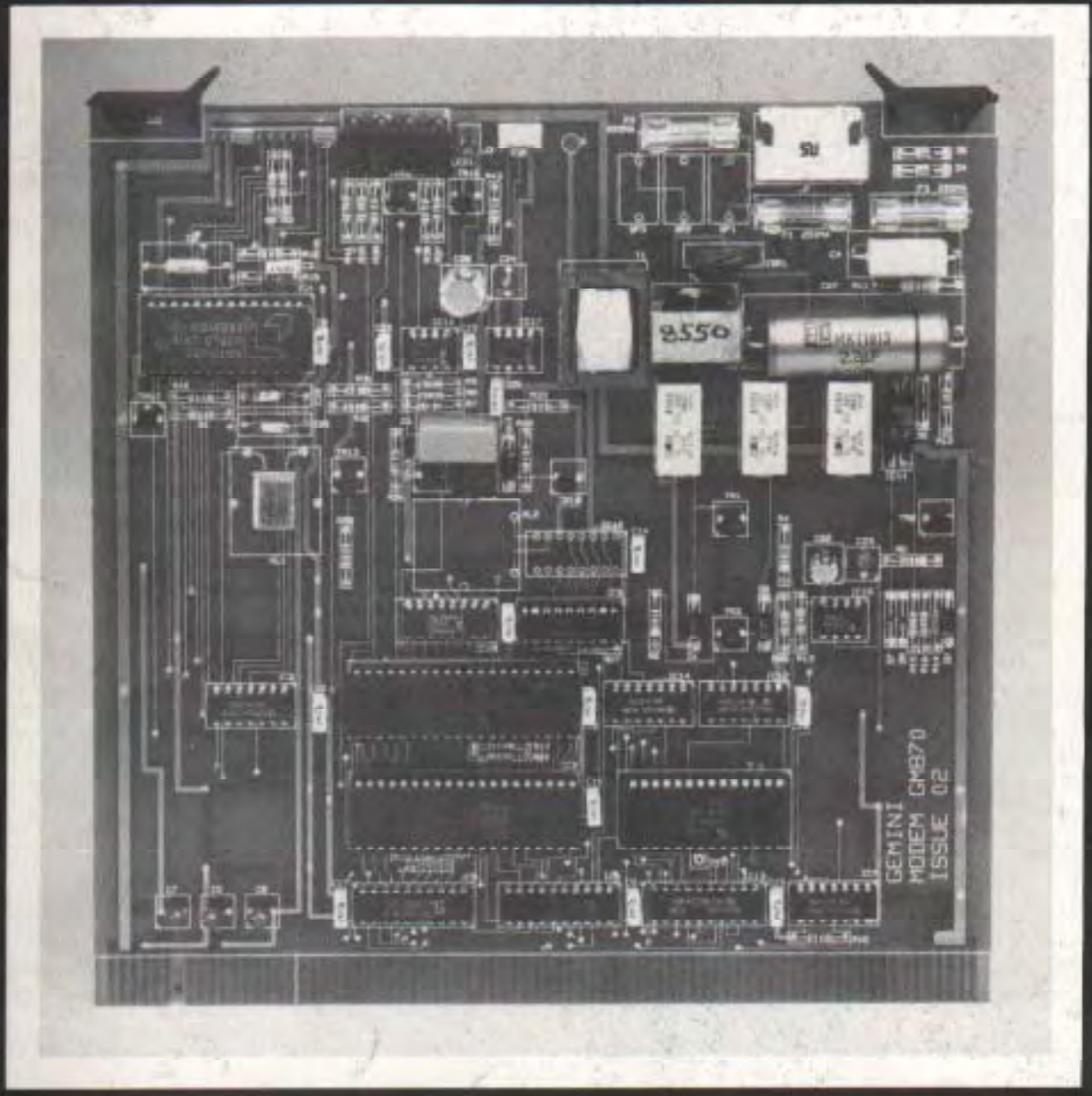
80-BUS NEWS

SPRING 1985 VOL. 4. ISSUE 1



GM870 80-BUS MODEM BOARD

SPRING 1985 **80-BUS NEWS** Volume 4. Issue 1.

CONTENTS

Published by Gemini Microcomputers Ltd. Printed by The Print Centre, Chesham.

SUBSCRIPTIONS

Annual Rates	UK	£9	Rest of World Surface	£12
	Europe	£12	Rest of World Air Mail	£20

Subscriptions to 'Subscriptions' at the address below.

EDITORIAL

Editor : Paul Greenhalgh

Associate Editor : David Hunt

Material for consideration to 'The Editor' at the address below.

ADVERTISING

Rates on application to 'The Advertising Manager' at the address below.

PRIVATE SALES

Free of charge to Subscribers by sending to the address below.

ADDRESS: 80-BUS News,
c/o Gemini Microcomputers Ltd.,
Unit 4, Springfield Road,
Chesham, Bucks. HP5 1PU.

EDITORIAL

I was not going to include an Editorial in this newsletter, but I notice that there is part of a page free and so I am taking the opportunity to fill it. The reason for the free space is that, just in case you haven't noticed, we have had this magazine typeset, and this has resulted in a dramatic reduction in the space taken up by a given amount of text.

One positive aspect of this typesetting is that our printing costs will be significantly reduced - produced using our normal techniques this magazine would be 70 plus pages long. We should therefore be able to contain subscription fees at their current annual level. There are however also a couple of negative aspects. Firstly this magazine has been subject to delays very much above and beyond our normal delays. Hopefully this will not be repeated as we all get used to the new modus operandi. Secondly there is the psychological effect of the magazine appearing to be smaller (as it is thinner) than normal, despite the fact that it does actually have a higher content than normal. Hopefully, as our readership has a very much higher than standard level of intelligence, this will not prove a problem. [Ed. – this guy thinks that creeping will get him out of anything!] [Ed. – Yes!]

Anyway, at long last here it is. Happy reading, and please let us know if you approve of the new format.

Letters to the Editor

Bulletin Board

I am the SYSOP of CBBS South-West. This is a bulletin board running the CBBS type of software that runs under CP/M 2.2. The system is *Nascom* based with a *Nascom* 2 and *Nascom* FDC with *Gemini* 64k RAM board and *Gemini* RTC. The Modems used are ex-*BT* and cater for 1200/75, 75/1200 and, of course, 300/300, and the selection of modem type is AUTO select. The System has been on-line for about 16 months and has received around 15,000 calls. The computer is on-line 24 hours a day and the number is 0392 53116. I am a member of AFPAS (Association of Free Public Access Systems). The word format is 8 bits, No Parity, 1 stop bit. I do get quite a few callers with *Nascom* and *Gemini* systems.

Yours truly, B Hitchcock, Alphington, Devon.

80-BUS Reader Survey

With reference to the 80-BUS reader survey (i.e. the Questionnaire) I have the following additional comments to make:-

> Why are *Gemini* still in the computer business? Many rivals of 5 years ago have long since gone. I believe that it is at least partly due to offering things which others do not, like 'hardware support' and 'software support' to end users.
>
> A user-expandable system is offered which meets the ever growing needs of owners. Multi-sourcing of some types of hardware for use on a common BUS is a feature which gives some feeling of security when parting with hard earned money. The fact that rival board suppliers suffer differences of opinion does not seem to matter too much so long as the end result works satisfactorily. It might be better if rival manufacturers settled their differences and concentrated their efforts on mutual survival of the 80-BUS.

We can all make rash decisions to buy imported 'plastic boxes' which will perform well in the immediate future, but what about product support in 2 – 3 years time? Or being in possession of a comprehensive hardware and system manual from Day 1? The internal workings of many rival computer systems are a well kept secret to owners.

Markets seem to be divided at present, between the following:-

> 8-bit, 64K RAM, CP/M-80 or other,
> 16-bit, 256K RAM, CP/M-86 or other, machines.

When considering larger machines or hard disks, the price takes off and reaches a level which is beyond the sort of users buying our computer. It is very desirable when a system is being expanded by its owner.

Obsolescent board types could be offered at reduced prices or as kits for those who have the ability to assemble and test them but are short on the means of paying. At least boards sold in this way would stimulate a market for software, which *Gemini* also sell. The trade in secondhand boards should be encouraged to assist customers improving their systems by adding facilities or disposing of redundant boards. Where would the motor industry be without a used vehicle market?

For the 80-BUS News, I would like to see an article on mains borne interference suppression. A different 'fridge in this household made my system unusable until modifications were done to mains input circuits. The magazine could support *Microsoft* BASIC in UK as nobody else seems to. To the user, features on aspects of programming in pure *Microsoft* BASIC, together with membership of CPMUGUK, gives a useful software base without a large financial outlay which would only be justified if the computer was being used in conjunction with a profitable business.

Keep up the good work.
Yours truly, A A Bryan, Cambridge.

Turbo Pascal – 1

Has anybody else bought TURBO PASCAL for a *Gemini*? I bought it a few months ago after reading glowing reports in various computer journals including the CPMUG magazine. I bought it from *Grey Matter* Ashburton, Devon, who can supply software in *Gemini* formats. I do not intend to review it as it has been done many times before. The software comes with a built-in screen editor and installation program. As I have come to expect, *Gemini* is not on the standard list of terminals but facilities are provided to enter each control code separately so that the *Gemini* IVC can be used. Obviously, you have to read the IVC Manual to check the required codes.

On attempting to use the screen editor (which is like a mini *WordStar*), it did not work and the computer hung up. We have all seen this before. The problem is associated with polling the keyboard while sending control codes to the screen. This upsets the IVC. I found that if locations ★4133 – 4135 are patched to ★0 and location ★413B changed from C8 (RET Z) to C9 (RET) then the editor works. This patch removes a call to the keyboard. The only difficulty that I have discovered so far is that the 'FIND + REPLACE' function will now only work in option 'N' mode and not in the

default mode. (See page 31 in the manual.) I must say that this has caused no real difficulty at all.

Yours truly, Tom Gibson, Middlesbrough, TS6 0HU.

Ed. — see my comments following the next letter.

Turbo Pascal – 2

I have installed *Borland International* Turbo Pascal onto my *Gemini/Nascom* system. With the *Nascom* video (generated using MOVCPMN) the system works perfectly, but when trying to use the IVC (Version 1.0), the system hangs after trying to edit a file. Other commands work perfectly and it appears to be only when cursor addressing is used. Have you come across this problem before or can you suspect a cause?

By varying screen size that the Pascal recognizes (from 10 x 20 thru' 25 x 48 to 25 x 80) and varying the delays generated after the cursor addressing, I can vary the time between invoking the editor and hanging occurring. By this I mean the number of characters typed. I would appreciate *any* help.

Yours truly, R T Lea, Sarawak, East Malaysia.

Ed. — Well, Mr Lea, you can either apply the modification suggested by Mr Gibson above, or you may like to get to the source of the problem. Version 1.0 of the IVC-MON (which was not in production for very long) did not support any nested escape sequences. Later releases, as well as including a number of enhancements, have been modified so that certain sequences, including cursor addressing, can accept nesting. The differences between releases of IVC-MON were published in 80-BUS News Volume 3, Issue 2, pages 48 and 49, in an interesting article giving an insight into the hardware and software design philosophy used in the development of the *Gemini* GM812 IVC, and its successor, the GM832 SVC. You should find, as should Mr Gibson, that replacing your IVC-MON V1.0 with any later release will eradicate your problem.

An IVC Problem?

I have recently come across an intriguing problem with *WordStar* when used on a system which incorporates IVC-MON 2.0 and the keyboard is attached to the GM812 video card. If the ESC key is pressed (as may be necessary if an error or interruption occurs), a List/Edit Function key message is produced - not much use if you haven't got the appropriate keyboard and a ★★★★ nuisance as well since the system may lock-up and often has to be cold booted to clear the problem.

I am currently running SYS and can have the keyboard attached to the CPU card or the IVC. By attaching the keyboard to the CPU card, the ESC key functions normally and *WordStar* does not become confused. If a version of SYS is not used, link 3 on the IVC will need to be adjusted so that the keyboard can be used on the CPU card.

Yours truly, Dr P D Coker, Orpington, Kent.

Ed. — the clue to your problem lies in the fact that the keyboard works OK on the CPU board, and not on the IVC. Firstly, why use it that way round anyway? I cannot think of any advantage to that approach, and a definite disadvantage is that you lose the type-ahead buffer. Secondly, I bet that you have a GM821 keyboard, and that you don't get the 'List/Edit ...' message until you have pressed the 'ESC' key twice, right? To explain what is happening I'll just give a little history on *Gemini* keyboards.

First of all came the GM821 keyboard (very originally known as the G613). This has 59 keys and can, with various permutations of Normal, Shift, Control, and Shift/Control, produce 128 ASCII codes, thus using 7 bits of the interface. The 8th bit is used as a strobe, and the keyboard can be connected to the keyboard interface on the GM811 CPU, GM812 IVC and GM832 SVC boards. When *Gemini* decided to produce the GM827 keyboard, with 87 keys, some of the keys were used to obviate the necessity for the Shift/Control permutations. In addition 30 keys were added to be user-definable separately in Normal and Shifted modes. These obviously require additional codes, but with only 7 bits available the question was 'How?'. The answer was to make them all double-byte codes, and to pick one existing code as the first byte 'key'. The one chosen was the ESC key, code 1BH. This key was therefore redefined as the first in the double-byte sequence, 1B 00, and all of the function keys carried on from there, F0 = 1B 01, F1 = 1B 02, ... and so on. The IVC-MON (Version 2.0 and later) was modified to include support for these codes. One of the user selectable links on the IVC (one of the switches on the SVC) determines whether the keyboard to be used is the GM821 or GM827/GM852. (The GM852 is a low-profile version of the GM827 with the only differences being that it is also available in a serial version, and that the very latest ones extend the user-defined keys into the Control mode as well.) If GM821 is selected then the code is passed on directly, if GM827/GM852 is selected then all codes are passed on directly unless a 1BH is received, in which case it waits for the next code and translates it to the user-defined string held in its workspace RAM.

And so back to Dr Coker's problem. If the IVC-MON (or SVC-MON) believes that the keyboard is a GM827/GM852, but it is in fact a GM821, then when the ESC key is pressed the keyboard outputs the code 1BH, which tells the IVC-MON that this is a special case, and that another code will be used to select a user-defined string. Consequently nothing will happen until another key is pressed, and depending on what this is will determine what the IVC/SVC actually passes onto the system. If this is the ESC key again (being pressed in the belief that the

first time was missed) then it just so happens that the string 1B 1B corresponds to the 'List/Edit ...' request.

For information and interest, let's just consider what would happen the other way round (i.e. keyboard is GM827/GM852, but IVC/SVC is set to GM821). The software is not looking for double-byte codes anymore, and so all function keys, plus the ESC key, will result in two characters being received, a 1BH plus another character. In particular, the ESC key will return 1B 00, and this will result in the correct action being taken for the ESC, but the 00 will put the system into Edit Mode. At the wrong time this may be disastrous! (N.B. The *Gemini* BIOSs have always allowed the Edit Mode code to be redefined if required, and the latest versions (V3.2 and later) allow it to be disabled altogether.

I hope that the above is useful (and understandable!).

BBC BASIC (Z80)

In the 'Letters to the Editor' in the July – August 1984 edition of 80-BUS News (Volume 3, Issue 4), mention was made by Chris Hellen of Colchester of a BBC BASIC (Z80) for Z-80 based microcomputers including those of 80-BUS construction. This BBC BASIC was written by R T Russell and marketed by *M-Tech* of Norwich.

Please may I ask if you know the full address of *M-Tech* of Norwich, from whom further details of BBC Basic (Z80) might be obtained?

Yours truly, Ian Manning, Bristol.

Ed. — Most *Gemini* dealers should be able to obtain this for you, and in particular I do know that *Off-Records* in London deal extensively with this product.

Pleasant Uncertainty

Dear Editor (etc, etc)

Hello, John. It's me again, ANGRY of Tonyrefail (remember!).

I've just read (and re-read) with great interest the VERY LATEST issue of 80-BUS News — July/Aug 1984.

What I want to know is, what's all this talk of printing my favourite 80-BUS oriented magazine on a 'regular-as-clockwork' type basis (Ed's comments, bottom of page 4). Look Tosh, if one wants one's reading matter delivered regularly, one could buy a subscription to *PCW* or *Practical Computing* or any one of the numerous other 'pulp' mags.

In my opinion, one of the best features of the good ol' 80-BUS is the sense of nerve-tingling expectation one gets from wondering just exactly WHEN the next issue will flop through the letter box. With other mags. one starts to think "July is here, I can hardly wait for this month's issue of XYZ magazine to arrive!". But when one has a subscription to YOUR particular rag, one usually thinks "July is here, I wonder WHEN the March issue of 80-BUS will arrive??!"

If you proceed with your threat to print 80-BUS News regularly, I will CANCEL my subscription!!

The regularity of 80-BUS is analogous to the general condition of most people's *Nascoms*, ie, "Will 80-BUS be printed this month?", and "Will my *Nascom* work when I switch it on" (sez it all really, doesn't it!).

Now to something totally different. Please inform Roger Dowling who's running a Users' Group for NasDos that I DEMAND an immediate retraction of his statement on Page 6 of this issue of 80-BUS — "The PolyDos Users Group only has 6 members". Let it be known to all, said Polydos Users Group now has a total membership of 8 (or 17 if you count everyone twice. Now I NEVER said maths was my strong point!). If Mr Dowling does not wish to retract his grossly inaccurate statement, please inform him that I am quite willing to be persuaded by a generous donation to my favourite bank account.

Oh heck, is that the time! If I hurry I'll just have enough time to flog a few peasants before lunch.

Yours truly, Dave 'Head-Crash' Richards, Tonyrefail, South Wales.

More Logic ?

Thank you for printing my article on *Lucas* in 80-BUS, Vol 3 No 4. Logic (Ilogic?) have sorted out my MDISK and XBASIC problem. I wrote to the General Manager (Mr Peter Seddon) who prodded the Engineering Manager (Mr Phil Purssell), who passed it on to John Phoenix. This last (untitled!) chap updated my XBASIC master disk for me, and MDISK under XBASIC is very nice, thank you very much. But who pays for all my postage?

The moral of the story:-

> If you're having troubles in an illogical way
> And your letters are being chucked into the bay
> Don't mess about with the bottom of the tip
> Climb every mountain
> And give Peter Seddon a lot of lip

Apologies to Mr Seddon, but it's been a year of nearly wasted programming!

Yours truly, Dr David Plews, Keighley, W Yorks.

CP/M User Group

I would like to echo David Plews' comments in praise of the CPMUGUK (Vol 3, Issue 4 of 80-BUS News). I joined some months ago and soon found their journal essential reading (as is 80-BUS News of course!). The quantity of virtually free software is incredible. I haven't even read all the index yet! The service from the 'Librarian' (Derek Fordred) is excellent; I normally receive disks within 9 days of posting off my request.

I have just received a graphics suite called PLOT 33 on SIG/M Volume 194. This suite allows the production of pictures, graphs, bar charts, etc, on my *Epson* MX80 type II. It can be driven from MBASIC, TURBO PASCAL and FORTRAN. It is extremely well documented — approximately 25 pages. It is easily configured to any of the popular dot matrix printers with addressable dot graphics. When I get the hang of it a bit better, I'll try and write a review for 80-BUS News.

On another subject, the free ad. I placed in Vol 3, Iss 4 worked — I sold my PCG within a week of publication. Dr Plews may do better from his ad. in the same issue if it has contained his correct phone number! Ed. — Dr Plews has sent us his correct number, it is (they are!) Day 0535 52511, Evening 0535 54157. I have so far failed to contact him. By the way, I may be able to help him in dumping AVC pictures to his printer — I have unravelled GDUMP from *Lucas Logic* and modified it to operate as a subroutine to be called from Basic, etc,. I have also relocated it.

In addition, I have written routines to dump the AVC image to my *Epson* the right way up and smaller than the sideways picture produced by GDUMP. Any of these should be easily adapted to drive David's *Gemini* 10X.

Now a quick plug — these "dump" programs and many others are currently circulating on the NASDOS USERS GROUP DISKS (yes, two!).

Yours truly, Colin Case, Rugby.

THE DH BITS

by Dave Hunt

In this section of miscellaneous ramblings, the always verbose David Hunt looks at copying files via dBASEII, changes that have occurred in different versions of dBASEII, disk sector skewing, disk blocking/deblocking, how to expand a *Nascom* based system, a tip on using the Pretzel program, and a look at the Gateway and Pathway programs.

Speeding up dBASE IIusing RAM-disks

Following on from my other article concerning adding machine code patches to dBASEII, a new routine has had to be written lately. So this bit is for dBASE aficionados, although it contains a warning for all who would write simple file copying routines.

Now those who know dBASE will know the geriatric performance it puts in when indexing files with a good few thousand records. Far too slow on a floppy disk, somewhat better on a Winnie and just reasonable on a RAM disk. Further, versions of dBASE earlier than V2.41 get very slow when updating files with multiple indexes.

An aside about dBASE V2.41

By the way, dBASE V2.41 has what appears to be horrific change of philosophy which I believe has led to its rapid replacement with the current version, V2.43, so anyone with version 2.41, watch out when trying the following:

```
@ x,y GET input
READ
FIND &input
DO WHILE input=record
(... do your thing ...)
SKIP
ENDDO
```

What all versions of dBASE, except V2.41 do, is that the FIND function finds the first occurrence of the find criteria, whilst the DO WHILE - SKIP loop then finds all following entries where the input and record criteria match, the loop cops out when the criteria no longer match. Basically it works through a matching list in order of entry until it no longer matches. V2.41 is something different. Using the same routine, the FIND will find the first occurrence as before, but for some unexplained reason, it then sets the record pointer to the end of the list, so the following DO WHILE - SKIP loop will immediately find the next record does not match and cop out. The way round it is to rewrite the program:

```
@ x,y GET input
READ
FIND &input
DO WHILE input=record
(... do your thing ...)
SKIP -1
ENDDO
```

In this case it appears to almost work backwards through the list (almost, as the first record found is still the first in the list), not a disaster, and in many ways more convenient as it's more often than not the last in the list you're looking for, not the first. But what about all the programs written for other versions of dBASE, they're incompatible!!

To tell the truth I didn't find this, Trevor at *ACC* did, when it screwed up an invoicing program I had written. He phoned me to discover what special feature it was that I'd added to the latest version of the program that didn't work. I started investigating! I assumed this was a new feature of V2.41, and you now had the option for either backwards or forwards search (very useful), but I could find no reference to it in the manual and no way of switching back to 'normal'. As I said, V2.43 followed hot on the heals of V2.41, and that works 'normally', so perhaps the V2.41 anomaly was considered as a bug. To be fair, I only had one copy of V2.41 to test, and this was replaced with V2.43, so perhaps it was a corrupt disk or something, although I've yet to hear of a corrupt disk causing a complete change in philosophy, corrupt disks usually crash.

And so back to the indexing problem.

Now lets take the real life situation, it's a stock control program working on a *Gemini* with a 10M Winnie. The main stock file has something like 2000 records in it and the transaction file has about 6000 and increases with each stock movement. The stock file has three indexes all on at once, whilst the transaction record is indexed by date and the paperwork reference number. Believe me there's some index churning going on. Given that indexing is pretty fast in a RAM disk, the problem is getting the indexes there safely.

One way round is to copy the files to the RAM disk before operating the dBASE program, and then copying them back on close down, you'll away remember to do it, won't you??!!? But what about the idiots who usually use the program, they'll either forget to copy the indexes first in which case the programs won't work, or worse, they'll forget to copy them back when they've finished. Another way would be to reindex the programs onto the RAM disk each time the program's started, Ok, and no need to copy the files back. But even with a RAM disk, the reindexing process takes some time. I suppose the whole program could be run under a

SUBMIT file, to PIP the indexes back and forth, but I don't like SUBMIT files for uneducated use, and anyway, they're messy. No the answer is to make dBASE do the job as part of the program.

Now dBASE can copy data files around with impunity, and it can create new files all over the place, but can it copy index files from one drive to another, no way! The same applies to .CMD, .FMT and .MEM files as well, but I can't see why anyone would want to copy these.

The simplest answer is a 'mini-PIP' program which will transfer the indexes to the RAM disk and back again. A very simple little program was written to open an input file, open an output file and copy it sector by sector. Dead simple you might think! Let's have a look at the snags which could befall the unwary.

We need two file control block for a start, and because the program is to be used generally, we need a space to transfer the names of the files to be copied into. No problem there. You can see from the first bit of the progam the 'to' drive name, and the 'from' drive name and file name space. I put them at the front of the program as that's easy to find when POKEing around to fill in the names. The POKE location for the names is always three bytes in from the start of the program no matter how I hack the program around and change its length. Note that there is no check to see if you're trying to write to the same drive. As the program is going to get it's parameters from the dBASE program, and no user intervention is required, that sort of mistake can't happen. Can it?

The program starts off by clearing the space behind the 'from' name as this is the file control block for the following 'search for first'. If you forget to clear this fcb, then CP/M does strange things with the 'search for first'.

The 'search for first' is to check for the presence of the named files, if they're not found, then the program returns home immediately. Here comes the first snag! The 'search for first' returns 0ffh if there are no files present, otherwise, it copies the sector of the directory where it found the first entry into the current DMA workspace and returns with A set to the number of the entry in that sector. Checking for 0ffh is dead easy, anything else in A assumes that at least one file was found. Having established the presence of the files, where the heck is the current DMA, as dBASE, sure as eggs is eggs, didn't leave it at the default DMA of 0080h. We need this address as we have to copy the file name we have found to the file workspace buffer, as we intend to make a list of all the files required before we start the copy process.

A little investigation revealed that different versions of dBASE leave the DMA at different addresses. All around 98XXh, but not consistent. So putting a fixed address into the program is not on. The next thought was that CP/M must know where the DMA address is, but there's no command to return it. Perhaps it could be found in a CP/M workspace someplace? That way I could cheat and calculate a fixed offset from some known location within CP/M. I looked, it appeared twice, and both in the BIOS. Now the notorious thing about BIOSs is that each version is different, so odds on that the DMA pointers will move around between versions, and anyway, which one of the two pointers to use? No forget that approach. The last resort was the obvious, set the DMA pointer myself. My objection to this was, what would dBASE do? Without disassembling large chunks of it, there was (and still is) no way I could be sure that dBASE would reset the DMA pointer correctly before using one of it's overlays, or whatever. Or whether in fact this would matter. I took a chance and set the DMA to 0080h. dBASE continued to work, and so far nothing odd appears to have happened. But I'm still unhappy about it.

So before the 'search for first', there is a 'set DMA' so I know where to retrieve the data.

Unfortunately, having found a file name, you can't use it immediately to copy the file to the new drive, as the copy process causes the 'search for first' — 'search for next' to forget where it was, so the file names must be found first and stored in a workspace for copying later. Having found the first entry, the position in the DMA buffer is calculated and the name found copied to the file workspace buffer. The program drops through to the 'search for next' and loops until all the names are in the buffer. So I know where the end of the buffer is, I stuck a null on the end of it.

The next bit was a piece of cake. Copy the first name found into the first and second fcb's, open the files and read a sector from the input file into the DMA and then write the DMA out to the output file. When the file was all copied, close the output file and pick up the next name in the list and do it all again until the name in the list was a null, indicating the end of the list. Simple and very quick.

I didn't bother to check the validity of the files as Winnie to RAM disk copies are always reliable and anyway what could I do if there was an error? The whole process was quite quick, shovelling about 200K of indexes in about 30 seconds.

From the foregoing, you can see I'm all for the KISS philosophy. KISS, 'Keep It Simple, Stupid' on the grounds that if it's simple it should work Ok, anyway, if it didn't then it should be a doddle to fix.

Having got the routine working, a thought occurred! Why not make the routine make the backup copy disks as the machine shuts down. That's about 650K from Winnie to disk. I tried it. It took 45 minutes!!! A very much confused DH sat down to

think about this one. Why did it only take 30 seconds to copy 200k from Winnie to RAM disk (and a little bit longer the other way round) and yet 45 minutes to copy 650 odd K from Winnie to disk?

The answer was very simple and easily forgotten, and goes way way back to the first disk systems for *Nascom* and *Gemini* (remember I had a finger in that pie all those years back). It's called disk skewing.

All you ever needed to know about disk skew, and were afraid to ask!!

More often than not, customers new to *Nascom* and *Gemini* disk systems using the *Gemini* CP/M ask what the format routine means when it asks for a skew factor. Now I don't know what the *Gemini* manuals say about skewing as, in common with most users, I haven't read them. But the point is easy to grasp once you know what's going on.

A virgin disk straight out of the box has nothing recorded on it so the first thing is to format the disk. This defines the way in which the data will later be stored on the disk. Think about what the format program does when it sets up the disk. The format program constructs an image of a disk track in RAM and then writes the image to the disk, a complete track at a time. The disk contains a number of tracks starting at track 0 at the outer edge and working inwards, track 1, track 2, etc., to the innermost track of the disk, be it track 34, track 39, track 76 or 80, depending upon the flavour of drive.

Now the disk is also split like the slices of a cake, radial lines from the centre of the disk to the outside. Where the lines cross the tracks, these become sectors, so each track is sliced up into a number of sectors. The data area of each sector will usually be in multiples of 128 bytes, 128, 256, 512 or 1024. Rarely more than this for reasons to be revealed. The first sector is indicated by the index hole in the disk. Note I said the data area of each sector, as a sector contains more than just the data bytes. The disk controller needs to be synchronized with the data on the disk, so the start of each sector contains some synchronization bytes. Next the controller likes to know were it's at, so there are a few bytes saying the track and sector numbers (and also the 'sides' byte, saying what side of the disk this is, a real pain that one). This is followed by a checksum, the header CRC. Then a few more sync bytes and the start of data mark. After all that, then comes the data, the format program sets these to 0e5h, followed by another checksum, the data CRC. To polish the sector off there are a few more bytes, the sector gap, usually some 0ffh's before the start of the next sector. The last sector perhaps has a few more 0ffh's on the end, to pad the sector up to the index hole, as you can never be too precise as to the actual exact number of bytes on a track, as the whole process is at the mercy of the vagaries of the speed constancy of the disk drive motor and the controller clocking speed. Putting too few sectors on a disk is preferable to too many, as the format program which sets all this up in the first place starts and stops writing when it sees the index hole. Too much data and the last sector will be incomplete causing fatal disk errors. (This error occurs when formatting disks on a cold day with the drives running slower than usual.)

From all that, you can see the incentive is to make as few sectors per track as possible. The fewer the sectors the less space wasted on sync and track and sector headers, et al, and the more real data space available. So why rarely more than 1024 bytes in a sector? Well the disk controller is a phase locked loop device, and when receiving data (as opposed to sync bytes which it can recognize) the data transitions should appear in the right places keeping the loop in lock, but mechanical problems like sticky disks, cold (or hot) drives, etc., tend to cause the data transitions to occur out of step. The phase lock tries, but there must come a time when it slips a 'bit'. The reliability of the system depends on the mechanical stability of the drive and media and the electrical stability of the controller phase lock. Tolerances in either direction must be allowed for, so the best way of keeping the whole thing in step is frequent sync pulses, which from a data storage point of view are a waste of space. So the trade-off is the number of sync pulses, hence the number of sectors, against the ability of the system to stay synchronized. For good reliabilty, this limits it to sectors of no more than 1024 data bytes. No doubt there are those who will argue that their machine can reliably read a whole track at a time. I don't doubt it, but what about the next machine?

So having demolished the way a disk is mapped, what about this business of 'skew'? Consider! CP/M is about to load a program. CP/M has a map of the disk internally (it's stored as part of the directory information). The map CP/M has is not the same as the disk map constructed by the format program. Simply it consists of blocks numbered in sequence from the end of the system tracks and directory space, block 0 might be at the start of a track, or could even be someway through a track. Blocks may be 1, 2 or 4K in size, and CP/M makes a calculation considering the sector size of the disk, and then adding the system track and directory space offset to determine the appropriate track and sector. For the sake of example, CP/M has calculated that the first track/sector of the program is 4/1. This information is passed to the disk controller to fetch track 4 sector 1. The disk head steps to the correct track, reads data until it recognizes the sync bytes, and reads the next sector header it sees. This header will tell it it's on the right track (or not), and as this happened at some random time, like as not, the sector number will be wrong. The disk, of course, continues to

rotate, until some time later, the correct track/sector header will pass the head. At this instant, the controller looks for the data start mark, and when that's found it then reads the following data.

Having got the data, control is passed back to CP/M which now has to calculate the next track/sector required. CP/M decides that the track/sector will be 4/2, the next sector. An instruction is issued to the controller, and the previous process is repeated, except that the controller now knows that it is on track 4 (the last sector was on track 4, and nothing has changed, so the head must still be on track 4). Unfortunately, whilst all this was going on, the disk of course continued to rotate, so by the time the system has made up its mind that it is sector 2 it is looking for, the header for sector 2 has just passed the head. This means that the disk must make a full rotation before sector 2 again passes the head. As the disk rotates at 300 rpm, that means that 200mS must elapse before the next sector can be read. Considering that a typical *Gemini* system reads a 512 byte sector in 20mS, this hanging about is a real pain. It would take ten times as long to load a program under these circumstances as it would if we arrange the sectors to be in the correct places when CP/M wanted them.

So we introduce 'skew' in to the format program, we alter the way the sectors are numbered on the disk, on a *Gemini* system, a 'straight' skew of:

```
Sector number   1  2  3  4  5  6  7  8  9  10

                    becomes something like

Sector number   1  6  2  7  3  8  4  9  5  10
```

I say something like this, as I haven't bothered to look at the actual *Gemini* skew, so I might have it slightly out, but no matter. This is a 'skew' of 2, that is the sectors appear in order in 2's, read one, skip one, and so on. See what happens, having read sector 1, and then requiring sector 2, the longest wait would be 20mS whilst sector 6 went past. Plenty of time to calculate the next track/sector as being sector 2.

Lets suppose that some program we have written is required to read the disk and actually does some preprocessing between sector reads, or that the system clock is running at 2MHz rather than 4MHz. Then things slow up again as even with a skew of 2, the 20mS between consecutively numbered sectors is not long enough. In that case, to restore the speed to something like normal, a skew of 3 may be required.

So by now the business of skewing the disk has been taken care off, the only remaining question is if this problem has been known for a long time, why didn't *Digital Research* do something to CP/M to correct the situation instead of skewing the disks on formatting? The answer is that they did, right from the start. This is called 'logical to physical sector translation'. Inside CP/M is the sector translation table. It assumes that the disk is formatted in 1, 2, 3, ... order, and therefore the CP/M calculation to get a certain logical sector must be translated into the appropriate physical sector.

Lets take the case of a skew of 2 illustrated above. CP/M calculates it wants logical sector 1, so it goes to the 'logical – physical sector translation' table and looks up logical sector 1 which comes back as physical sector 1. For the next sector, it calculates logical sector 2 which is translated into physical sector 3, logical sector 3 becomes physical sector 5, and so on. There are a couple of good reasons for not using the 'logical to physical sector translation' one is that you can't alter it easily. But worse, if the disk were transported to another machine with a different 'translation' the sectors would be read in the wrong order, so data would become garbage and programs would crash. The advantages of using physical disk skewing is that you have control of it and even if a disk were transported to a different machine and the skew were wrong, then the disk would read slowly, but it would get there eventually and get it right.

The very early *Heneled/Gemini* GM805 had a 'logical to physical translation' of 6 which was originally intended for 8" disk systems. (I think it was built in permanently to CP/M 1.4, so couldn't be changed to something more appropriate.) This made the GM805 system slow, and I remember sitting down one evening with Eddie Pounce and playing with strips of cardboard numbered in sectors and sliding one against the other until an optimum skew was arrived at. As I remember, it ended up as a negative number, as we already had a positive skew of 6, we needed to skew the disk backwards to improve matters. The skew tables for 2 and 4MHz were then added into the format routine written by Richard Cowdroy supplied with the system to become Version 2.0 of the formatter. All great fun when you have both physical skewing and 'logical to physical sector translation'.

Back to the problem

So, back to the beginning, how come my stupid copy routine took so long to write out its files to disk. Simple, the timing was all to pot. But not by the obvious skewing problem of the routine taking too much time between sectors. It was in fact a whole different can of worms called 'sector deblocking', something I'll hold over until another time, but brief explanation is required.

CP/M was originally designed for use with 8" systems conforming to IBM3740 spec. The disks had physical 128 byte sectors, so it was sensible to make CP/M work internally with 128 byte sectors.

With the advent of high capacity disk systems, and the need to make the sectors larger, CP/M had a problem as physical sectors were no longer 128 bytes long. CP/M 2.2 introduced this idea called sector deblocking which overcomes the problem of different systems with different size sectors, at the same time leaving CP/M internally compatible with 128 byte sectors, which CP/M programs still use (compatibility between CP/M 1.4 and CP/M 2.2). CP/M now carries out a two stage disk access. Firstly it calculates the start of physical track/sector containing the 128 bytes it's after. This n-length sector (512 bytes in the case of *Gemini*) is read into a work space called the deblocking buffer. Next CP/M calculates the place in the buffer where the 128 byte sector resides and copies it to the DMA address as if the 128 bytes came straight from the disk in a old CP/M system (or one which still uses physical 128 byte sectors).

The problem was I was reading and writing 128 byte logical 'sectors' (in inverted commas, as the *Gemini* physical sector size is 512 bytes). The process for one single 128 byte 'sector' goes like this:

READ (from Winnie)

CP/M calculates the appropriate physical track/sector where the 128 bytes required are situated.
Wait for that sector to appear (very fast as the Winnie controller probably knows where it is).
Read in that physical sector to the deblocking buffer.
Calculate the position of the 128 byte logical sector.
Copy the 128 bytes required to the DMA area.

WRITE (to disk)

CP/M calculates the appropriate 512 byte physical track/sector where the 128 bytes are going to go.
Wait for that sector to appear (up to 199mS delay)
Read that physical sector into the deblocking buffer.
Now write the 128 byte logical sector from the DMA buffer into the deblocking buffer.
Wait for the physical 512 byte sector to appear again (200mS delay)
Write out the deblocking buffer to the 512 byte physical sector on the disk.

So you can see, there might be the best part of half a second between consecutive 128 byte sectors, and there's an awful lot of 128 byte sectors in 650 odd K!! OK, so conversely, how come it worked so well between the Winnie and the RAM disk. Again, simple. In the case of the RAM disk, the thing is RAM, so no delays in waiting for sectors to come past, and it's configured in 128 byte sectors anyway (I think). Just calculate the address and shove the data at it. The Winnie is less obvious. It's all to do with the Xebec controller fitted. The real purpose of the Xebec is to act as a high speed interface between the Winnie which reels data in and out at mega-bits per second, and the computer which is quite pedestrian by comparison, only working in hundreds of kilo-bits per second. The Xebec contains a large RAM buffer for the Winnie to work in, so shovelling 128 bytes at a time is just like talking to a RAM disk. The Xebec's own processors (two wierd dedicated things and a Z80) and RAM take over after the data has been captured, so again, no hanging about for sectors to appear in the right place before a read or write occurs.

So how could I speed up the Winnie to disk copy routine? Well the cause of the problem is this swapping of data about in the deblocking buffer. The deblocking logic is quite clever, although if you've read the source, quite messy. It knows not to flush the buffer if the same physical track/sector information is still current and no data write has been made to it. If a write to it has been made, then is still doesn't write whilst the same physical track/sector remains current. However, as soon as the physical track/sector is changed, or the file is opened or closed, then the buffer is flushed if no writes were made, or written out then flushed if writes were made to it.

In a sequential read or write situation then, the deblocking knows what's going on, so assuming that a write of a newly created sector was to be made, the 512 bytes of data is accumulated in the deblock buffer, and when complete, the whole physical sector is written out without any delay caused by reading in the sector and then writing it out. Likewise on a read, reading sequentially means that four 128 byte logical sectors are read into the deblock buffer and processed one at a time, the next read would be of the next 512 physical byte sector. (On a *Nascom* or *Gemini* the normal 'skew' of 2 allows this to happen efficiently.) So the answer would be not to read in 128 bytes then write it out again, but to read in at least 512 bytes and then to write it out again. In fact 512 bytes is not really enough. Better still to read in as much as possible into the TPA, say from the end of the copy program to the base to CCP, and then to write it out as a lump. This is how SWEEP or PIP work (have you *Gemini* MultiNet network owners noticed that PIP seems slower with the smaller TPA on the server using the 24K network system than when using the 64K normal system?).

I didn't bother to do this with the 'mini-PIP' for transfering the indexes about, as the calculation of available TPA space and the complications of reading and writing buffer-fulls of data wasn't worth the effort. Apart from that it meant more 'decimalized' POKE space within the dBASE program, making the thing more difficult to patch by

hand if need be. The program fulfilled its main purpose, and the backup disks are made using the separate BACKUP program which had been used previously.

***Nascom* — Where now?**

Fairly regularly people walk in the shop and say, "I've got an old *Nascom* 2 and don't know whether to get rid of it or spend some loot and upgrade it.". Well two things are apparent, firstly there are a lot of *Nascom* 2's still around (there should be, as nearly 10,000 were made before Lucas got there paws on the show), and secondly a lot of people are still dedicated to the machine which in all probability they sweated blood over building some three or four years back.

The problem is doing anything with a *Nascom* works out expensive. Possibly the cheapest way out is to flog the *Nascom* to some unsuspecting mug and then go and buy a *Beeb*, a *Spectrum*, a *Commodore* or some other such Mickey Mouse computer. But then none of these has the potential of an expanded *Nascom*, despite the plethora of add-on bits you can get for them.

All the following comments apply equally to the *Nascom* 1, but remember that for optimum results the N1 should be capable of running at 4MHz (although the disk systems will run as slow as 2MHz with no wait states) and the N1 should have a decent buffer board. The original *Nascom* buffer board wasn't too clever! The *Gemini* GM806AK combined backplane-cum-buffer board is fine. (This IS still available, so don't believe any dealer who says he can't get it for you anymore.)

The only viable upgrade of a *Nascom* is towards disks, a goodly few have already taken that course, and I think those who have gone that way are happy. Expensive certainly, but still worth the money as the end result still costs less than going out and buying an equivalent machine from new. However, even this route is becoming more circumscribed as Lucas seem to contract their *Nascom* operation and *Gemini* have reduced the number of options available. MAP are still around and in there somewhere, but I haven't heard from them for some time.

All permutations would need at least 48K of RAM and all would require disk drives. The drives would be either single or double sided 80 track drives. The drive boxes available from the various 80-BUS manufacturers are all comparitively expensive, and cheaper equivalents can be found advertised as options for the Beeb computer. But watch the specs. Check that what you are buying does or does not include power supplies (as required) and that 40/80 track types are really 80 track types and not some 'cod' used by the various DOSes for the Beeb.

The basic choices left are:

1) *Nascom* NASDOS with the *Nascom* controller card, with or without the AVC card.
2) *Nascom* CP/M with the *Nascom* disk controller and AVC cards.
3) *Gemini* CP/M with the *Gemini* disk controller and IVC or SVC cards.
4) MAP CP/M with the MAP VFC disk controller/video card.
5) Careful permutations of 3 and 4.

Choice 1) Uses the original *Nascom* screen display and needs the *Nascom* disk controller card. It may well not be the best choice, for although NASDOS allows the use of your original *Nascom* software, little if any new software has been made available.

Choice 2) Uses the original *Nascom* screen display and needs the *Nascom* disk controller card. Not a bad choice, CP/M 2.2 seems to be the route to take. The *Nascom* CP/M uses the *Nascom* AVC as either a colour display or as an 80 column display (very necessary for CP/M). The *Nascom* AVC is a bit slow in the 80 column mode.

The problem with either choices 1) or 2) is likely to be support. Most of *Nascoms'* old faithful dealers now no longer stock *Nascom* products and Lucas do not seem to be encouraging new dealers. The only dealer (as far as I am aware) able to offer full support for *Nascom* products these days is *B & L Micros* at Kenilworth. So unless you live in the Midlands, the *Nascom* options might not be advisable.

Choice 3) Uses the *Gemini* GM829 disk controller card or the (now obsolete) GM809 card. Possibly the most universal choice. There were more *Nascom/Gemini* hybrids in the recent 80BUS survey than anything else except straight *Gemini* systems. *Gemini* have recently reduced the permutations of CP/M 2.2 for *Nascom*, so the only version currently available uses single or double sided 80 track drives and must use either the *Gemini* SVC or the (now obsolete) IVC card. This makes it the most expensive choice, but it wins on both overall operating speed and disk capacity. You could save money by using secondhand GM809 and GM812 cards as these still seem to be around in small quantities from people who have upgraded to GM829 and GM832 respectively.

Choice 4) Uses the MAP VFC combined video and disk controller card. Available with either CP/M 2.2 or CP/M 3. (None of the other manufacturers have opted for CP/M 3 as there is little to show for the increased cost.) About the same price as the straight *Gemini* choice using secondhand cards.

Choice 5) Careful permutations of *Gemini* and MAP cards with a suitable choice of CP/M (*Gemini* or MAP, depending upon the exact permutation) will work but there seems no advantage in opting for a

permutation unless there is some already existing reason for making this sort of hybrid.

The choices are more more limited than they were (no 35/40 track drive options for instance) but still allow the same degree of system flexibility. CP/M means that there is acres of software available (both proprietory new stuff at high prices and the Curates Egg which can be aquired from all sorts of places (CP/M User Group et al) at very modest prices).

So an upgraded *Nascom* is still a very much viable proposition, what you end up with might appear a bit odd (no flashy box) but still good value for money, despite the exhortations from the various big manufacturers to buy 16 bit machines. To me there seems little point in ditching the dear old Z80 based systems as the availability and cost of good 16 bit software is not justified for home use and still very expensive for business use. There is very little in the way of all the useful utilities which are available for CP/M 80 (as opposed to CP/M 86 or MSDOS). Anyway, most of the currently available 16 bit software seems to extremely wasteful of RAM space and very very slow when compared to its 8 bit equivalent. So much for the increased crunching power of 16 bit processors, not many of the proprietory software sources seem to be interested in using it.

Little tip for PRETZEL 2 users.

Rudd Thornton of Largs writes "Having saved Prestel pages with PRETZEL 2 how to get them printed as PRETZEL 2 saves the PRESTEL screen image which only contains carriage returns, not CR/LF's? The answer is WORDSTAR.

```
Load up              >WS FUN.PGE ↩
Find and replace     ^QA
What?                ^P ↩ ↩
                (Two returns; '^P ↩' is
                the WORDSTAR overtype or CR
                without LF = 0dh)
With?                ^N ↩
                (Hard carriage return = 0dh,0ah)
Options?             GN ↩
                (Globally, no questions asked)
```

The new FUN.PGE will print correctly with PIP or under WS.

This is a simple little tweak and will work with wordy pages (but of course no graphics). PRETZEL 2 now has a graphics printer option for those interested, which will work with any *Epson* compatible printer, it costs £11.50 from *Henry's*.

The Gateway

Peter Curtis, the man who wrote the original NASPEN, GEMPEN and DISKPEN text processors has recently 'finished' his latest creation called 'Gateway' with its companion program 'Pathway'. I say 'finished' as it has been released on an unsuspecting public in it's current form to see what reaction it creates. There's bound to be some user feedback and he hopes to produce a final version soon (with free updates to those who bought the earlier versions). Unlike Peter's earlier programs this program is not based on a simple idea, rather it is an attempt to make a simple job of an extremely complex idea. In the main it succeeds. Gateway is not system dependent, it will work on any CP/M 80 machine.

The basic idea of Gateway is to allow you to tag acres of text, turning it into a database without the limitation of fields or other boundaries. Searches take place over lines, paragraphs or the chapters or even the whole file, with the ability to work within user inserted boundaries in the form of default keys inserted into the text. The tagging can be automatic or stop for user intervention, and the tagging may take place using logical operators, AND, OR or NOT or on straight searches. This allows relational searching to be achieved, cross referencing things within the text. When I say text, Gateway can handle diskfuls of text at once, across large numbers of files. So a Winnie full of technical instruction manuals is not outside its scope.

During the search and tagging process, Gateway builds up its own 'tag' and 'directory' files telling it where it found what it was looking for. These are then used to sort and extract data as required, shifting the text into different orders, taking related extracts, merging bits from different files into one, and so on. The original text files remain untouched.

The Gateway defaults (and any changes to the defaults) are stored in a small file which Gateway constructs for itself called a XXX.LNK file. These defaults allow the user to swop jobs or to leave the system and then pick up work at the point where it was left. XXX.LNK files are constructed for each job, along with the 'tag' and 'directory' files for the job in hand. So different work can be carried out on the same text if desired.

Gateway itself is not too difficult to drive, it has a very informative 'Help' facility which is a Gateway cross-referenced file itself. This is coupled with an intelligent auto prompt, which prompts in full for inexperienced users or omits the prompt options for the more experienced. The intelligent bit is that the prompt selects the next most likely command within the context of what the program was doing. In most cases a series of returns is all that is required to drive it. Of course you can break out of the automatic sequence at any time, simply by entering a new command instead of accepting the command offered. Most command inputs occur at the bottom of the screen, the remainder of the screen being used for text display. When Gateway finds what it's looking for, the text is displayed with

the relevant part highlighted in inverse video. Little concession is made to the cleverness of the *Gemini* screen addressing, as the program is intended for use with CP/M machines in general. To this end the screen is treated in the simplest fashion consistent with tidiness and readability.

I'm slowly working through all the 80-BUS stuff I've got on disk, so next time, instead of my saying 'I think it was written up somewhere' I'll be able to say exactly what, when and where! That's some 1.8M bytes on three disks covering about 100 files. Mind you, it'll take some time, as I have to decide the search criteria and introduce the search keys as required. A further problem is that I don't have all of the past 80-BUSes, only the parts I've written or edited, Paul has the rest, if he hasn't reformatted and re-used the disks.

The problem is what to make Gateway look for! As, for instance, in the case of letters files, Gateway doesn't know that CEGB and electricity are related until you do a cross search for either or both. This problem is eased a little within Gateway by an option which says 'ignore' case, so 'Central Electricity Generating Board' and the word 'ELECTRICITY' could be tied up without further intervention. But you still have to know what way you want the data presented and how to find it in the first place. In some ways Gateway could be likened to the early days of the laser, a solution looking for a problem. It needs imagination to figure out what you want Gateway to do, once you've decided, Gateway can do it.

The Pathway

The companion program to Gateway, Pathway, is something else. Parts of it have immediate uses. Now most of you will be aware that not all text processors write straight ASCII text to a file. WORDSTAR for instance riddles the text with 'bit 7' set and uses only line feeds instead of CR/LF's within paragraphs. If you use the CP/M 'TYPE' command (without CCPZ) on a piece of WORDSTAR text, you'll see what I mean. Other text processors leave text controls lying about to a greater or lesser extent.

Gateway will handle text from any source, but merging different bits of text from different processors leaves a problem for the text processor ultimately used to edit the result. Ideally Gateway would like standard ASCII text, or at least, all the same type of text, but conversion from one source to another is a bit of a pain. CONVERT, part of Pathway does just that. It has three tables, the input conversion table, the output conversion table and a video conversion table. Each table consists of 256 comparison strings. The data is read from an input file and all incoming characters are compared with the input table and are either left unchanged, converted to another character or converted to an n-length string. The idea is to convert all input to a given standard, say ASCII. Likewise, on the way out to the output file, all characters are compared with the output table and similar conversions can take place. A third table may be used for the output to convert video control strings which might be used for special functions on some video terminals or printers.

The main purpose of Pathway is to format and print text without the intervention of a text processor. The text would (normally) come from Gateway output files, but any text may be used. The input and output conversion tables used by CONVERT can be invoked, so special print controls may be inserted as required. Pathway also accepts an extensive set of 'dot' commands within the text, akin to WORDSTAR, for such things as line length, page length, headings, footings, margins and many more. Unfortunately these commands are not exactly compatible with those used by WORDSTAR. The net result is neatly formatted text, to suit the printer in use from any old source file regardless of the original format.

Gateway and Pathway are all clever stuff, and represent a different and very efficient way of dealing with large amounts of text identification and retrieval. I suppose it would really come into its own for use with disk versions of the Encyclopædia Brittanica or for cataloguing libraries or some such. Using it to cross index my letters files and on my bits of 80-BUS text seems a trivial occupation for a program of the obvious power of Gateway. I feel a little intimidated by the concept of the program. Perhaps I feel inadequate because I can't think of a really good job for it to get to grips with. I like it, I use it (it's not difficult), I just find it difficult to enthuse over it.

Private Advertisements

Items Wanted

RAM A board or basic *Nascom* 2 system; RAM B considered. Eric Wright, Newcastle 091 285 3762.

Cheap *Nascom* 2, preferably without PSU, keyboard or memory. Condition not important provided it hasn't been jumped on or excessively butchered, and has most or all of its ICs. Tel: Dr P D Coker, (Orpington, Kent.) 0689 58510.

```
Index file loader for Dbase     M-80    17 May 1985     21:53   PAGE    1
                        Title   Index file loader for Dbase

                                .z80
0000'                           aseg

                        ; CP/M equates
0005                    bdos    equ     0005h
0080                    dma     equ     0080h
000F                    opnfil  equ     15
0010                    clsfil  equ     16
0011                    srchfst equ     17
0012                    srchnxt equ     18
0013                    delfil  equ     19
0014                    rdfil   equ     20
0015                    wrfil   equ     21
0016                    makfil  equ     22
001A                    setdma  equ     26

                                org     100h
                                .phase  0c000h

C000    C3 C04A                 jp      start
                        ; Workspace
C003                    dstdrv: defs    1           ; Destination drive name

                                                    ; Source fcb
C004                    fcbl:   defs    1           ; Source drive name ..
C005                    fcbla:  defs    11          ; .. file name mask ..
C010                    fcblb:  defs    23          ; .. rest of source fcb
C027                    fcb2:   defs    35          ; Destination fcb

                        ; Find the files to copy
C04A    11 0080         start:  ld      de,dma      ; Set the DMA address
C04D    0E 1A                   ld      c,setdma
C04F    CD 0005                 call    bdos
C052    CD C0C0                 call    clr
C055    11 C004                 ld      de,fcbl     ; Search for first
C058    0E 11                   ld      c,srchfst
C05A    CD 0005                 call    bdos
C05D    FE FF                   cp      0ffh        ; Quit if not found
C05F    C8                      ret     z
C060    CD C0B3                 call    loc         ; Put it in save
C063    11 C10D                 ld      de,save
C066    01 000B                 ld      bc,11
C069    ED B0                   ldir
C06B    D5                      push    de

C06C    11 C004         srch:   ld      de,fcbl     ; Search for next
C06F    0E 12                   ld      c,srchnxt
C071    CD 0005                 call    bdos
C074    FE FF                   cp      0ffh        ; No more ?
C076    28 0C                   jr      z,schdon
C078    CD C0B3                 call    loc         ; Put it in save
C07B    D1                      pop     de
```

```
Index file loader for Dbase     M-80    17 May 1985     21:53   PAGE    1-1

C07C    01 000B             ld      bc,11
C07F    ED B0               ldir
C081    D5                  push    de
C082    18 E8               jr      srch

C084    D1          schdon: pop     de          ; Mark the end of the list
C085    AF                  xor     a
C086    12                  ld      (de),a

                    ; Now copy out the files
C087    21 C10D             ld      hl,save     ; Point to the list
C08A    E5                  push    hl

C08B    CD C0C0     copylp: call    clr         ; Clear the fcbs
C08E    3A C003             ld      a,(dstdrv)  ; Copy dest. drive to fcb2
C091    32 C027             ld      (fcb2),a
C094    E1                  pop     hl
C095    E5                  push    hl
C096    7E                  ld      a,(hl)      ; Test for all done
C097    B7                  or      a
C098    28 17               jr      z,copydn
C09A    11 C005             ld      de,fcb1+1   ; Copy the name to fcb1
C09D    01 000B             ld      bc,11
C0A0    ED B0               ldir
C0A2    E1                  pop     hl          ; Copy the name to fcb2
C0A3    11 C028             ld      de,fcb2+1
C0A6    01 000B             ld      bc,11
C0A9    ED B0               ldir
C0AB    E5                  push    hl          ; Save the list pointer
C0AC    CD C0CB             call    copy
C0AF    18 DA               jr      copylp

                    ; All done, so go home
C0B1    E1          copydn: pop     hl          ; Clear the stack ..
C0B2    C9                  ret                 ; .. and go home

                    ; Subroutines
                    ; Locate file in directory list
C0B3    87          loc:    add     a,a
C0B4    87                  add     a,a
C0B5    87                  add     a,a
C0B6    87                  add     a,a
C0B7    87                  add     a,a
C0B8    26 00               ld      h,0
C0BA    6F                  ld      l,a
C0BB    11 0081             ld      de,dma+1
C0BE    19                  add     hl,de
C0BF    C9                  ret

                    ; Clear out fcb1 and fcb2
C0C0    AF          clr:    xor     a
C0C1    06 3A               ld      b,start-fcb1b
C0C3    21 C010             ld      hl,fcb1b
C0C6    77          clr1:   ld      (hl),a
C0C7    23                  inc     hl
C0C8    10 FC               djnz    clr1
C0CA    C9                  ret
```

```
Index file loader for Dbase     M-80    17 May 1985     21:53   PAGE    1-2

                        ; Copy the named file
C0CB    11 C027         copy:   ld      de,fcb2     ; See if destination ..
C0CE    0E 0F                   ld      c,opnfil    ; .. file already exists ..
C0D0    CD 0005                 call    bdos
C0D3    FE FF                   cp      0ffh
C0D5    28 08                   jr      z,copy1
C0D7    11 C027                 ld      de,fcb2     ; .. if so, delete it
C0DA    0E 13                   ld      c,delfil
C0DC    CD 0005                 call    bdos
C0DF    11 C027         copy1:  ld      de,fcb2     ; Now make the destination file
C0E2    0E 16                   ld      c,makfil
C0E4    CD 0005                 call    bdos
C0E7    11 C004                 ld      de,fcb1     ; Open the source file
C0EA    0E 0F                   ld      c,opnfil
C0EC    CD 0005                 call    bdos

                        ; The copy loop
C0EF    11 C004         copy2:  ld      de,fcb1     ; Read block into dma
C0F2    0E 14                   ld      c,rdfil
C0F4    CD 0005                 call    bdos
C0F7    B7                      or      a           ; Test for end
C0F8    20 0A                   jr      nz,copy3
C0FA    11 C027                 ld      de,fcb2     ; Write out the buffer
C0FD    0E 15                   ld      c,wrfil
C0FF    CD 0005                 call    bdos
C102    18 EB                   jr      copy2
C104    11 C027         copy3:  ld      de,fcb2     ; File end, so close it
C107    0E 10                   ld      c,clsfil
C109    CD 0005                 call    bdos
C10C    C9                      ret

                        ; Workspace for file names list
C10D                    save:   end

Index file loader for Dbase     M-80    17 May 1985     21:53   PAGE    S

Macros:

Symbols:
0005    BDOS            C0C0    CLR             C0C6    CLR1
0010    CLSFIL          C0CB    COPY            C0DF    COPY1
C0EF    COPY2           C104    COPY3           C0B1    COPYDN
C08B    COPYLP          0013    DELFIL          0080    DMA
C003    DSTDRV          C004    FCB1            C005    FCB1A
C010    FCB1B           C027    FCB2            C0B3    LOC
0016    MAKFIL          000F    OPNFIL          0014    RDFIL
C10D    SAVE            C084    SCHDON          001A    SETDMA
C06C    SRCH            0011    SRCHFST         0012    SRCHNXT
C04A    START           0015    WRFIL

No Fatal error(s)

★ Routine to move index files

SET CALL TO 49152
@ 18,0 SAY "Copying indexes to the M: drive, please wait."
```

```
* Poke in the machine code
POKE 49152,195,74,192
POKE 49226,17,128,0,14,26,205
POKE 49232,5,0,205,192,192,17,4,192,14,17,205,5,0,254,255,200
POKE 49248,205,179,192,17,13,193,1,11,0,237,176,213,17,4,192,14
POKE 49264,18,205,5,0,254,255,40,12,205,179,192,209,1,11,0,237
POKE 49280,176,213,24,232,209,175,18,33,13,193,229,205,192,192,58,3
POKE 49296,192,50,39,192,225,229,126,183,40,23,17,5,192,1,11,0
POKE 49312,237,176,225,17,40,192,1,11,0,237,176,229,205,203,192,24
POKE 49328,218,225,201,135,135,135,135,135,38,0,111,17,129,0,25,201
POKE 49344,175,6,58,33,16,192,119,35,16,252,201,17,39,192,14,15
POKE 49360,205,5,0,254,255,40,8,17,39,192,14,19,205,5,0,17
POKE 49376,39,192,14,22,205,5,0,17,4,192,14,15,205,5,0,17
POKE 49392,4,192,14,20,205,5,0,183,32,10,17,39,192,14,21,205
POKE 49408,5,0,24,235,17,39,192,14,16,205,5,0,201

* Set up the destination, source and file mask string
* (In this case, to M: drive, from B: drive, all files like ST-*.NDX.)
STORE CHR(RANK("M")-64) + CHR(RANK("B")-64) + "ST-?????NDX" TO files

* Poke the files string into the fcb
STORE 1 TO c
DO WHILE c<=12
  POKE 49154+c,RANK($(files,c,1))
  STORE c+1 TO c
ENDDO

* Now do it.
CALL
```

Private Advertisements

Items For Sale

Nascom 1 in Vero frame, 10A PSU, V&T Superdeck, 64k RAM B, fully populated I/O board, Sound board, EPROM programmer/eraser, 2 EPROM boards containing extended Nas-Sys 3, Crystal BASIC 2.2, *Hi-Soft* PASCAL 3, Editor/Assembler and cassette operating system. Documentation and some source listings. Only £320; with GP80A Printer, £400 ono. Also a 3A PSU for £20. RAM A board (no RAM) for £15. High speed tape interface £5. Please contact Mr M Parker, Botley, Oxford, 0865 725495 evenings/weekends.

Nascom 2 Computer, *Gemini* GM802 Memory Board (16K), GM807 3A PSU, all encased in a small bureau complete with 20" B&W TV and cassette recorder. £125.00. Tel: E W Hair, 01 578 5423.

2-off 16K *Nascom* 2 computers about £200 each. One has EPROM burner and Zeap. 1-off 48K *Nascom* 3 computer about £400. Also has Zeap and word-processing program. Also for sale are 3-off B/W Monitors and 3-off Decwrite type Printers. Any reasonable offer accepted. Some items need attention. Potential owners must see goods first and collect. Pay cash. Contact Mr Parsons, Worth School, Turners Hill, CRAWLEY, Sussex, or ring Copthorne 714821 after 10.00 pm.

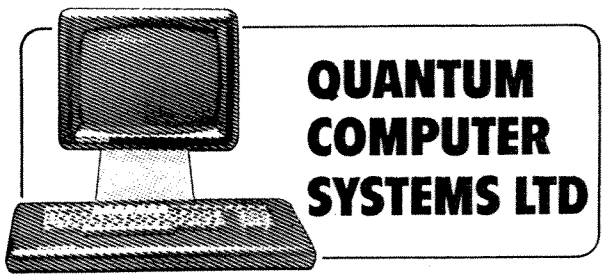
Nascom 3 Microcomputer; 48K memory — NAS-PEN included. Offers around £250. Telephone Joseph Townsend, 023 063 8886.

GM803 EPROM card with manuals for £30. Telephone: Tom Gibson, Middlesbrough 0642 452775 (evenings or weekends).

Two *Gemini* GM803 EPROM boards, one fully populated with 16 x 2716 EPROMs containing MBASIC and GBASIC, and with 1 x 2764 (in MK36000 socket) containing GemZap. The other board is partly populated with 6 x 2716 EPROMs containing GemDebug and GemPen. Prices £100 and £90 (ONO) respectively. Contact Martin Davies, 0684 72178.

EPROM Programmer, (*I.O. Systems* type), 2708 to 2732. Self-contained power supplies. Software on tape or PolyDos format disk. I'm upgrading to *Gemini* GM860. £35 ... Tel: John 0357 21672.

Compas 3 for *Gemini* QDSS at £175. Rights to be transferred as per licensing agreement. Tel: Mr P Young, Belfast, 0232 790508



Springfield Road Chesham Bucks HP5 1PU
Telephone (0494) 771987 Telex 837788

TURBO PASCAL V3

* Absolute address variables
* Bit/byte manipulation
* Direct access to CPU memory & data ports
* Dynamic strings
* Free ordering of sections within declaration part
* Full support of CP/M facilites
* In-line machine code generation
* Include files
* Logical Operations on integers
* Overlay system
* Program chaining with common variables
* Random access data files
* Structured constants
* Type conversion functions
* Wordstar type editor
* Twice as fast as TURBO V2
* One step compile
* Installed for Gemini IVC/SVC

WHAT THE CRITICS SAY

PC MAGAZINE "Language deal of the century...TURBO PASCAL. It introduces a new programming enviroment and runs like magic."

POPULAR COMPUTING "Most Pascal compilers barely fit on a disk, but TURBO PASCAL packs an Editor, Compiler, Linker, and Run time library into just 29k Bytes of RAM."

BYTE "What I think the computer industry is headed for: Well documented, standard, plenty of good features, & a reasonable price."

TURBO PASCAL IS AVAILIABLE NOW
GIVE YOUR SYSTEM A TURBO BOOST

TURBO PASCAL V.3	CP/M 80	69.95
" " "	CP/M 86	102.95
TURBO 8087 version	CP/M 86	109.95

TURBO TOOLBOX

Designed to compliment the power and speed of Turbo Pascal, TURBO TOOLBOX consists of three modules created to save you from "rewriting the wheel" syndrome.

TURBOISAM Files using B+ Trees
Makes it possible to access records in a file using keys (e.g. "Smith" or "Rear Bumper") instead of just a number. Even though you have direct access to individual records in the file you also have easy access to the records in a sorted sequence. A must if you plan to work with direct access files. Source code is included in the manual.

QUICKSORT ON DISK
The fastest way to sort your arrays. Preferred by knowledgeable programmers. Available for you now with commented source code.

GINST (General Installation Program)
Now...the programs you write with Turbo Pascal can have a terminal installation mode just like Turbo's!. Saves hours of work and research, and adds tremendous value to everything you write.

TURBO TOOLBOX 54.95

TURBO TUTOR

Don't know Pascal?..... Let Turbo Tutor teach you. Supplied as a disk of demonstration programs and a very informative and entertaining manual this package will show you how to use your Turbo Pascal to best advantage.

TURBO TUTOR 34.95

All disks supplied in Gemini QD-96tpi format as standard, please state your format if different. All Prices are exclusive of Vat (15%) and Carriage & Packing 1.50. Personal Callers by appointment Only.

CONDITIONAL SUBMIT FILES

By Steve Willmott

The CP/M SUBMIT facility is much like an elementary Job Control Language. The public domain EXSUB facility is very similar. They both support a job submission with parameters which enable repetitive jobs like the development cycle edit, compile, link and run to be conveniently defined and then used generally. When submitted only the command file and the parameters which define this job (e.g. the source file to be compiled) need be supplied. A typical example file called MAC.SUB is:

```
ED $1.MAC
M80 =$1
L80 /P:100,$1,LIBRARY/S,$1/N/E
$1
```

and invoked in response to the CCP prompt to develop the program DEMO.MAC by

```
SUBMIT MAC DEMO
```

The SUBMIT utility processes the MAC.SUB file substituting the parameter string 'DEMO' in place of the place holder $1. It creates a new file called $$$.SUB with one command line in each 128 byte record in the reverse order. Whenever the CCP runs it looks for this file on disk and reads the last record into the command area below address 0100 hex, removes the last record from the file $$$.SUB and obeys the installed command as if it had been entered manually. If the CCP finds the file $$$.SUB empty then it deletes the file from disk thereby ending the job.

However, there is at least one annoying deficiency in CP/M in this area and that is that the only way to stop a submit file is to hit the keyboard while the CCP is reading the next command from disk. CP/M provides no means of any of the transient programs run in the TPA to return a success or failure indication such that the CCP could pick up this status and automatically abort the submit file. If the LIBRARY.REL file used above is large it can take minutes to perform the linkage to produce the DEMO.COM object file. When the compile fails and the linker is entered one is tempted to hit the keyboard very hard, particularly if it is a small syntax error and a long wait for the linker to complete. It seems unsatisfying to just hit the restart button.

Most decently designed programs output a report on how well it has run. Perhaps unsurprisingly the report is left on the screen for the user to see. Each program normally generates this report in a fairly fixed format. A compiler will state how many errors or warnings have or have not been found. With a Gemini IVC or SVC video board (or indeed a NASCOM with memory mapped display or I assume an AVC or whatever) one is able to read back from the display and analyse this report. The result of the analysis could cause the abortion of the submit file. This is what the CHECK utility does when it is run as part of a submit file. The program has been written for the IVC or SVC, but since the M80 source for CHECK is listed here it should not be too difficult to modify for other systems. A typical use would be in the submit file MAC.SUB:

```
ED $1.MAC
M80 =$
CHECK 'NO FATAL ERROR(S)'/CU
L80 /P:100,$1,LIBRARY/S,$1/N/E
$1
```

The command syntax for CHECK is:

```
CHECK 'string'/switches
```

The supplied string is compared to a previous line on the display according to the optional switches. If the check is successful CHECK merely exits back to the CCP and the submit file continues. If the check fails then CHECK prompts for abortion of the submission. If the answer is Yes the executing submit file '$$$.SUB' is deleted from the disk, thereby aborting the submit file. If the answer is No CHECK returns to the CCP and everything carries on as normal.

One can experiment with CHECK by using the screen edit facilities provided by the BIOS. However, the following provides some of the rationale behind the program design. The supplied string must be in single quotes, because since the /switches are optional one cannot supply a string of spaces or a null string — the CCP will not see them. The '/' separator before the switches merely conforms to common syntax. The switches all have defaults which allow them to be optional. They can be supplied in any order. Some of the switches have been built in to increase the pattern matching ability of the utility. Considering each switch in turn:

B indicates that previous blank lines are significant and modifies the effect of the L switch. The default is to ignore blank lines.

C indicates that the supplied string must match the complete displayed line. The example above means that warnings from the M80 assembler will be trapped as the warning report follows the fatal error report on the same line. The default is to allow a match anywhere in the line.

F indicates that the string must match the finish of the displayed line. The default is a match

anywhere in the line. This switch will be overridden by a C switch setting.

Ldd indicates that the displayed line to be checked is the dd'th previous to the CHECK command line. An error is reported if this line is up off the screen. The B switch affects the line counting. The default value is one ie. the previous line on display.

N indicates that the presence of the supplied string prompts for abortion of the submission. The default condition is that a matching string according to the other switches indicates that the submit file should continue.

S indicates that the string must match the start of the displayed line. The default is a match anywhere in the line. This switch will be overridden by an F switch setting.

U indicates that the displayed line should be converted to upper case before comparison with the supplied string. This is normally necessary as the CCP capitalises the command when it is put into the command area. No capitalisation is an option under CCPZ — the public domain rewrite of CCP for the Z80. The default option is no case conversion.

W indicates that a '?' in the supplied string is wild ie. as with the CP/M ambiguous file name convention '?' will match any character in the displayed line. The default is that '?' treated as a normal character.

I hope that readers of the 80-BUS News find this short article interesting and the supplied program useful.

```
   TITLE   CHECK SUBMIT
.Z80
.COMMENT !

CHECK 'string'/switches

This utility is run as part of a submit file. It a
checks a previous line on display against the supplied
string. If this string matches the displayed line
according to the switches the submit continues,
otherwise it can be aborted after a prompt.
The optional switches are:

B   causes blank lines to be counted with the L
    switch, defaults to ignore blank lines,
C   the string matches the full displayed line,
    defaults to only part,
F   the string matches finish of displayed line,
    defaults to anywhere,
Ldd causes the dd'th previous line to be compared
    with the string, defaults to one,
N   the string must not be in the displayed line,
    defaults to contained,
S   the string matches start of displayed line,
    defaults to anywhere,
U   the displayed line is converted to uppercase
    before comparison, defaults to no case change,
W   indicates that '?' in the supplied string are
    wild cards, defaults to ordinary '?' for matching.

Switch C overrides F which overrides S.

It is intended to trap compiler error summaries etc
and stop the continuation of the submit file. An
example submit file is:

    ED $1.MAC
    M80 =$1
    CHECK 'NO FATAL ERROR(S)'/CU
    L80 /P:100,$1,$1/N/E
    $1

    S C Willmott   26 April 1985
```

Private Advertisements

Items For Sale

Nascom 2 with CP/M and Nas-Dos, MAP 80 RAM card (full 256K), *Nascom* AVC with CP/M and Nas-Dos software, *Nascom* FDC, one SSDD disk drive in twin drive box and PSU, *Gemini* 5A PSU and 8 slot backplane in rack frame. *Gemini* EPROM board with all the usual firmware, Bits and PC's EPROM programmer — lots of spare 2708's and 2716's, PAL encoder card (needs attention). *Nascom* 1 with 32K RAM A card and BASIC ROM, *Cottis-Blandford* interface, 3A PSU (needs attention). Lots of disk and tape software and all the manuals. Offers for whole or parts to: Dr David Plews, Tel. Steeton 0535 52511 (day), 0535 54157 (evening).

```
DEBUG   EQU   0

VCDATA  EQU   0B1H   ;VIDEO CARD DATA PORT
VCSTAT  EQU   0B2H   ;VIDEO CARD STATUS PORT

NSWTCH  EQU   0      ;MUST BE LS BIT
BSWTCH  EQU   1
CSWTCH  EQU   2
FSWTCH  EQU   3
SSWTCH  EQU   4
USWTCH  EQU   5
WSWTCH  EQU   6
```

```
WBOOT   EQU   0
FDOS    EQU   5
FCB     EQU   5CH
CBUFF   EQU   80H

RCONF   EQU   1
WCONBF  EQU   9
DELF    EQU   19

BEL     EQU   07H
LF      EQU   0AH
CR      EQU   0DH
ES      EQU   1BH
WILD    EQU   '?'

   LD   (STK),SP      ;SAVE CCP STACK POINTER
   LD   SP,STK        ;SET UP STACK
   XOR  A             ;INITIALISE SWITCHES
   LD   IX,SWTCH
   LD   (IX),A
   INC  A             ;SET DEFAULT L SWITCH LINE NO
   LD   (LIN),A
   CALL GETCUR        ;GET CURRENT POSITION OF CURSOR
   LD   (CURPOS),HL   ;AND SAVE IT
   LD   DE,CMDFAL     ;CHECK THAT A STRING HAS BEEN
   LD   HL,CBUFF      ;  SUPPLIED
   LD   B,(HL)        ;NO. CHARS IN COMMAND LINE
   LD   A,B
   CP   2         ;AT LEAST THE SEPARATING SPACE AND '
   JP   C,REPORT      ;IF TOO SHORT A STRING SUPPLIED
   INC  HL
   INC  HL
   LD   A,(HL)
   CP   ''''
   JP   NZ,REPORT     ;IF NO LEADING '
   LD   L,B         ;PICK UP LAST CHAR IN COMMAND LINE
   LD   H,0
   LD   BC,CBUFF
   ADD  HL,BC
CHKSTR: LD   A,(HL)   ;SEARCH BACK FOR TRAILING '
   DEC  HL
   CP   ''''
   JR   NZ,CHKSTR     ;IF NOT '
   INC  HL
   LD   A,L           ;CALCULATE LENGTH OF STRING
   SUB  CBUFF+3
   JP   C,REPORT      ;IF NO TRAILING '
   LD   (STRLEN),A    ;AND SAVE FOR MATCH COUNT
   LD   A,(CBUFF)     ;CHECK SWITCHES
   ADD  A,CBUFF
   SUB  L
   JR   Z,RUN         ;IF NO SWITCHES SUPPLIED
   CP   2
   JP   C,REPORT      ;IF COMMAND TOO SHORT
   LD   B,A           ;SAVE NO. SWITCHES + 1
   INC  HL            ;PICK UP SWITCH SEPARATOR
   LD   A,(HL)
   CP   '/'
   JP   NZ,REPORT     ;IF NOT /
   DEC  B             ;NO. OF SWITCHES

CHKSW:   INC   HL     ;CHECK EACH SWITCH
   LD   A,(HL)
   CP   'B'
   JR   NZ,CHKC
   SET  BSWTCH,(IX)   ;SET COUNT BLANK LINES
   JR   NXTSW
CHKC:   CP   'C'
   JR   NZ,CHKF
   SET  CSWTCH,(IX)   ;SET COMPLETE LINE
   JR   NXTSW
CHKF:   CP   'F'
   JR   NZ,CHKL
   SET  FSWTCH,(IX)   ;SET FINISH OF LINE
   JR   NXTSW
CHKL:   CP   'L'
   JR   NZ,CHKN
   LD   A,B      ;CHECK NO CHARS LEFT IN COMMAND LINE
   CP   3
   JP   C,REPORT      ;IF NOT ENOUGH FOR Ldd
   CALL DIGIT         ;PICK UP MS DIGIT
   LD   C,A           ;C := A * 10
   ADD  A,A
   ADD  A,A
   ADD  A,C
   ADD  A,A
   LD   C,A
   CALL DIGIT         ;PICK UP LS DIGIT
   ADD  A,C           ;ADD IN MS DIGIT
   LD   (LIN),A       ;STORE LINE COUNT
   JR   NXTSW
CHKN:   CP   'N'
   JR   NZ,CHKS
   SET  NSWTCH,(IX)   ;SET NOT CONTAINED IN
   JR   NXTSW
CHKS:   CP   'S'
   JR   NZ,CHKU
   SET  SSWTCH,(IX)   ;SET START OF LINE
   JR   NXTSW
CHKU:   CP   'U'
   JR   NZ,CHKW
   SET  USWTCH,(IX)   ;SET CONVERT TO UPPER CASE
   JR   NXTSW
CHKW:   CP   'W'
   JP   NZ,REPORT
   SET  WSWTCH,(IX)   ;SET ? AS WILD CARD
NXTSW: DJNZ  CHKSW    ;IF MORE SWITCHES
RUN: LD  HL,(CURPOS)  ;CURSOR POSITION
   DEC L              ;START FROM COMMAND LINE
BACK:   DEC   L       ;BACK A LINE
   JP   P,CHEK        ;IF NOT OFF TOP OF SCREEN
   LD   HL,(CURPOS)   ;RESTORE CURSOR TO ORIGINAL POSN
   CALL PUTCUR
   LD   DE,TOPFAL
   JP   REPORT
CHEK: CALL  PUTCUR    ;ELSE POSITION CURSOR
   CALL GETLIN        ;GET LINE FROM SVC
   BIT  BSWTCH,(IX)   ;CHECK COUNT BLANK LINE SWITCH
   JR   NZ,CLIN       ;IF YES COUNT LINE
   LD   A,C           ;CHECK LENGTH OF LINE
   OR   A
   JR   Z,BACK        ;IF EMPTY LINE
```

```
CLIN: LD  A,(LIN)      ;COUNT LINE BACK
   DEC  A
   LD   (LIN),A
   JR   NZ,BACK        ;IF REACHED LINE
   PUSH BC             ;SAVE LINE LENGTH
   LD   HL,(CURPOS)    ;RESTORE CURSOR TO ORIG POSN
   CALL PUTCUR
   POP  BC             ;RESTORE LINE LENGTH
   LD   DE,LINE        ;DE = @ DISPLAYED STRING
   LD   HL,CBUFF+3  ;IGNORE INITIAL SPACE & LEADING '
   LD   A,(STRLEN)
   LD   B,A            ;B = NO. CHARS IN STRING
   OR   C
   JP   Z,MATCH        ;IF NULL STRING AND BLANK LINE
   BIT  CSWTCH,(IX)    ;CHECK COMPLETE LINE SWITCH
   JR   Z,CHKFIN       ;IF NOT SET, ANY MATCH
   LD   A,B            ;COMPARE LENGTHS
   CP   C
   JR   NZ,NOMTCH      ;IF DIFFERENT LENGTHS
   JR   PRECMP
CHKFIN: BIT  FSWTCH,(IX) ;CHECK FINISH OF LINE SWITCH
   JR   Z,PRECMP       ;IF NOT SET
   LD   A,C
   CP   B
   JR   C,NOMTCH       ;IF DISPLAYED LINE TOO SHORT
   EX   DE,HL       ;HL := @ LINE + (C) - (B), SAVE HL
   PUSH BC             ;SAVE CHAR COUNTS
   LD   B,0
   ADD  HL,BC
   POP  BC             ;RESTORE CHAR COUNTS
   LD   C,B
   LD   B,0
   OR   A
   SBC  HL,BC
   LD   B,C    ;MUST COMPARE SAME NO. CHARS AS STRING
   EX   DE,HL     ;DE = @ OF END OF STRING, RESTORE HL
PRECMP: INC  C   ;C=COUNT CHARS IN DISPLAYED STRING+1
BEGCMP: PUSH BC        ;SAVE START OF COMPARISON
   PUSH  DE
   PUSH  HL
COMP: DEC  C  ;COUNT DISPLAYED CHARS TO BE COMPARED+1
   JR   Z,NOMTCH       ;IF DISPLAYED LINE TOO SHORT
   BIT  WSWTCH,(IX)    ;CHECK FOR WILD CARDS
   JR   Z,CHKUP        ;IF NOT SET
   LD   A,(HL)         ;CHECK FOR WILD CARD IN STRING
   CP   WILD
   JR   Z,SKPCMP       ;IF WILD CARD SKIP COMPARISON
CHKUP: BIT  USWTCH,(IX)   ;CHECK CASE SHIFT SWITCH
   JR   NZ,USHIFT      ;IF CONVERT TO UPPER CASE
   LD   A,(DE)         ;PICK UP DISPLAYED CHAR
   JR   DOCOMP
USHIFT: LD  A,(DE)     ;PICK UP DISPLAYED CHAR
   CP   'a'            ;ENSURE UPPER CASE
   JR   C,DOCOMP
   CP   'z'+1
   JR   NC,DOCOMP
   SUB  20H            ;CONVERT TO UPPER CASE
DOCOMP: IF  DEBUG      ;IF IN DEBUG MODE
   PUSH AF             ;SAVE LINE CHAR
   CALL PVID           ;OUTPUT LINE CHAR
   LD   A,(HL)         ;AND STRING CHAR TO BE COMPARED
   CALL PVID
   POP  AF             ;RESTORE LINE CHAR
   ENDIF
   CP   (HL)           ;COMPARE CHARS
   JR   NZ,NXTCMP      ;IF NO MATCH
SKPCMP: INC  DE        ;BUMP DISPLAYED STRING @
   INC  HL             ;BUMP CHECK STRING @
   DJNZ COMP           ;IF MORE CHARS IN CHECK STRING
   JR   MATCH          ;STRING MATCHES
NXTCMP: BIT  SSWTCH,(IX)  ;CHECK START OF LINE SWITCH
   JR   NZ,NOMTCH      ;IF YES THEN NO MATCH
   POP  HL             ;RESTORE START OF COMPARISON
   POP  DE
   POP  BC
   INC  DE ;STEP TO NEXT CHAR ON DISPLAY START COMPARE
   DEC  C         ;COUNT DISPLAYED CHARS FULLY CHECKED
   JR   BEGCMP         ;FOR NEXT ATTEMPT TO MATCH
MATCH: XOR  A          ;A = 0 FOR MATCH
   JR   CHKNOT
NOMTCH:  LD  A,1       ;A = 1 FOR NO MATCH
CHKNOT:  LD  HL,SWTCH ;A := (A) XOR (NSWTCH)
   XOR  (HL)
   AND  1
   JR   NZ,FAIL        ;IF SUBMIT DEEMED TO HAVE FAILED
   LD   DE,OKMSG       ;ELSE CONTINUE WITH SUBMIT
   JR   REPORT
FAIL: LD  DE,PROMSG    ;PROMPT FOR ABORT
   CALL MSG
   LD   C,RCONF        ;READ CON:
   CALL FDOS
   AND  0DFH           ;REMOVE CASE BIT
   CP   'Y'            ;IF YES
   JR   Z,ABORT
   CP   'N'
   JR   NZ,FAIL        ;IF NEITHER Y OR N ENTERED
   LD   DE,CONMSG      ;STILL CARRY ON
   JR   REPORT
ABORT: LD  DE,SUBFCB   ;DELETE SUBMIT FILE
   LD   C,DELF
   CALL FDOS
   LD   DE,DELFAL
   INC  A
   JR   Z,REPORT       ;IF SUBMIT FILE NOT FOUND
   LD   DE,ABOMSG      ;ABORTED SUBMIT
REPORT: CALL   MSG     ;REPORT OUTCOME
   IF   DEBUG
   JP   WBOOT          ;WARM BOOT
   ELSE
   LD   SP,(STK)       ;RETURN TO CCP
   RET
   ENDIF

DIGIT:   INC   HL      ;PICK UP DIGIT FROM COMMAND LINE
   LD   A,(HL)
   DEC  B              ;COUNT CHAR
   CP   '9'+1
   JR   NC,REPORT      ;IF NOT DECIMAL DIGIT
   SUB  '0'
   JR   C,REPORT       ;IF NOT DECIMAL DIGIT
   RET                 ;A = BINARY
```

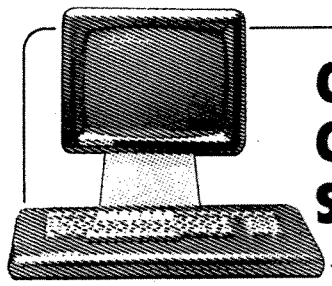
```
GETLIN: LD  A,'Z'      ;GET LINE FROM VIDEO CARD
   CALL PVESC          ;GET LINE COMMAND
   LD   C,0            ;C=COUNT OF CHARS IN LINE
   LD   DE,LINE        ;DE = BUFFER @
GTLIN: CALL  GVID      ;GET CHAR
   CP   CR
   RET  Z              ;IF CR THEN END OF LINE
   LD   (DE),A         ;ELSE STORE IN BUFFER
   INC  DE             ;BUMP BUFFER @
   INC  C              ;COUNT CHAR
   JR   GTLIN          ;FOR MORE

GETCUR: LD  A,'?'      ;HL := CURRENT CURSOR POSITION
   CALL PVESC          ;GET CURSOR COMMAND
   CALL GVID           ;GET ROW
   LD   L,A
   CALL GVID           ;GET COLUMN
   LD   H,A
   CALL GVID           ;GET CHAR
   RET

PUTCUR: PUSH  HL       ;CURSOR POSITION := HL
   LD   BC,2020H       ;ADD OFFSET
   ADD  HL,BC
   LD   A,'='          ;POSITION CURSOR COMMAND
   CALL PVESC
   LD   A,L            ;PUT ROW
   CALL PVID
   LD   A,H            ;PUT COLUMN
   CALL PVID
   POP  HL
   RET

PVESC:  PUSH  AF       ;OUTPUT ESCAPE (A)
   LD   A,ES
   CALL PVID
   POP  AF

PVID: PUSH AF          ;OUTPUT (A) TO VIDEO CARD
PVO:  IN   A,(VCSTAT)
   RRCA
   JR  C,PVO
   POP AF
   OUT (VCDATA),A
   RET

GVID: IN A,(VCSTAT)    ;INPUT A FROM VIDEO CARD
   RLCA
   JR  C,GVID
   IN  A,(VCDATA)
   RET

MSG: LD  C,9           ;OUTPUT MESSAGE (DE)
   JP  FDOS

SUBFCB: DEFB 0,'$$$     SUB',0,0,0,0
CMDFAL: DEFB BEL,'??? Bad command: CHECK ''string''/switches',CR,LF
DEFB 'where optional /switches maybe B, C, F, Ldd, N, S, U and/or W'
DEFB CR,LF,'$'
TOPFAL: DEFB BEL,'??? No previous non-blank line',CR,LF,'$'
DELFAL: DEFB BEL,CR,LF,'??? No SUBMIT file found',CR,LF,'$'
PROMSG: DEFB BEL,CR,LF,'CHECK fail - abort SUBMIT (Y/N)? $'
ABOMSG: DEFB CR,LF,'SUBMIT aborted',CR,LF,'$'
CONMSG: DEFB CR,LF,'CHECK overridden - continuing',CR,LF,'$'
OKMSG:  DEFB 'CHECK okay - continuing',CR,LF,'$'

SWTCH:  DEFS   1
LIN:    DEFS   1
STRLEN: DEFS   1
CURPOS: DEFS   2
LINE:   DEFS   80
DEFS    32
STK:    DEFS   2

END
```

NEW GEMINI BIOS

Gemini have now released their new BIOS V3.2 incorprating many advanced features not normally found in CP/M 80 systems. Features include the ability to add different types of disk drives including 3.5" and 8"; read and write in twelve different disk formats; CCPZ allowing 16 separate user areas per drive with hierarchical search; Max of 5 logical drives can be set up for floppy based systems or 8 for Winchester based systems; support for the GM833 ram disk or Gemini page mode ram disk. For those lucky enough to own Gemini winchester systems user definable directory sizes are offered together with support for two different system tracks.

Bios V3.2 is available as an upgrade for those with an existing Gemini CP/M. We regret that we are unable to supply this BIOS for use with Nascom based systems.

To obtain your copy of Bios 3.2 send your original Gemini Master CP/M disk together with a cheque for 35.65 inc Vat & postage to: 'BIOS Upgrade', QUANTUM COMPUTER SYSTEMS LTD. Springfield Road,Chesham,Bucks, HP5 1PU.



QUANTUM COMPUTER SYSTEMS LTD

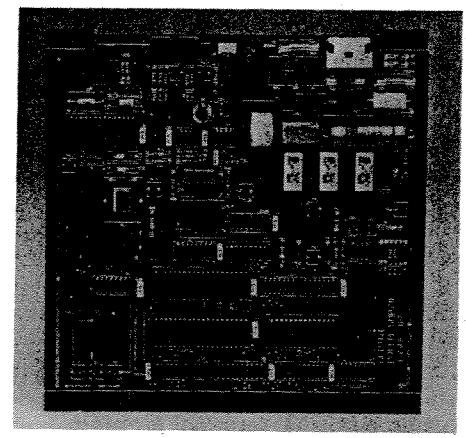
Springfield Road Chesham Bucks HP5 1PU
Telephone (0494) 771987 Telex 837788

NOW AVAILABLE 3 NEW 80-BUS BOARDS

GM870 MODEM BOARD

The Gemini GM870 Modem board is an 80-BUS board thus alleviating the need for extra serial ports. The board is compatible with any of the Gemini CPU boards and is designed around the AMD 7910. The modem provides low speed data communication based on CCITT standards.(300/300 & 1200/75 baud) The board has both auto-dial & auto answer capability. Software is provided with the modem on disk.

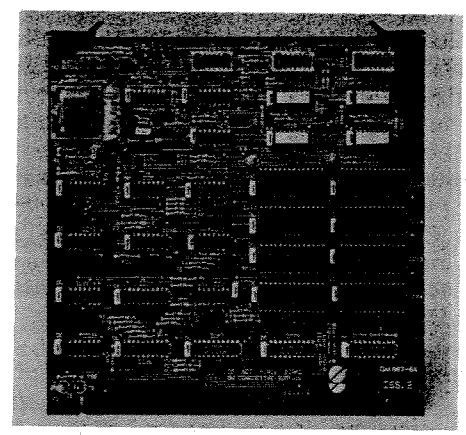
GM870 175.00



GM863 STATIC RAM BOARD

The Gemini GM863 80-BUS Static RAM boards are available in either 32k or 64k versions. An on-board rechargeable battery provides memory content retention during power-down periods. The GM863 supports the Extended Addressing mode when used with the GM813 CPU board and may also be set to any one of four pages of the Gemini page mode system.

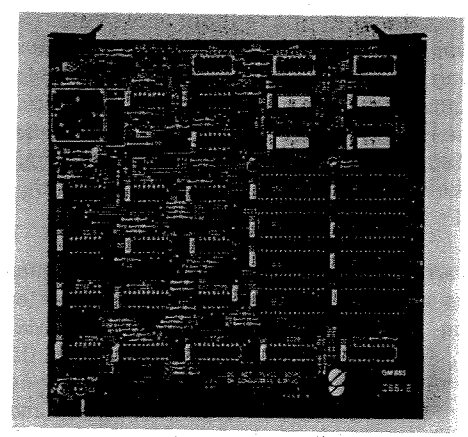
GM863-32 150.00
GM863-64 215.00



GM853 EPROM BOARD

The Gemini GM853 'Bytewide' Eprom board contains 8 sockets to accept many different types of memory devices, ranging from 64kbit (8k x 8) up to 512kbit (64k x 8). This includes the standard 2764, 27128, 27256 & 27512 devices. The GM853 supports the Extended Addressing mode when used with the GM813 CPU board and may also be set to any of the four pages of the Gemini page mode system.

GM853 95.00



All Prices are exclusive of VAT currently 15% Carriage & Packing 2.50
Personal Callers by appointment Only.

PRESTEL UPDATE

by Robin Luxford

A review of the *Henelec Prestel* Terminal program and some installation notes on the *GEC* LTU-II kit.

Although the 300 BAUD *Prestel* service has been accessible to CP/M 80-BUS computers for some time (see 80-BUS NEWS Vol 2. iss 6.), the appearance of the new Henelec PRETZEL 2 software from *Henry's* has made the 75/1200 BAUD service readily available. Since the 300 BAUD ports on the *Prestel* computers strip all the graphics characters and replace them with asterisks, the graphics are pretty dull. The full set of characters and mosaics loaded into the *Gemini* SVC/IVC video boards by PRETZEL 2 makes the display of pages far more interesting and represent a fair compromise as the SVC/IVC can not reproduce colour. PRETZEL 2 also reprograms the SVC/IVC to the *Prestel* screen format of 40 columns by 24 lines, making text and graphics appear in the correct aspect ratio, a considerable improvement over the normally cramped output of the 300 BAUD service displayed using a normal terminal program.

PRETZEL 2 has facilities for saving and reloading pages to and from disk (either on or off line), and also for dumping the current screen image to printer. The program contains a screen editor allowing the user to alter and experiment with pages without being connected to the *Prestel* computer. On line the system works in the approved *Prestel* full duplex mode where all keyboard input except the program's command codes is echoed to the screen via the *Prestel* computer. Off line all keyboard entry is direct.

My only criticisms of the software are the lack of a directory facility, coupled with the fact that the program stores the screen images individually rather than in the form of a random access file. The latter is wasteful of disk space as each image is only 1K long but with a *Gemini* CP/M block size of 4K, three quarters of the space allocated is wasted. The former is simply inconvenient as the number of saved pages soon mounts up, and it is difficult to remember what names have been used or may be deleted.

The program is written entirely in Z80 machine code and is supplied on disk as a fully commented Z80 source file suitable for the *Microsoft* M80 assembler. All documentation is supplied on the disk along with some twenty demonstration pages. The user has to patch the I/O section to accommodate the modem used and select the various options for *Nascom/Gemini* processor cards and either SVC or IVC and then assemble the program for use. One assembly option is to include the users' identity number and password which are then sent automatically on log-on. Minor text changes would make the source file suitable for most other Z80 assemblers. Another advantage of having the source file is that it is possible to patch in any auto-dial facilities which the user may require such as those described on page 22 of 80-BUS NEWS vol 2. iss 6.

Overall I have found PRETZEL 2 an excellent piece of software particularly at the £30.00 charged for it. The well chosen graphics more than make up for lack of colour on the screen and its storage and printer facilities offer much more comprehensive use of *Prestel* than an adaptor connected to a TV set.

My modem is the 75/1200 BAUD modem available as a surplus unit from the same source as the program. The *GEC* LTU-11 modem was designed for a *GEC Prestel* terminal, but slight hardware modifications make it entirely suitable for use with either *Nascom* or *Gemini* processors. The unit is supplied in two parts, as a printed card about 6 x 4 inches, without circuit diagrams but with full descriptive manual, and a sealed box containing the auto-dial relays and isolation transformer, measuring some 3 x 5 x 2 inches (oddly enough the isolation box is supplied with circuit diagrams but no description). The modest power supplies are derived from the computer. The modem card expects TTL level inputs and outputs so a simple interface is required to make it suitable for the RS232 from either a *Gemini* or *Nascom*. Additionally the data to and from the card is inverted so use is made of a simple 74LS04 to buffer the data, and provide the inversions.

Data out from the RS232 is clipped using two resistors and zener diode to convert the + and - 12V RS232 swing to a swing between 0 and 5 volts, this is then inverted by one of the inverters in the 74LS04. Data output from the modem is inverted by another inverter in the 74LS04 and fed directly to the RS232 input. Although this input is not at RS232 levels, both the *Gemini* and *Nascom* input circuitry cope quite adequately. Dialing information is taken from a port and fed directly to the dialing/isolation box which requires TTL level inputs. A word of warning about the dialing/isolation box. All inputs including the power supply are protected by what appear to be fusible zener diodes, which go short circuit if a voltage across any line exceeds 6V. These devices, which are probably included to protect the computer from lightning strikes on the telephone line are, i) very fast, and, ii) very unforgiving. So care is required to ensure correct working voltages on the dialing/isolation box.

The whole system works extremely well and complements the PRETZEL 2 software nicely. As the whole lot, modem and software cost less than £60.00 and provided a couple of evenings entertainment sorting it all out, I consider it money well spent.

Things your Mother never told you about M80 and L80

by D.W.Parkinson

Having looked at some of the comments on returned 80-BUS questionnaires, this article will please some readers, but irritate others. I trust that those that don't use CP/M-80 and *Microsoft's* M80 and L80 package will bear with me — after all this article may prompt you to go back and read about the more obscure features of your assembler/high level language. The article assumes that the reader has a reasonable working knowledge of M80 and L80.

Microsoft's M80

First I'll start with *Microsoft's* M80 macro assembler. (CAVEAT — The features exploited in these examples are present in release 3.44 of the assembler (09 Dec 1981) and do not necessarily apply to earlier releases.)

M80 includes a pseudo-op — '.printx' — that prints a message on the system console whenever it is encountered during an assembly. If you ever assemble large files, then this can be used to keep you informed of the progress of the assembly and is a useful indicator that things are progressing as they should. The syntax of the command is:

```
.printx <marker><text><marker>
e.g. .printx ★ I/O section reached ★
```

i.e. Following the '.printx' command M80 takes the first non-space (or non-tab) character that it finds, and then sends that character and the following text to the console. It stops when it sees the same marker character again. I tend to use the '★' character as my marker.

As well as indicating when certain 'landmarks' are reached, it can be used to confirm that the assembler is actually doing what you think it is doing, and also that your assembled program matches any particular system requirements. The former obviously covers the case where an assembler program utilizes the conditional assembly feature of M80, and confirms that you have your conditional flags set correctly. (If you have several slightly different versions of a program to match differing environments, then it is easier to maintain one copy of the program than several.) In the latter case, you may require a certain point of your program to fall at a particular address. This can be done with an 'ORG' statement, but you may inadvertently add some extra code to the start of the program some months later, forgetting that this will cause one bit of code to overlay another later on in the program!
e.g.

```
ifdef NASCOM ; If we have defined NASCOM version
.printx  ★ NASCOM version ★
else
.printx  ★ GEMINI version ★
endif
&

....      ; End of section of code
ret
if $.GE.66h  ; Check it ends before start of NMI routine
.printx  ★ ++++ error NMI routine clobbered +++++ ★
endif
org 66h      ; Start of NMI routine
.....
```

Occasionally you may want to know the size of a program — perhaps it has to fit within an EPROM. In the past I used to enter a single line at the end of the program consisting of the one word 'error'. The assembler always threw this out as an error, and from the error message I could see the address that the program had reached. With version 3.44 of M80 a much better way of achieving this effect appeared. It makes use of the macro feature of M80 together with parameter substitution.

Suppose you are assembling a program and want to know several things:

(i) The length of the code segment of the program.
(ii) the length of the data segment of the program.
(iii) the length of a particular routine within the program. (Perhaps in some circumstances it might have to be overlayed by an alternative routine.)

There are two steps you have to take to achieve this:

Step 1 is to ensure that you have labels within the program to which the assembler will assign the appropriate values. (e.g. 'lcode equ $-start' at the end of the code segment.)
Step 2 is to create a macro to print out what you want to see:
e.g.

```
show   macro   a,b,c
.printx  ★ Program code length is a bytes ★
.printx  ★ Program data length is b bytes ★
.printx  ★ Access subroutine length is c bytes ★
endm
```

If you use the macro in the normal way (show lcode,ldata,alength) the result would be the useless messages

```
★ program code length is lcode bytes ★
```

... and so on, with just the direct substitution of

'lcode' for 'a', 'ldata' for 'b', and 'alength' for 'c'. However if you precede the label names with a '%' sign, M80 treats them in a different way. Instead of replacing parameter 'a' by the character string in the corresponding position in the macro call, it replaces it by the VALUE of that character string, (assumed to be a label), converted to the current radix. That brings me to one final 'tweak' to using the macro — we normally think in HEX when confronted with addresses and opcodes, (or at least I do), and so in order that the information is displayed in hex we need to change the current radix to 16. So the code in the source file should look like:

```
.radix  16  ; Change radix to 16
            ;so we get Hex display
show %lcode,%ldata,%alength   ; Use
            ;the VALUES  of the labels
.radix  10  ; Back to normal radix of 10.
```

If you only want to see the message once, rather than on each pass of the assembler, the code can be bracketed by:

```
if2         ; If pass 2
... as above..
endif
```

Microsoft's L80

If you have a large program it is sometimes convenient to split it into several smaller modules. There are then two approaches you can take on combining these modules. The first is to combine them during assembly by using the 'include' command of M80. The second is to assemble them separately, and combine them at link time with L80. In the latter case use is made of the 'entry' and 'extrn' commands of M80 to specify which labels can be referenced from outside a particular module, (the entry points), and which labels are in other modules, (the externals). However this approach may give you quite a long command line for L80 which it can be irritating to keep retyping:

```
e.g. l80 bitmain,bitl,bit2,bit3,
     bit4,bit5,bit6,library/s,fred/n/e
```

Which links together the seven programs bitmain, bit1 – bit6, items from the library file 'library', and saves the lot under the filename 'FRED.COM'. You can save yourself effort by creating a submit file with the above in it, or by programming up one of the function keys on the Gemini keyboard with the command string. Alternatively you can make M80 and L80 do some of the work. This is done by using the '.request' command of M80. The '.request' causes M80 to code-up a request to L80 to search the specified file(s) for the routines to satisfy any undefined externals. In the above example 'bitmain' could be edited to contain the line:

```
.request bitl,bit2,bit3,bit4,bit5,bit6,libry
and as a result the L80 command would reduce to:
L80 bitmain,fred/n/e
```

There are two caveats to using this approach: Firstly M80 restricts external names to six characters, and thus your file names in the '.request' command must six characters or less. If they are seven or eight characters long M80 will truncate them to six characters, and then L80 will be unable to find the files. Secondly L80 does a library *search*. i.e. L80 will only link in those files that contain entry points that are in its table of as yet undefined externals. e.g. If the entry points in 'bit2' are only referenced by 'bit6', then the file 'bit2' will not be loaded when it is reached, as its entry points will not match any of the 'wanted' externals referenced by 'bitmain' and 'bit1'. It is only when the file 'bit6' is reached that the external references of that file to 'bit 2' are added to the 'undefined externals' table. Thus the files in the '.request' list must be in an order which ensures that they are all loaded. (If 'bitmain' references all of them you have nothing to worry about.)

$MEMRY

If you are linking together several routines to form a large program, the problem of workspace may arise. By workspace I mean 'where is the workspace?'. In the case of a single program requiring — say — two 4k buffers the answer is simple. The program can end with:

```
 buff1     equ  $           ; Start of buffer 1
 buff2     equ  buff1+4096  ; Start of buffer 2
bufend     equ  buff2+4096  ; End of buffer area
```

(Note the program ended with EQUs. If DEFS had been used, the data space would have been saved as part of the program — a total waste of disk space.) The same approach can be used when you link programs together with L80, BUT you must ensure the module with the above definition is THE LAST MODULE linked in, (including any library modules), otherwise the 'buff1' and 'buff2' areas will overlay program code. Replacing the EQUs with DEFS would solve the problem, but once again would lead to wasted space on disk.

However L80 does provide a mechanism for getting round this problem, but it is slightly clumsy to use. This is the variable $MEMRY. When L80 finishes linking a program it looks

to see if a global variable with the name $MEMRY has been declared. If $MEMRY exists, then L80 will set it to point to the first byte of free memory following the program. The code below gives a simple example of its use and contrasts it with the straight forward approach.

```
single assembly  multiple assembly
                 entry $memry    ;Define Entry point
 ......          .......
ld hl,buffer     ld hl,(buffer) ;Point at memry buffer
ld a,(hl)        ld a,(hl)      ;Load byte
 ......          .......
ld hl,buffer+128 ld hl,(buffer) ;Point at byte
                                ;128 in buffer
                 ld de,128
                 add hl,de
 ......          .......
 ......          .......
                 $memry equ  $  ;$MEMRY same as BUFFER
buffer equ $     buffer defw 0  ;Define buffer
```

The basic difference is that 'buffer' becomes an indirect reference to the buffer area. There is no real difference in the code when loading HL with a pointer to the base of the buffer area, (one just becomes an indirect load), but the irritation is when you want to load the address of a specific location offset within the buffer area. In the former case the assembler has already done the calculation for you, and you are still only loading HL with an immediate value, while in the latter case you have to explicitly do the calculation in your own code.

Happy hacking!

My dear chap, we seem to have overlooked the fact that 'C' is not as difficult as we first thought.

Converting WordStar text files to improve their readability

by P.D.Coker

One of the problems associated with using WordStar is the lack of clear legibility of the text files it produces, unless they are being edited by WordStar itself. This is perfectly all right on most occasions, but there are times, particularly when deciding to delete or dump a lot of text files, when the tedious business of loading them into Wordstar, reading and then exiting, is a bit too much trouble. Using the TYPE command in CP/M gives a visually disturbing result with lots of inverse video characters at the end of words and no nice tidy paragraphs. [Ed. — this is not true of CP/Ms installed with CCPZ, i.e. all Gemini Winchester based systems, and all Gemini CP/Ms of BIOS 3.0 or later.] This WordStar 'feature' is also not particularly good if you wish to use PEN on a file originally written using WordStar, since the formatting commands used by WordStar are totally different and a lot more time will have to be spent deleting them and substituting the appropriate commands for PEN.

The strange characters which WordStar inserts are used for justification, tabulating and other formatting functions and they are not normally displayed on the screen when WordStar is in use. Characters such as these have ASCII codes greater than 127 (dec.) and are interpreted by the video controller in an 80-BUS system as inverse (black on white) letters or symbols. Normal text and some other symbols used by WordStar have ASCII values from 0 – 127, and the formatting commands will not interfere with these.

There are two approaches to this problem. One is to use the [Z] option in PIP which sets the parity bit to zero; the following command line will read the text file B:RUBBISH.TXT and produce a cleaned-up version on drive A as GOOD.TXT:

```
A>PIP GOOD.TXT=B:RUBBISH.TXT[Z]
```

This works nicely because the eighth bit is set to 1 in characters beyond 127 and these are the ones which cause the peculiar effects. The drawback to using PIP/[Z] is that you cannot see the edited file unless you call it up by using the TYPE command. There may be various tweaks to the PIP command which will produce a listing on the CON: function — in this case, assigned to the CRT: device, but I haven't investigated these, and the thought of typing extended PIP command lines (I haven't got a programmable set of function keys on my machine) didn't appeal!

As far as I am concerned, a much more satisfactory approach is to use a small BASIC program which examines the text file character by character and decides whether or not to print and save the character in a separate, new file, displaying its progress on the screen. It is slower than using PIP but the new text scrolls up at a reasonable rate, so this is not a major drawback.

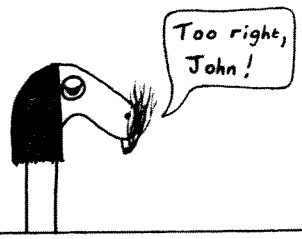
The program asks for the names of the source and destination files and then reads the data from the source file, character by character. The file names must be in upper case letters and if either of them does not reside on the logged—in disk, the disk name must be quoted — e.g. B:WORKING.DOC. If a character is found whose ASCII value is greater than 127, it is converted back to a value less than 127. The program grinds on until it reaches an End-of-File marker, prints out a message and then asks if either the source or destination files are to be deleted.

A trial run will show that certain print control characters are not converted properly — such as ^B, ^D, ^Q, or ^E. This is because their ASCII equivalents are less than 127 (control characters have ASCII codes between 0 and 31). This means that there will be normal video graphics symbols corresponding to these codes in the destination file and on the screen. The overall effect isn't too serious, however, except where a lot of fancy printing is involved in the source file.

```
10 PRINT"WordStar tidy-up program"
20 PRINT:PRINT"File names must be in UPPER
       CASE":PRINT
30 INPUT"Source [dsk:] filename.ext";F1$
40 INPUT"Destination [dsk:] filename.ext";
       F2$
50 OPEN "I",1,F1$
60 OPEN "O",2,F2$
70 Z$=INPUT$(1,#1)
80 IF ASC(Z$)>127 THEN Z$=CHR$(ASC(Z$)-128)
90 PRINT Z$;
100 PRINT #2, Z$;
110 IF EOF(1) THEN 120 ELSE 70
120 CLOSE 1:CLOSE 2
130 PRINT:PRINT"Finished":PRINT
140 PRINT"Delete Source File ";F1$:INPUT R$
150 IF R$="Y" THEN KILL F1$
160 PRINT"Delete Destination File ";F2$:
       INPUT R$
170 IF R$="Y" THEN KILL F2$
180 END
```

The interpreted BASIC program implies that MBASIC.COM must be on one of the drives in the system, and the conversion process will be rather slower than would be the case if a compiled BASIC program was used. With a largish file, the difference in execution time of the .COM version can be less than half that of the .BAS, and one has the added advantage that MBASIC is no longer required. To give some idea of the time taken, this article took 54 seconds to 'tidy' using interpreted BASIC and 26 seconds using the compiled version. To produce the compiled version, one needs access to BASCOM.COM, L80.COM and OBSLIB.REL; the resultant .COM file does not need BRUN.COM — which would be the case if BASLIB.REL were used instead of OBSLIB.

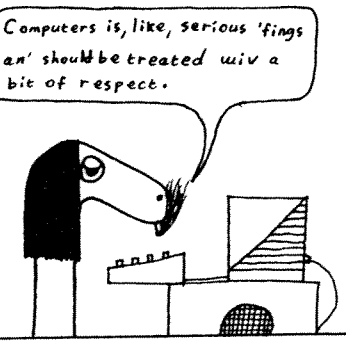
Lawrence Lipshitz - the voice of reason.



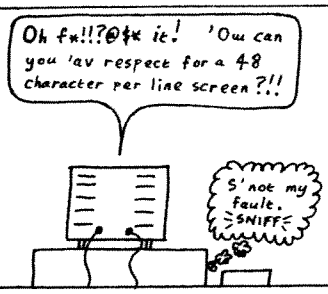
Tap, tap. Hammer, hammer etc., etc.

Really serious computing noises.

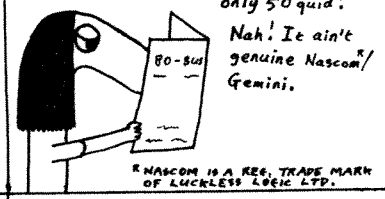
WHERRRee



But....



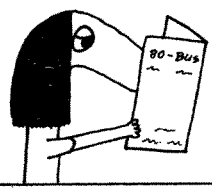
In the **very latest** issue of 80-BUS



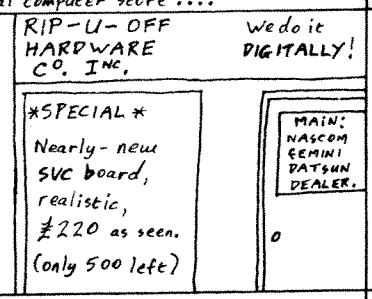
® NASCOM IS A REG. TRADE MARK OF LUCKLESS LOGIC LTD.

Mumble, mumble. Pluto, Nascom AVC, etc., etc.

Woh! Vat's ver one I need, it's a Gemini SVC. Only costs £200+..!!



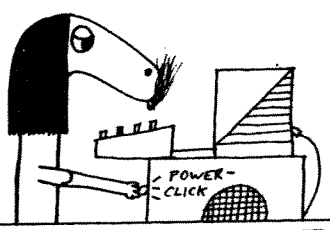
Lawrence, cash in hand, visits his local computer store....

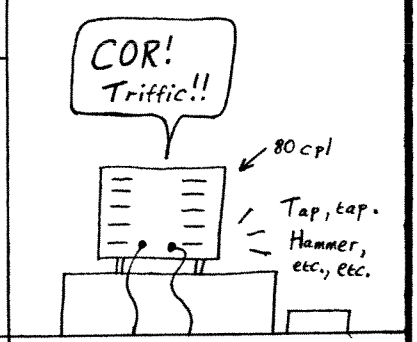


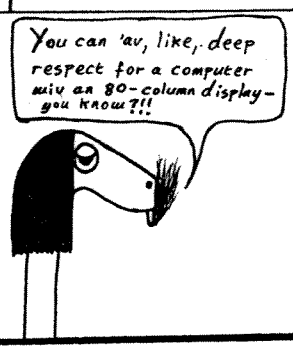
Where a deal is made.

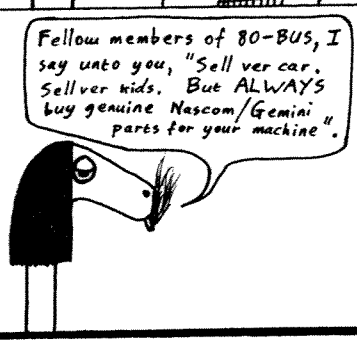
Lawrence, C/W SVC board, goes home....

.... and fits the SVC into his N-1.









This episode has NOT been paid for by Nascom and Gemini Micros Ltd.

If you would NOT like to pay in a similar fashion, contact:-

D.G. Richards. Tonyrefail, Mid. Glam. South Wales.