunately SYS cannot now be purchased, but I believe that there are quite a few about.)

If I can ride one of my hobby-horses here - Whilst I fully understand the problems of copyright and pirating, I feel that it is a completely retrograde step to withhold the BIOS source code from system purchasers. The vast majority of users are being penalized for the sake of the odd dishonest person who could probably be dealt with effectively by the law anyway. I feel that piracy is being made the excuse for trying to prevent people from expanding their systems except the Gemini way. SYS at one stage included Winchester support, but later this had to be removed. (Because users didn't need to go to Gemini for a new BIOS or to buy a Winchester ?) Then SYS itself was withdrawn because of copyright problems. (See App. 2, Ref 4)

The new GM849 card seems to me to be a similar case in some ways. I recently wanted to purchase a spare Disk Controller card for several machines at work that are in heavy use, and I was annoyed to find that the GM849 card that supercedes the GM829 needs version 3.4 of the BIOS to run it. (When the CP/M was upgraded to Version 3.2 BIOS at work a few months ago there was no mention that it would not drive the 849 - nor was there a mention of the 849). The CP/M on the machines in question cannot be easily replaced due to special custom routines, so I will have to try to alter the CBIOS to support the 849. This sort of work I can do without.

Logically one would expect that Gemini would want to offer the best BIOS available, but their BIOS does not include a number of useful features such as locked EDIT and extended Screen Paging, and without the source code, it is of course impossible to customize it for additional features such as Clock support, or the type of keyboard feature described later. Neither are MAP products supported, again denying users of the BUS the most flexible system. Once one is accustomed to such features, it would be difficult to return to a BIOS without them. I appreciate that there must be limits to what can be provided as 'CONFIG' options, but I do not see that as any reason to deny the users ie: CUSTOMERS, the facility to do their own customizing.

Fortunately I have been able to combine many of the features of SYS with the MAP CP/M 2.2 BIOS, and then added in some of my own routines to provide myself with the features that I want. (For personal use only.) Customization like this takes a lot of time though, and obviously would not be possible for many users, even if the required source codes are available. The restrictive approach must be denying users the best operating environment, and thus could be detrimental in the long run.

Enough of the complaining - If Gemini BIOS 3.x is what you have it is still a lot better than most CP/M BIOS's. If you are fortunate enough to have SYS18 then you will have everything that you need except possibly Winchester support. The MAP CP/M BIOS is very good and easily adaptable to different disk types and formats, including Winchesters, but lacks a few features like screen dump, screen paging and VBOOT, which can be added anyway.

If you do have the source code of your BIOS available then you may like to modify it to include some simple but extremely useful keyboard activated hardware support. The rest of this article will describe the sort of thing that I have added to my BIOS.

I first had the idea of providing keyboard support to hardware in order to cause the printer to advance to top of next page.

In order to permit a direct keyboard command it is necessary to trap and process the relevant keystrokes instead of passing them to CP/M. This means that it is necessary to modify the lowest level BIOS routine reading the keyboard. The 'special' keys must be intercepted and processed and then control returned smoothly to CP/M.

The first problem was to find a suitable key for the purpose. After a long perusal of key allocations I decided to use ^T. This choice was based on the fact that few programs used ^T and it could be dispensed with, and also that 'T' was a synonymous with printer T)hrow. The keyboard routine (pkbd: in SYS and BLINK: in MAP) was accordingly modified so that if ^T was typed, the printer advanced to top of next page, and the BIOS lines per page counter was reset.

It did not take long to realize that I could utilize this method to provide a number of other simple features that could avoid me loosing what little hair I have left. Due to the shortage of key allocations, I decided to make ^T the lead-in key, and to program the software to expect a second control key depending on the desired function. This had the advantage of allowing a key to be chosen, that related to the function to be achieved. In addition, it allowed the option of sending on a ^T to CP/M if desired.

As an aid to the user, a message is displayed on the locked top line of the screen whilst the system is waiting for the second character, reminding the user of the options available. After the second character has been typed, this message is erased. After the routine has been processed, a CR character is returned to CP/M. This also happens if an erroneous second character is typed. Keyboard software is not noticeably affected since only the first ^T is searched for. Any other character is returned normally to CP/M.

A problem became apparent a while after I added a Real Time Clock to my BIOS. (I have described this in some detail - see App. 2. Ref 5). Since the article was published I discovered that if the clock updated during screen EDIT, or operation of screen oriented software, (eg: Cursor Addressing), the Clock could interfere with the 'ESC' sequences and corrupt the screen. At first I solved this problem by disabling the clock or running a BIOS without the clock. Since this resulted in incompatibilities in the various CP/M systems, I later decided to make the clock 'passive' and provide three ways of updating it.

The display now updates on Warm/Cold Boot. On CALL from an external program. By user initiated demand from the keyboard as described below.

This has eliminated the screen corruption problems referred to above. The number of functions supported is optional. It is currently limited in my case by the BIOS size equalling the 4k of BIOS space available on the system track. I have at present added seven functions as direct keyboard commands. They are :-

a) ^T,^D - Call clock routines and Update Date and Time display on Locked top line at Left hand side of the screen.

b) ^T,^P - Send a Form Feed to printer, causing it to advance to top of next page, and reset BIOS lines per page count.

c) ^T,^R - Reset IVC/SVC to 80 wide. Useful for the (very) rare time when screen corrupts or gets out of vertical sync.

d) ^T,^N - Switch attached (Epson) Printer to Normal Print.

e) ^T,^C - Switch Printer to Compressed Mode.

f) ^T,^S - Reset Screen Paging flag to 0, to enable paging if a 'W' or 'K' was issued and a Warm/Cold Boot is undesirable.

g) ^T,^T - Pass ^T on to calling program.

The listing above, on pages 49 and 50, shows in upper case part of the original BIOS code (MAP BIOS), and lower case shows the modifications, but note that the cursor definitions have been moved into CURON/CUROFF to fit in with other BIOS modifications.

If there is plenty of spare space in the BIOS, quite a number of printer support features could be incorporated, and even function key redefinitions.

Appendix 1

I have detected a problem with BDOSZ. Recently I have added a Winchester disk to a Gemini system. Despite the fact that CP/M Plus is available, CP/M 2.2 on the machine in question has been customized extensively along the lines of this article and this operating system is still preferred for many purposes. Consequently I needed compatibility between CP/M 2.2 and CP/M Plus in Winchester support. I eventually cobbled together a CBIOS with all of the required features and compatible Disk format by using a couple of MAP CP/M 2.2 BIOS's and some routines from SYS.

CP/M 2.2 at that stage had CCPZ and BDOSZ in operation. I then found that both CP/M 2.2 and CP/M Plus were treating the floppy drives the same in respect of the way the Directory Block allocation was being written, but wrote them differently to the Winchester. CP/M Plus was putting a 00 byte between block numbers on the Winnie directory, but CP/M 2.2 was not. After reverting to standard CP/M 2.2 BDOS the Directory on the Winnie was the same on both versions of CP/M. The 00 byte is not present on the floppy directory under either operating system but present on the Winnie under both. If anyone knows of a patch for BDOSZ to make it treat the Winnie properly, I would be pleased to obtain details.

Appendix 2

Some useful References:

1)	Disks and CP/M	I.N.M.C.80 News No. 5
2)	Customizing your CBIOS	80BUS News. Vol 2, Iss 1
3)	SYS-Latest Developments	80BUS News. Vol 2, Iss 1
4)	SYS is dead - long live 7	80BUS News. Vol 2, Iss 4
5)	CBIOS Real Time Clock	80BUS News. Vol 3, Iss 6
6)	CP/M Features & Facilities	CPMUG U.K. Vol 1, No 1.
7)	The BIOS	CPMUG U.K. Vol 1, No 5.
8)	The Console Command Processor	CPMUG U.K. Vol 1, No 8.
9)	ZCPR Replacement CCP	CPMUG U.K. Vol 2, No 2.
10)	The SB180 - Software	Byte. Oct 1985
11)	Soul of CP/M	Sams & Co. (U.S.A.)
12)	Mastering CP/M	Sybex. (U.S.A.)
13)	CP/M manuals.	Digital Research.

A Sideways Look at Benchmarks by P.D. Coker

A few years ago, PCW published a series of Benchmark programs in BASIC, together with a list of micros on which they had been run; these were arranged in order on the basis of their average performance on 8 programs which were designed to test various features of the machines' versions of BASIC. No Nascom or Gemini Nasbus/80-BUS machines were tested although one machine was designed by Gemini (Mimi 801, which did quite well in the tests).

The benchmarks were structured so that for tests 2 - 7, subtracting the timing for the previous test from the current test time would give the time due to the routine under examination. The eighth benchmark differed from the rest, which tested the timings for arithmetic functions, GOSUBs and array handling since it dealt with transcendental functions such as SIN, LOG and exponents:

```
100 REM Benchmark 8 (PCW)
110 PRINT "S"
120 K=0
130 K=K+1
140 A=K^2
150 B=LOG(K)
160 C=SIN(K)
170 IF K<1000 THEN 130
180 PRINT "E"
190 END
```

I tried this one on the following machines: (CPU and system clock frequency in brackets)

UK101	(6502, 1MHz)	72.1 secs
BBC B	(6502, 2MHz)	51.3 secs
Nascom 2	(Z80, 4MHz no FDC)	50.1 secs
$BBC + TORCH	Z80 2nd processor	30.8 secs
*Nascom 3	(Z80, 4MHz)	53.0 secs
*Gemini MultiBoard	(")	52.2 secs
*MAP-80 Systems	(")	51.9 secs
+MAP-80 Systems	(")	50.8 secs

$ Using Z80 BBC BASIC and CP/N
* Using MBASIC and CP/M 2.2
+ Using MBASIC and CP/M 3.0 (CP/M Plus)

I then found that the timing for the standard BBC was 5.1 seconds according to PCW! *Evidently their Beeb was supercharged! More likely, line 170 of the* program had been amended to 'IF K<100 THEN 130' since the average value of 14.6 over the 8 tests was based on a timing of 51 seconds for no. 8. Colleagues who have run this test on other machines mentioned in the PCW list have also commented upon inaccuracies in the reported speeds. The Z80 version of BBC BASIC as used on the TORCH second processor was extremely fast but disk access with this system is dreadfully slow. I did not have access to the Acorn 6502 or Z80 second processors - it would have been interesting to see their timings.

More recently, a series of Benchmarks in PASCAL was produced by the same magazine; these were 15 in number and designed to take account of the greater range of features found in this language, but the overall philosophy was the same, namely, by subtracting the timing for the previous test from the current test time, some idea could be gained of the time for the routine under investigation. I was able to test COMPAS v. 1.08, HP4 and ProPascal v. 2.1 using CP/M 2.2 and CP/M Plus and came to the conclusion that for BM15, which was a PASCAL version of the BM8 BASIC version, HP4 was quite a lot faster than COMPAS, possibly because it uses 7 rather than 11 significant figure accuracy. The results were as follows:

	CP/M 2.2	CP/M 3.0
COMPAS	48.2 secs	47.1 secs
HP4	8.0 secs	7.9 secs
ProPascal	11.5 secs	11.3 secs

Parkinson (1983) utilized a version of a much more rigorous benchmark while testing the HSA-88B High Speed Arithmetic processor. None of the PCW benchmarks tested the accuracy of the particular language/machine combinations using the intrinsic arithmetic/trigonometrical functions of BASIC or PASCAL, but Parkinson found an excellent test in Duncan (1983) which provides some interesting results. The program used is as follows:

```
5 REM Benchmark from Dr Dobbs' Journal no. 83 (120-122)
10 NI=2500
20 A=1
30 FOR II=1 TO NI-1:A=TAN(ATN(EXP(LOG(SQR(A*A)))))+1:NEXT II
40 PRINT USING "A = ffff.ffff";A
```

The purpose of the program is to produce a result as close as possible to 2500 in as short a time as possible. Parkinson produces results using both compiled and interpreted MBASIC as well as a modified MBASIC, HiSoft HP5 Pascal and a simple assembly language version; the last three were run using the HSA-88B, and all were run at 4MHz.

I used CBASIC, COMAL-80, PASCAL and FORTRAN versions of the program and also tried it out on a UK101 and a BBC B (both in its standard version and with the TORCH Z80 processor). The results are shown below, with the MBASIC results for comparison.

MBASIC and Fortran 80 work to 8 digit, ProFortran and ProPascal to 7 digit accuracy in single precision and respectively, to 17 and 14 digit precision when in double precision. COMPAS and COMAL have no double precision facility and work respectively to 11 and 7 digit accuracy. The BBC BASICs (both 6502 and Z80 versions) use 9 digit accuracy.

		CP/M 2.2	CP/M 3.0
MBASIC (interpreted)	2304.86 in	225 secs	218 secs
" (compiled)	2304.86 in	183 secs	
CBASIC (semi-compiled)	2485.76 in	2200 secs*	
COMAL-80 (interpreted)	2407.10 in	89 secs	
ProFortran (compiled)	2773.50 in	107 secs**	103 secs
Fortran 80 (")	2304.86 in	185 secs**	180 secs
COMPAS (")	2500.00 in	330 secs**	306 secs
HP4 (")	319.67 in	54 secs!	52 secs!
ProPascal (")	2772.60 in	106 secs**	102 secs
BBC B BASIC (interpreted)	2499.69 in	310 secs	
BBC Z80 BASIC (")	2498.82 in	188 secs	
UK101 8k Microsoft BASIC	2506.44 in	301 secs	

* 14 digit accuracy
** TAN not implemented; user supplied function (SIN/COS)
! I cannot understand this result at all - any offers?

Additionally, I ran Microsoft BASIC86 and Digital Research Personal BASIC (both interpreted rather than compiled), on a GM888 8/16 bit 8088 co-processor in a MultiBoard system with the Gemini version of CP/M 86. The clock frequency is 8MHz, but according to their catalogue Gemini reckon that with the interaction with the 4MHz 80-BUS it is more like 6MHz overall; however, the accuracy and timing of both were somewhat poor!

	Time (secs.)	Value
BASIC86	130	2179.85
Personal BASIC	186	1090.76

I understand that this version of BASIC86 (5.21) is the genuine, non-patched article but the version number is exactly the same as my version of MBASIC (Z80). Its precision would have been improved by using double precision for variable A, but the time penalty would be quite severe. The DRI BASIC is an interesting version capable of running MBASIC programs which, in spite of its poor precision in this application (6 figure accuracy) and its slowness, is a delight to use.

Incidentally, I was surprised to see that Parkinson appeared not to have tried COMPAS with the HSA-88B since I would have expected some improvement in execution time.

Factors affecting Benchmark timings

Several factors will ultimately affect the timing (and accuracy) of a Benchmark test.

Hardware

System configuration and architecture will have important effects on disk i/o, memory and screen access timings. Unless a test is specifically designed to test these, their usage should be as low as possible.

System clock frequency

The importance of this is self-evident. A test which is run at 4 MHz should normally complete in a shorter time than one run at 2 MHz unless the language implementation is less than optimal (as, for example, the patching of the 8080-based Microsoft languages to run on the Z80 processor).

Software

The use of a compiled language rather than one which is interpreted will always speed things up and the accuracy will not be affected. The main problem as far as both accuracy and speed are concerned lies in the efficiency of the algorithms which are used by the particular version of BASIC or whatever language, and the extent to which rounding errors may affect the results of calculations.

On micros, most BASICs work (in single precision) to an accuracy of 8 digits but the BBC version uses 9 digits and CBASIC, 14. Rounding errors will be marginally less in 9 digit than 8 digit and much less in 14 digit, although the speed penalty may well be too great. Some versions of PASCAL and both versions of FORTRAN lack a TAN function; this has to be supplied as (SIN/COS), either as a user-supplied function included at compile-time, or possibly as an additional line of code such as TAN=SIN(A)/COS(A) (which will slow things down rather less). My FORTRAN tests used the latter approach.

The Z80 version of BBC BASIC appears to be a lot faster and only marginally less accurate than the 6502 version. It is available for 80-BUS machines and it would certainly seem to be a good buy both in terms of price and facilities

since MBASIC is both expensive and long in the tooth. It would be very interesting to compare the speed and accuracy of the Microsoft compatible Mallard BASIC. I understand that it is substantially faster than Microsoft, presumably because it is written in Z80 rather than 8080 code.

The trigonometric functions produced penalties in all cases in both accuracy and/or speed. If the program was run without them, the result was very close to 2500 in all but one case (HP4) - with a range of 2499.49 to 2501.03. The timings were halved (approximately). HP4 gave a result of 2472.00; rather a surprising lack of accuracy, attributable in the main to lower precision arithmetic.

In this particular application, there is very little access, apart from console operations, to whatever operating system is in use and one is effectively comparing the console performances of the CP/M machines. I did try the ProPascal version of the Dobbs benchmark on a Research Machines 380Z-D; it took 124 secs to complete - about 20% slower than on the 80-BUS machines.

What now?

It will be fairly obvious that a benchmark test is not the "be-all and end-all", neither is it an infallible guide to which language should be used on your machine. For most non-professional users, the question of a second language may not arise unless they go for HiSoft PASCAL - and the professional may be using applications packages written in one language which are not available in any of the others.

From my own experience, I must say that differences in timing due to architectural/configuration features among any of the 80-BUS machines are small, provided that they are running the same version of CP/M (in my case 2.2). CP/M 3.0 (CP/M PLUS) is available for all 80-BUS machines provided that at least 128k of RAM (for its banked memory function) and Gemini or MAP 80 video and floppy disk controllers are available in the system. It is a delight to use and has many handy facilities which make it much more versatile than CP/M 2.2. It does increase the speed of execution of these Benchmarks a little (up to 5 or 6%); in terms of real-time applications, this small saving in time may be worth considering. I'm also very glad that I don't have too much to do with CP/N as used on the BBC/TORCH. It has some nice facilities but it is incredibly slow. CP/M, for all its snags, is supercharged by comparison!

Ultimately, then, the choice is dictated by the application for which you will be running most of your programs. Few people will want to carry out several thousand trig. operations but it does imply that a slow performance in the benchmarks given in this article will be a disadvantage when running programs in which a lot of use is made of graphics. The same is likely to be true of programs in which there is a significant amount of 'number crunching', and accuracy (or the lack of it) may be a serious problem.

I haven't yet come across a benchmark which will test the speed of string handling, and disk i/o tests will depend crucially upon the type of drive, stepping rate, type of FDC, and such software horrors as the efficiency of the BDOS code and the way in which data is written to and read from the disk. There's a lot to be said for using a virtual disk or some of the newer 96TPI drives with 3 ms stepping rates, a great improvement on the Micropolis' 10 ms and Pertec's 25 ms. It would appear that the HSA-88B offers a lot of scope for increased speed and (possibly) accuracy but costs as much as or more than many compilers. It would be interesting to see how a 16 bit FORTRAN or PASCAL copes with the Dobbs Benchmark both with and without the aid of the 8087 arithmetic co-processor that can be added to the GM888.

By way of a postscript, the performance of the obsolete UK101 was quite impressive and supports my feeling that ROM BASICs on 6502 machines tend to be rather more efficient than either disk or ROM BASICs on the majority of 8080/Z80 machines. I had, some time ago, run the first of the programs mentioned in this article on a UK101 running at 2 MHz and it pottered through in about 40 seconds.

What was even more surprising was the ability of the 6502 (bog-standard 1MHz version) to run at 2MHz. Most 2 MHz Z80s tend to wilt if you attempt to run them at 4 MHz - maybe the device speed selection criteria are more stringent these days.

I was also quite impressed with COMAL-80's timing - it was by far the fastest of the languages tested and its accuracy was pretty reasonable as well. I've never understood why COMAL has never 'caught on' in this country - it's very popular in Denmark and is fairly easy to get on with.

References

Duncan, R (1983) 16-Bit Software Toolbox. Dr Dobb's Journal, no.83 (September 1983), 120 - 122.

Parkinson, D.W.(1983) Arithmetic Processor Review. 80-BUS News. vol. 2 no. 5 (September/October 1983), 7 - 14.

NEWBURN

NEWBURN ELECTRONICS 58 MANSE ROAD, BALLYCARRY,
Co. ANTRIM BT38 9LF
Tel 09603-78330

NE871 80-BUS 32 OPTO-INPUT BOARD. **£350**

Provides 32 opto isolated inputs to the 80-Bus. 5V to 125V sensitivity. On-board led indication for all 32 inputs.

NE874 80-BUS 16 CHANNEL 8 BIT A/D **£275**

May be configured as 16 balanced or 8 unbalanced + 8 balanced inputs. Input sensitivity between 100mV and 10V.

NE875 80-BUS 16 CHANNEL 8 BIT D/A **£330**

Provides 16 analogue voltage outputs 0-10V. Current supply on each output up to 40 mA. Short-circuit protection.

NE876 80-BUS 64 CHANNEL DIGITAL INPUT **£195**

Provides 64 digital inputs to 80-bus. Inputs are protected up to +/-50V. Any input voltage between -50 and +50V may be used to switch inputs, threshold is +2V.

NE877 80-BUS 64 CHANNEL DIGITAL OUTPUT **£215**

Provides 64 digital outputs from the 80-Bus. Each output is short-circuit protected and capable of output of 50V at 500mA. Ideal for driving relays & lamps etc.

Industrial klippon block can be supplied for direct plant connections for all the above boards.

NE840 14 SLOT BACKPLANE **£67**

Correctly terminated for 80-Bus. May be reduced in size. Supplied complete, less connectors.

GM833/2M 2 MEGABYTE UPGRADE KIT **£290**

Expands a standard GM833 (512K) to 2 Megabytes. No changes are required to the standard Gemini bios to support this board.
The board may be upgraded by us for £350.

CA802/1M 1 MEGABYTE COSTGOLD UPGRADE KIT **£250**

Expands a standard Costgold (256K or 128K) board (CA856) to 1 Megabyte capacity. No changes are required for MS-Dos or CP/M86. The board may be upgraded by us at a total cost of £300.

Please add 15% Vat and £4 p/p per order

NEWBURN

EV COMPUTING LTD

700 BURNAGE LANE BURNAGE MANCHESTER M19 1NA
(Registered office) Telephone: 1061 431 4668

EV Computing Ltd. Forthcoming Products for 1987

-- PIO Interface boards --
-- For Industrial control applications --

All the below boards connect to either the Gemini or Nascom 26 way PIO ports. The software is supplied on disk to be re-configurable depending on PIO port addresses. A lead to plug into the 26 way PIO connector is provided with all boards.

If you require more PIO's then use a Nascom I/O board or a Gemini GM816. Both give 3 more PIO's.

LCD Interface panel £ 35.00

Supports Alphanumeric LCD modules from 16x1 to 40x2
Software to run in ROM or under CP/M supports most standard terminal control codes.
Can replace CP/M console output device
On board bleeper and viewing angle control

LCDs can be supplied from £ 35.00

3 channel A/D convertor £ 80.00

Three multiplexed 8 bit channels with optional sample & hold
Software supplied will run under CP/M or in ROM
On board gain & offset trimmers for each channel
On board -5 Volt convertor allowing AC operation

Keyboard Interface panel £ 25.00

Supports matrix type keyboards from 3x4 to 8x8 key locations
Software to run in ROM or under CP/M
Can replace CP/M console input device
Easy User configurable keyboard type & layout
Shift/Control/Function key features
Several standard configuration files supplied

Keypads available from £ 5.00 for 6x4 matrix

CMOS RAM file £ 75.00/64K, £195.00/256K

64K to 256K Battery backed CMOS RAM
Software to emulate disk drive under CP/M as well as low level driving software.
Designed to solve industrial storage needs, systems available to store upto 100 Mbyte if required.

Note: All software supplied is written under CP/M and uses the M80 assembler & L80 link loader which must be provided by the user.

All prices exclude VAT, Please specify CP/M disk format required.

CP/M Plus (vers 3)

For NASCOM and Gemini computers

Features:

CP/M 2.2 file compatibility
Banked memory system
Fast warm boot from banked memory
Faster disk access:—
Directory hashing, memory cashing, multi sector I/O
Better implementation of USER levels
Greatly extended and user friendly utility commands
20 transient utility commands
Includes MAC the DRI assembler
Multi command entry on single line
Multiple drive searching facility
Console redirection
Password file protection
Date and time file stamping
Larger disk and file handling
29 additional BDOS calls
Extended BDOS capability by easily attached RSXs
Winchester, floppy and virtual disk
Mixed drive/formats
Full source code of BIOS supplied
PLUS PLUS PLUS !!!!!!!!!!!

Now Only £199

Excluding post and packing and VAT

Are you

Developing Systems

Consider our modular approach
Nasbus/80 Bus compatible

CPU card
Z80 CPU incorporating memory mapping
64k RAM on board (expandable)
Z80 S10 providing two RS232 channels
CTC providing programmable baud rates
P10 providing parallel/centronics I/O
Parallel keyboard port

VIDEO card (VFC)
80 by 25 line output
Fast memory mapped display
On board floppy disk controller
Can be used with CPU card under CP/M
Available in kit or built and tested

DISK card (MPI)
Mixed 3", 3.5", 5.25", 8" drives supported
SASI Winchester interface
Z80 S10 providing two serial channels
CTC providing programmable baud rates

RAM card
64k to 256k (in 64k steps)
Supports 64/32k paging 4k mapping
Available in kit or built and tested

CLOCK card (RTC)
Attaches to any Z80 P10
Retains Centronics parallel output
Battery backup

PRICES

CPU	£230	MPI	£185
VFC	£199	RAM (64k)	£180
RTC	£35	RAM (256k)	£285

All prices exclude carriage and VAT

For further information contact:

MAP 80 Systems Ltd

Unit 2 Stoneylands Road, Egham, Surrey
Tel: 0784 37674