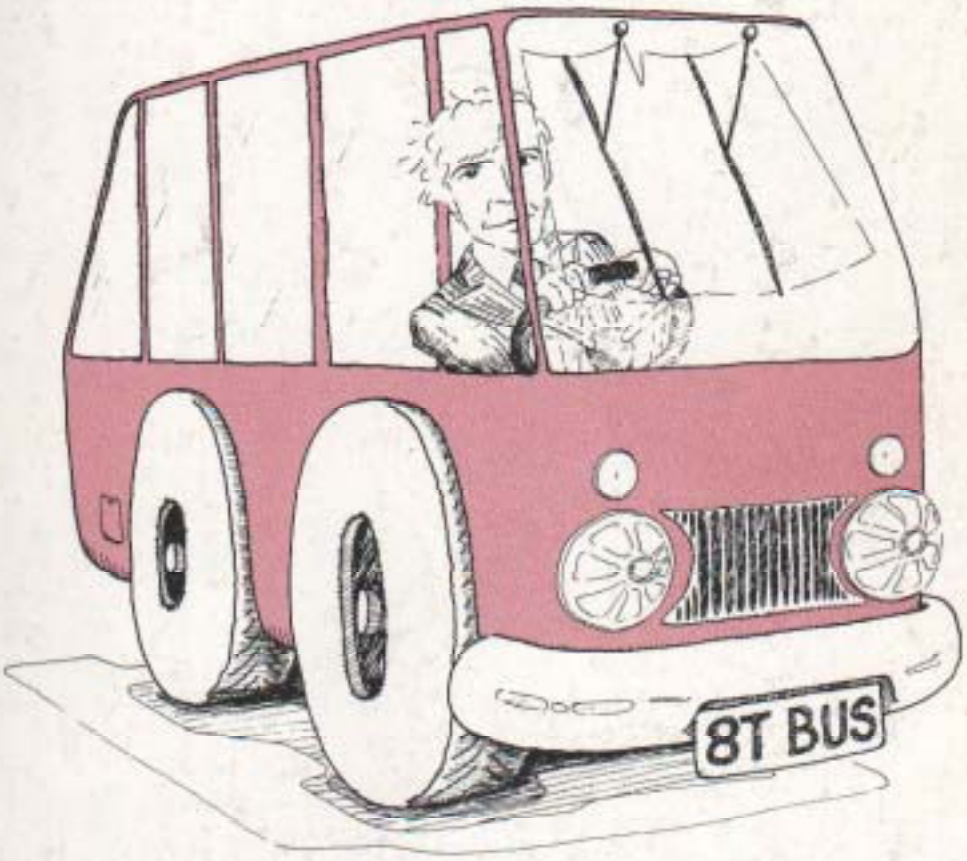
80-BUS NEWS

JANUARY–FEBRUARY 1984 VOL. 3 ISSUE 1

- NASCOM ASSEMBLERS REVIEWED
- POLYDOS FILE UPDATE PROGRAM
- CP/M DIS-ASSEMBLER REVIEWED
- LAWRENCE!



The Magazine for
NASCOM & GEMINI USERS

£1.50

January–February 1984. 80-BUS NEWS Volume 3. Issue 1.

CONTENTS

Page 3	Letters to the Editor
Page 7	Utters from the Gutters
Page 13	Doctor Dark's Diary - 20
Page 16	CP/M File Transfer Via IEEE488
Page 17	Dave Hunt's Section
Page 22	Some Private Sales
Page 23	NASCOM BASIC Disassembled - Part 5
Page 35	Aunt Agatha's Agony Column
Page 40	Some More Private Sales
Page 41	Polydos File Update Program
Page 46	Review of Nascom Assemblers
Page 51	Random Rumours (& Truths?)
Page 52	Lawrence Lends a Helping Hand
Page 53	Ads

SUBSCRIPTIONS

Annual Rates (6 issues)	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.

ADDRESS: 80-BUS News,
Gemini Microcomputers Ltd.,
18 Woodside Road,
Amersham, Bucks.

Letters to the Editor

Please note that occasionally there may be a considerable interval between the receipt of a letter and the time that it materialises in this column. Consequently events that have occurred in the meantime may make the letter out of date. If this is the case we apologise for any inconvenience thus caused.

MBASIC, 16 bit, & other bits

I was pleased to find that at least one 80-BUS user had read my article on disk MBASIC (Letters - 80-BUS News Vol.2, Iss.4).

The version of MBASIC that I use is 5.21 (July 1981) and it really is possible to include comments on the same line as program statements without, as Mr Stuckey suggests, the obligatory colon before the apostrophe. This appears in all manuals I have seen which have been issued after 1977 and applies to both Disk and 'Extended' versions only. It is also available in a number of mainframe BASICs (such as PDP and DEC - which I have used for more than 10 years).

It will be very interesting to see what transpires in the way of high level languages and operating system(s) as support for the promised 80-bus 16 bit CPU card (from Gemini, I understand, although they denied having any such revolutionary ideas a few months ago!). I suspect that Pascal will be available, and possibly Fortran as well (since Prospero Software already have a 16-bit PASCAL available and it would be useful to have a 16-bit version of their excellent implementation of FORTRAN when they get round to it); it would be a pity if some form of BASIC will not be supported, but it seems probable that its use will be confined to 8-bit machines, to die a lingering death. Will we need to spend lots of money on CP/M-86 or will MSDOS be the preferred operating system - and will our 8-bit CP/Ms be totally redundant? What about compatibility with our AVCs, MAP RAM cards and colour cards? I look forward to hearing more about it - price, estimated launch date, etc.

Does anyone actually use COMAL-80? I noticed that Atherton's book was mentioned in the last issue of 80-BUS News - and having used the aforementioned text, I rapidly came to the conclusion that it was better to learn PASCAL and forget that COMAL ever existed since it is, in reality, only a "souped-up" and rather poor subset of BASIC with vaguely Pascal-like structuring. A comparison between interpreted MBASIC and COMAL-80 using the PCW benchmark programs showed that COMAL was significantly slower on almost all the programs apart from BM8 (which involved some work with intrinsic functions) where it was twice as fast as the BASIC, taking 27 seconds to carry out a 1000 times loop with log, sine and exponentiation.

Yours sincerely, P D Coker, Farnborough, Kent

[Ed. - I note from a magazine that Prospero are advertising their Pascal and Fortran in both 8 and 16 bit forms - it is to be hoped that the 16 bit versions are not just code conversions of the 8 bit versions, if this is the case they are likely to be slower!]

Angry at Nascom

I am writing to you to air my anger and frustration at Lucas Logic. I have been a faithful supporter of Nascom for 5 years, but Lucas Logic's absence from the PCW Computer Show was the final straw.

With the launching of two new products - the LX Printer and the LX80 MicroComputer - it was the ideal platform to advertise. Obviously they couldn't give a damn!

Lucas Logic apparently have lost their commitment to Nascom Microcomputers and products. Their relationship with Dealers is appalling.

The sooner Nascom is bought by a decent company the better. In my opinion in their hands Nascom is dead.

Yours in extreme anger and disappointment,
Dr D Plews, MB ChB, Keighley, W Yorks

Rory a Sadist?

Whilst reading Rory O'Farrell's article "Happy Talk", I got the impression that he is some sort of sadist. The time taken to load a file into a disassembler, save it into 4K blocks as a hex file, then finally to transfer the file, must be a least 10 minutes per block.

For some time now, I have been working on a quick and cheap data transfer program for the home enthusiast. The transmitting of data is done via a BASIC program which reads any file type and transmits via the serial port. For reasons mentioned in Rory O'Farrell's article, the program cannot transmit pure binary object code, it is therefore necessary to convert the file to some form that may be recognised by PIP and not to send it Control Zs. The file is received on the second machine via the RDR: into a file. A small BASIC program is then required to convert the program back into its binary form.

Finally, a little tip for NASCOM CP/M users using the AVC board. When most people boot-up CP/M they type AVCTXT straight away to give an 80 column screen, or modify the BIOS via the config program to execute a file AUTO.SUB. You then create the file AUTO.SUB and enter the command AVCTXT.

Have you ever wondered why it takes 15-20 seconds to execute AVCTXT after boot-up? The reason is simple, when Auto load is specified, it sets a jump in the BIOS, the address jumped to is to load SUBMIT.COM, it is then the job of SUBMIT.COM to load AVCTXT and execute it, therefore, taking time loading two files to achieve one goal.

Pondering on this, I decided to delve into MOVCPM. I found the bytes which loaded SUBMIT.COM and changed these to AVCTXT and resaved MOVCPM. CP/M now boots-up AND executes AVCTXT in less than 5 seconds. The trick is:

```
DDT MOVCPM.COM
- D 0A00
          Notice that locations 0A08 - 0A17 contain SUBMIT AUTO.
          Change these locations to the following:-

     0A07            -  This is the length of the command to be executed
     0A08 41 A
     0A09 56 V
     0A0A 43 C
     0A0B 54 T
     0A0C 58 X
     0A0D 54 T
     0A0E 2E

SAVE 41 MOVCPM.COM
MOVE 55 *
SYSGEN
SOURCE (press enter)
DESTINATION B
```

Yours faithfully, P.A. Dutton, Northfield, Birmingham.

Re:C.B. A.K.A. Dr.D

In a recent hard-copy interchange with that super-programmer, peerless hardware hack and expert wine connoisseur Chris Blackmore, the subject of software exchange loops cropped up. Chris suggested I write to you and deal directly "with the Mafiosi who have a firm grip on the throat of 80-BUS NEWS" concerning non-CP/M exchange loops. Right, here goes.

My N-1 (yes, there are still a few around!) is fitted with a GM805 disk drive and runs D-DOS (boo, hiss, boo), DCS-DOS and POLYDOS. I am currently in disk-to-disk contact with one Alan Wood Esq of St Mellons, Cardiff, using both DCS-DOS and POLYDOS. If an exchange loop exists for either or both of these DOS formats, would you put me (us) in contact with the organisers. Alternatively, if said loops do not exist you could direct any interested parties my way and I'll have a bash at setting a loop (or loops) going, 'like wot yer doc. did for CP/M'. Ah, fame at last.

I will start an exchange loop, and I will also name it unto you. And it will be called ..... TOROID FERRUGINOUS, the greatest software exchange in the history of time and space (or, the biggest anticlimax EVER).

Another point raised in conversation was the inordinately long time taken by 80-BUS to cough up the readies for articles published. Now I know that 99% of the contributors who write to you do so just to see their names in print and/or to help other readers. But please take note, C.B. and D.G.R. DO IT FOR VER MONEY!! Why do you think Chris Blackmore makes home-brew wine? It's because he's too poor to visit the local wine bar, okay, ya. And my Nascom rusts in the corner 'cos, 'cos, I can't afford to pay yer 'lectric bill. And all this just because you employ a disslexic, er, dislexic, um, diss-lexik accountant who doesn't know how to sighn, er sign a cheque!. I'm seriously thinking of trading in my Nascom ..... for an I.B.M. 370!

Happy hacking, D.G. Richards, 29 Martin Crescent, Tonyrefail,
Mid. Glamorgan, South Wales, CF39 8NT.

Re. Easicomp Board

With regard to Mr A Brown's information on the Easicomp Sound Board, in part 2 of your I/O map (Vol.2, Iss.4), please note that, although the manual states that ports 10 and 11 (decimal) are used if NASIO is provided, I find that the board actually decodes ports 130 and 131 (82H and 83H). I assume from this that Easicomp changed the design of their board at some point, and hence boards of both types exist. Incidentally, in common with the PSG design published in INMC 80 News, Issue 5, I now use ports 8 and 9!

Many thanks for printing the "Nascom ROM BASIC Dis-assembled" articles! Although I had dis-assembled this previously, the added documentation is very helpful.

Regarding the contents of your magazine, I feel you are leaving many of your readers behind by printing lengthy articles on very specific topics. This is an unfortunate side-effect of the flexibility of an 80-BUS system. What about including a short questionaire as part of your subscription form, to get a better idea of what equipment people own?

On a different note, I would be interested in Doctor Dark's program exchange idea, assuming you have had replies from anyone else interested in using tapes. While on the subject of software, I notice that you have been publishing very little software and even fewer software reviews. Is this because no-one is writing programs for the Nascom anymore?

Yours sincerely, Kevin Smith, Aberdeen.

Dealing with RFI

I am glad to see that the problem of RFI is being considered seriously (80-BUS News Vol.2, Iss.5) and delighted to learn that we may get an issue devoted to Amateur Radio. For a starter, may I suggest the following additional hints on the problem of RFI.

1) A metal case to enclose your computer is vital. Mine came from a Radio Rally, custom-made out of 16swg steel except for a 1/8th aluminium front panel for #15. This vast box measures 19x14.5x9 inches and leaves a lot of room for all the expansion boards.
2) Multi-core screened cable for keyboard, printer and etc, may be obtained from Maplins whose catalogue may be read (and purchased) at W.H.Smiths. Screened cable is a must for stopping RF radiating from lovely aerials dangling from your computer.
3) Ferrite rings (try Ambit International) are effective. All leads (power, cassete, tv/monitor) emerging from your case to be wound a few times around the rings, as near as the case as practicable.
4) A mains filter (Ambit). This is effecitve both ways. It stops RF escaping into the mains where the house wiring acts as a massive aerial and prevents the mains crud getting at your computer.
5) Desperate measure. In Wireless World, September 1983, G3NRW recommends a 150 pF capacitor between the +12 V rail and ground.
6) Another desperate measure. Try earthing all your bits (computer proper, tv/monitor, cassette, etc) at only one point.

RFI appears worst for amateurs at 70 MHz. I achieved a measure of success for that band with some of the above, plus critical computer/radio spacing and complete success by executing HALT! ON 144 MHz, I have a few S1 whistles, but presently the lid is off the box and a Hobbit mechanism dangling on an unscreened lead.

By the way, has anyone made Hisoft Pascal talk to the Hobbit?

Yours, G. Orford, Bristol.

A Lunatic Writes

Dear Dirtbags,

Oldsters, who are, like, total vacuum-heads and do not understand the problems of a young person of today, often get the wrong idea about Waldo 'D.R.' Dobbs, who is like me, man. They think that I have no romance in my soul, which is, like, this incredibly strange thing that I have inside my body. To them I say: "You are totally incorrect, man". It is, like, the total lack of 80-BUS NEWS that causes these sauzzball manifestations, man! Why, when I see these utterly horrible PLASTIC BOXES, I am, like, totally overcome with emotion. There are these worthless diseased MOLLUSCS all over the magazine stands in W.H. Nasties, reading about these disturbingly repulsive micros, man.

Degenerate reptile, man, I ask you to stop this senseless deprivation of REAL COMPUTERS that you are, like, doing to me, man.

Monstrously, hazardously, like, yours, man.

Waldo 'D.R.' Dobbs [No address given.]

Utters from the gutters

By Mick Waters

Before getting down to the meaty bits, I should say that I am one of Dr. Dark's dodos in that I have never before submitted anything for publication. In the past I haven't subscribed regularly to either the INMC or 80-BUS News and used to pick up my copies about once a year on my rare excursions to one of the London dealers. This year I made a resolution to persuade (con) a relative into buying me a subscription as a Christmas present and so now I should be able to read the latest scandal almost as it happens. I don't pretend to be an authority on Nascom hardware but have managed over the past five years or so to become reasonably acquainted with the way they work. Mind you, being an RAF technician, I am an expert on modern electronic techniques (circa 1960) and can write volumes on pentodes, triodes and other glass encapsulated transistors with heaters. Since I have been about 12 months behind everyone else, this article may be too late for solving the problems of a couple of readers who asked for advice as far back as the March/April 83 issue but here goes regardless.

Nascom/SIMON

The first problem(s) of interest came from S. Willmott and from the Sept/Oct 83 issue of Dr. Darks diary. First Dr. Dark, your problem with SIMON getting in the way when verifying a page-mode style RAM-DISK appears to me to be due to the fact that the Nascom computers do not support page mode and SIMON will be present (and generating a RAMDIS signal) no matter what page of RAM is selected. The answer? Read on.

Mr Willmott asked how SIMON may be removed from the system once its job is done. I did (yet) another mod to my N2 as I objected to losing 4k of RAM and also to the prospect of buying a page mode EPROM board for one chip. My solution is not very elegant but it works. Below is a table of advantages and disadvantages which should be weighed up before going any further:

For it

1. Its CHEAP. (Free in fact)
2. It doesn't involve hacking your faithful Nascom around.
3. It works.

Against it

1. The tape drive may no longer be used for anything else.
2. SIMON doesn't work apart from booting the system up as it effectively switches itself out.
3. You must power up with a disk in drive A.
4. Pressing RESET doesn't work any more. The only way to reset is to switch off then switch back on. This creates problems when a program goes into a loop as you can't then RESET and examine the memory.
5. This mod. may only be used if an IVC is fitted as chips are borrowed from the redundant Nascom video circuitry.

In my opinion, the advantages outweigh the disadvantages, you may not think so. If you decide to try this mod, proceed as follows:

1. Pull out the chips used for the video circuitry except IC8.
2. On LKS1 remove all links except for the link between pins 3 and 14.

3. Connect a wire from TP10 to LKS1 pin 4.
4. Remove IC8 and bend pins 1, 2 and 3 (carefully) out straight so that they will clear the socket and replace IC8.
5. Remove IC71 and do the same with pin 8.
6. Connect a wire from IC71/8 to IC8/2.
7. Touch solder a wire onto IC24/12 and connect the other end to IC8/1.
8. Touch solder a wire onto IC18//13 and connect the other end to IC8/3.

The effect of this mod is only to provide a chip select and a RAMDIS signal when the tape drive is on. Certainly on my machine, it is always on when powered up so SIMON is connected in and any RAM in the system occupying those addresses is disabled. Once SIMON is entered other than for booting up or once CP/M is loaded and started, port 0 is reset and SIMON disappears from the memory map. Plug in and power up and all should be as before (on a 60k max CP/M) provided SIMON is not used as a monitor. If the system is powered up without a disk in drive A, your machine will crash once port 0 is reset by SIMON.

Before a full 64k system can be installed, the cold boot loader must be modified to switch out SIMON before loading the CP/M system. To do this, perform the following sequence of operations:

1. Use MOVCPM to generate a 64k system. Allow room for SYS if you use it. Save the result as CPM64.COM.
2. Load CPM64.COM under DDT or ZSID.
3. Use the S command to modify the addresses given below to the values indicated.

   090B 18 097C 3E
   090C 6C 097D 01
   0979 AF 097E 18
   097A D3 097F 8D
   097B 00
4. Use ^C or G0 and save the modified CPM64.COM.
5. Use SYSGEN to put the file on drive A. This can be done by typing:
   SYSGEN CPM64.COM
   after which, specify drive A as the destination.
6. Unplug & plug in again with the new system in the A drive and Hey Presto, you should have a 64K system up and running with no SIMON in the way. As an added bonus, it won't get in the way when paging RAM either.

Faulty RAM Board

My second bit of first-aid is probably of no use to Kevin Weatherhead who has, by now, probably cured his faulty RAM board himself. If not then here goes.

Kevin doesn't say which type of RAM board he is using, so I shall go into the Nascom RAM A & B boards and the Gemini GM802 64k RAM card. The problem is this. When Kevin powers up, his RAM card is full of rubbish as expected. He can write zeros to the card but not ones. Fortunately, all three RAM cards use the same principle so only a brief "how it works" is necessary.

When the Z80 wants to read data, the address lines contain the required address. This address is decoded by the RAM cards to select the appropriate RAM chips. The /WR signal from the Nasbus/80-bus provides the /WR signal to the RAM chips so that they accept data rather than send it. This signal will be a '1' when reading and this side appears to work. As Kevin can also write to the chips, albeit in a limited fashion, this appears to be switching correctly. Data to be written to the RAM chips is gated into the RAM after

buffering by a 74LS244 whose enables are always tied to 0v. When reading, the /RD signal is used to enable a similar LS244 for buffering the data leaving the card. Assuming that the LS244 is faulty, swapping the two buffers should give an indication of whether this is so. If on power up, all of the RAM appears to contain either all ones or all zeros and no attempt to change them succeeds then replace the LS244. A table of chip numbers for all three RAM cards is given below.

LS244's used as write buffers:

RAM A card	RAM B card	Gemini G802
IC1	IC26	IC44

If changing the chip specified doesn't cure the problem then I can only suggest that the board is returned to the manufacturers via your dealer.

CCPZ

The last bit of what appears to be a takeover bid by me concerns three bits of software that should by this time, be available from Henrys. The first item is about CCPZ. As stated by Dave in a previous issue, there is a bug in all versions (with the possible exception of the Gemini release version) up to V4.1. After this version, CCPZ was made available in Macro-80 source code (hooray) and had the bug fix in it together with an implementation of the "." command. Since then, another bug has come to light when using SUBMIT files with the GET command. Try typing:

```
GET ADDR FILENAME.TYP
DIR A:
DIR B:
```

If you get the first command repeated three times then you have the bug. This is because CCPZ changes the default DMA address when getting the requested file and forgets to change it back. The result is that further SUBMIT commands are read to memory above the last file read. CCPZ then checks the default buffer and of course finds the last command which is repeated. With version 4.3, this bug has been fixed and there is no reason why everyone shouldn't be using CCPZ now. Order yours tomorrow! (No I don't work for Henry's).

BDOSZ

My second bit of software news concerns a Z80 implementation of the CP/M BDOS, by some strange coincidence, called BDOSZ. In true CCPZ fashion, the space saved by doing the conversion has made room for extra goodies while still retaining CP/M 2.2 compatibility. This package came about as a result of the good fortune of Chris Bellingham of Canterbury, who found a part completed source version of something which would behave rather like the CP/M 2.2 BDOS (author unknown) in an ancient file tucked away in the memory banks of the VAX mainframe at the University of Kent. Chris was at that time running CP/M 1.4 with a second hand Henelec FDC and drives. Being on a students grant, he had never considered the change to CP/M 2.2 as being economically viable. Having found the BDOS and typed the source into his Nascom, there followed a good deal of hard work to finish the BDOS and debug it. Once completed and armed with CCPZ and a home brew BIOS, he had CP/M 2.2 for next to nothing.

At about this time, I obtained a copy for interest value. The error reporting was as Chris originally found it and if possible was worse than that in DRI's BDOS. Also, at the end of the BDOS was a large unused area which we thought might be able to provide some extra goodies. As Chris was busy, I got the job of upgrading BDOSZ to something that would cure the limitations placed on users by the standard CP/M BDOS.

After the usual amount of swearing and cursing, it was finished and working, the results of this are now available at a very modest price.

Now for what it does:

Users of CP/M will have realised by now that the BDOS's error trapping and reporting leave a lot to be desired. How many times have you tried to erase or change a file only to find that either the disk or file is read only? Worse still, if after 6 hours processing, your latest masterpiece tries to update its data file and finds it read only. Unless you have the facility to change the attributes, all of that time has been wasted because you can only reboot, set the file to R/W and start again. Finally, have you got any source files that you would like to assemble, producing listing files to disk, but the PRN file is larger than the capacity of your disk? Infuriating isn't it that you can't just change the disk and carry on?

BDOSZ attempts to overcome these and other problems so that your valuable time is not wasted and your trusty Nascom or Gemini isn't kicked around the floor in temper. An example of each of the BDOS error types and BDOSZ's actions in these cases are given below:

BDOS Err On x: Select

In this case, BDOSZ outputs the message "Drive x: select error" followed on a new line by "Enter valid drive or ^C". The BDOS will then wait for a key press and either log in the selected drive or perform a warm boot as appropriate. This facility is useful if a program being debugged causes a select error and the user wishes to continue testing it.

BDOS Err On x: R/O

In the case of a R/O disk, BDOSZ will print the message "Disk x: is set R/O" followed by "Do it anyway? (Y/N/^C)". If the user types "Y" then the disk will be reset and the function completed as though the error had never existed.

BDOS Err On x: File R/O

Under CP/M, if the command "ERA *.*" is given and one of the files to be deleted is R/O then the above message will be displayed. The CP/M BDOS doesn't tell you which file it is referring to. Under BDOSZ, the message "File FILENAME.TYP is set R/O" followed by the message "Do it anyway? (Y/N/^C)". If the user types "Y", the files R/O attribute bit is cleared and the file will be deleted, renamed or written to as requested as though the file had been R/W. If the user types "N" then the queried file will be left unchanged and still R/O. If, as in the above example, the command ERA *.* was issued, this facility allows the deletion of all R/W files and selected R/O files in the same command.

NOTE. BDOSZ will not query a file unless it is R/O.

BDOS Err On x: Bad Sector

This error is not considered recoverable and most BIOS's these days contain facilities to re-try. If your BIOS can't sort out the problem, then the BDOS has no chance. However, in an attempt to provide meaningful error messages, BDOSZ will display either "Disk x: read error" or "Disk x: write error". In either case, if ^C is pressed, a warm boot will be performed. Pressing any other key will cause the error to be ignored.

Other Errors

Two other errors of a potentially disastrous nature are treated as recoverable by BDOSZ. These are disk directory full and disk full. In the

former case, the BDOS will print the message "Disk x: directory full" and in the latter "Disk x: full". In both cases, the message "Change disks? (Y/N/^C)" will be displayed. If the user types ^C, the system will perform a warm boot. If "N" is typed, a standard "disk full" or "directory full" code will be returned to the CCP or calling program. If "Y" is typed, in the case of a full disk, the currently addressed file will be closed. In either case, when the BDOS is ready to proceed, it will display the message "Change disks then hit any key or ^C". If the user types ^C, the system will be rebooted. If not then a new disk should be in the current drive and BDOSZ will reset the new disk to make it R/W, log it in, create a new file on this disk with the same name as before and continue with the write. The calling program will not be aware of the disk change. BDOSZ will erase any file of the same name occurring on the replacement disk unless it is R/O. In this case, BDOSZ will query before deleting.

As standard, BDOSZ sends a BEL character (ASCII 7) to the console device for those users with a bleep facility to warn the user that an error has occurred.

BDOSZ has been in use now for about 12 months on three different machines. No bugs have been found to date and the extra facilities provided have proved invaluable.

[Ed. - a warning. Owners of Gemini systems with BIOSs of version 2.8 or greater beware! These BIOSs do a check on power-up to determine if, and how many (up to 4) Gemini GM833 'RAM-DISK' boards are present. If any are present the BIOS copies the CCP and BDOS to drive 'M' and modifies the BDOS to use this for warm boots. Whether this will work with BDOSZ or not, who knows?]

MDIS

The last piece of this software trilogy is for those of you tempted by Henrys adverts for MDIS. For those who haven't seen the ad., MDIS is a CP/M disassembler with differences. It includes all of the excellent features found in David Parkinson's NAS-DIS and then some. The last version of MDIS to be released was 2.1, version 2.2 was substituted after a small unexpected "feature" was discovered. But now, version 2.3 is available with oodles of extras. [Ed. - Since the time of writing, vers. 2.6 has appeared.] To those who haven't yet obtained a copy, you were possibly right to wait. For those who have, never mind, upgrades are available at the cost of a copy charge. Simply return it to the dealer you bought it from. Being somewhat biassed in favour of MDIS, I wouldn't attempt to review it but will provide a list of its features (note that features in this case isn't in quotes).

1. MDIS produces either Z80 or 8080 mnemonics as requested by the user but will default to the mnemonics used by the CPU in use.
2. Assembler source files may be produced. Current versions provide 100% compatibility with the Microsoft Macro-80 assembler.
3. Listing files may be directed to the CON:, PUN: or LST: devices or may be sent to a disk file.
4. Labels are produced automatically. Labels are a four digit hex number related to the address where the label is to be inserted. To make the labels assembler compatible, they are given an alphabetic prefix. One of four prefixes will be used for each label depending on whether MDIS thinks the label refers to code, data, both or doesn't know.
5. A cross-reference listing may be supplied which lists each label and each address where the label is used.
6. Allows data areas to be specified as either hex bytes, ASCII, an address table (with labels substituted for addresses) or a look-up table in the form:

byte
address
byte
address
Again, labels will be substituted for addresses in look-up tables.

7. On-screen editing using the cursor controls is available on all line-inputs to MDIS.
8. MDIS works with XSUB. (One in the eye for DISZILOG).
9. Allows tables of data area addresses and types to be redirected to come from a text file on disk. This facility may be combined with input under XSUB.
10. Screen paging is available if required but not to SYS standards. (Who needs both?)
11. MDIS allows printed listings to be stopped after each page so that single sheet paper may be changed.
12. Margins may be specified on all output listings so that mounting in ring binders is possible. A unique feature enables alternate pages to have margins so that if single sheet paper is in use, both sides may be used.
13. Listings are formatted so that data areas are separated from code by a blank line to improve readability. Additionally, blank lines are also inserted after JP (ss) instructions and any unconditional jump, jump relative or return instructions.
14. MDIS decodes NAS-SYS ,restarts correctly.
15. Disassembles right up to FFFFH and will not overflow to 0, unlike many other disassemblers.
16. MDIS allows selected portions of the object program to be disassembled (a useful subroutine for example) and will produce a stand alone program for reassembly.
17. MDIS permits the user to enter titles and subtitles which will be used on page headings or after the appropriate pseudo-ops on Macro-80 source files.
18. The user is allowed to define his own edit keys (even on systems without user defineable keys on the keyboard).
19. MDIS will accept a printer initialisation string to be entered for ease of use.
20. A user subroutine area has been included for those special initialisation jobs (like redefining the numeric keypad as cursor controls on the Rotec keyboard - a routine to do this is provided free).
21. MDIS works on ANY system running CP/M-80 [properly!!! Superbrains don't like it. - Ed.].

At #50, MDIS surpasses all other CP/M disassemblers in both performance and price. Too good to be true? Most of the Microvalue Dealers will provide a free demonstration to callers on request. Finally, those of you who have versions 2.1 or 2.2 will know that MDIS was protected against disassembling itself and against copying to other systems. In spite of the possibility of rip-offs, version 2.3 has had these checks removed for two reasons. Firstly, it was causing problems for some dealers where some computer manufacturers didn't serialise their CP/M systems properly and second, I was feeling hypocritical as how do you think I obtain those pieces of software that I don't write for myself.

DOCTOR DARK'S DIARY – EPISODE 20.

Well, I never thought all those centuries ago that I would end up writing this many articles for the magazine! I think I owe you all thanks for putting up with me for so long, and to show you that you should not have done, here is my latest load of waffle.

I read the other day, somewhere or other, that the Nascom 1 is alive and well, and costs #50 in kit form from Lucas. You can't keep a good machine down, and that's a fact. You tell the kids of today you had to pay over #200 for one, and they'll laugh in your face...

Another thing I chanced upon, in a weekly publication called "Computing", was a review of Boris Allan's book about Logo. At the bottom of the page was a picture of lots of school children in a room with lots of Nascom 3's. It was nice to see that all the claims made by Acron about their monopoly are not 100% true after all. Apparently, Nasnet works. I have tried to use Acron's Ekonet (the names have been changed so that I won't be sued by the guilty!) but it is slower than first class post, when more than two people are using files on the same day!

And now, down to serious stuff, probably not a moment too soon! The Pluto graphics board is short of decent software, unless the people who are writing it don't want it reviewed, and are advertising it in something I don't read. Before you can get round to writing any really big programs for the thing, you need some sort of standard software interface to it. The one that follows is not quite complete, and one day I will get round to writing the definitive version. I have tried several times to write something like this, and have learned several interesting things from my failures along the way. My first versions were in assembler,.and very effective, but I wanted something on a higher level, for ease of programming. The first attempt in Pascal was going to have a separate procedure or function for each of the functions of the Pluto board itself. As the set of routines grew longer, I kept compiling them, to check for errors. At a certain stage, the Hisoft Pascal 4 compiler became angry, and would not compile the end of the program. It works well enough with longer programs made up of fewer routines, so I can only assume that a stack somewhere is filling up, or maybe it is a table of addresses that gets too full? Anyway, it was a very ugly program, so I began again with a different approach.

What I really wanted to be able to do was write something along the lines of "pluto(plot,42,42)", in which the name of the required routine and the necessary parameters were passed in the manner shown. But not all of the routines take the same number of parameters. And some return values, while others do not. Pascal does not provide a construction that will let you do anything so variable. So the version that follows is given only the routine name in the call, and a set of global variables that mimic the Pluto board's internal variables must be set to the values you would have passed as parameters, had that been possible in Pascal. All clear? I thought not. If you want to set the current colour to red, you have to write:

```
ccol := red;
pluto(sccol)
```

This is more or less friendly to use, but not by any means perfect! If anyone has any good ideas about how to improve matters, I for one would be glad to see them.

I know it is not as pretty as it should be, but it does work. At least, those parts that don't say "not yet written" work! Any comments will be read with interest, and offers of Pluto Palette boards to test the next version with will be met with considerable grovelling.

```
PROGRAM pluto;
CONST
{Pluto port addresses.}
status = 160; data   = 161;
{Pluto routine numbers.}
allocp = 163; arc    = 193; bfill  = 176; bfils  = 177;
bfilsp = 179; clrcwp = 172; copy   = 133; copyts = 132;
ffill  = 130; ffilp  = 174; ffils  = 173; ffilsp = 175;
ibcol  = 144; iccol  = 141; icp    = 150; icdp   = 148;
icsp   = 146; icwp   = 147; ifcol  = 143; ipat   = 183;
ipcol  = 187; irsel  = 185; istat  = 134; istyle = 142;
itcol  = 145; iwprot = 149; limage = 159; limagc = 188;
liner  = 157; liners = 158; lineto = 128; lsym   = 160;
lsymc  = 189; mover  = 152; movers = 153; moveto = 151;
pinit  = 166; plot   = 154; plotr  = 155; plotrs = 156;
rfill  = 129; rimage = 161; rimagc = 190; rpix   = 167;
rpixr  = 168; rpixrs = 169; rsym   = 162; rsymc  = 191;
sbcol  = 135; sccol  = 137; scdp   = 131; scsp   = 164;
scwp   = 165; sfcol  = 136; sfpatr = 180; sfpats = 181;
shires = 170; slores = 171; spat   = 182; spcol  = 186;
srsel  = 184; sstyle = 138; stcol  = 139; swprot = 140;
{Pluto colours.}
black   = 0; green   = 1; blue    = 2; cyan    = 3;
red     = 4; yellow  = 5; magenta = 6; white   = 7;
{Pluto pixel block maximum sizes, to set maximum size
 of the array used by pixel load and save routines.}
pixwide = 10; {Or other value.}
pixhigh = 10; {Or other value.}
TYPE
{Pluto colour type.}
plutocol = black..white;
{Pluto subrange types.}
byte = 0..255;
xcoord = 0..639;
ycoord = 0..287;
xinc = -639..639;
yinc = -287..287;
xincsh = -127..128;
yincsh = -127..128;
horiz = 1..640;
vert = 1..288;
VAR
{Pluto colour variables.}
ccol, bcol, fcol, tcol, pcol : plutocol;
{Pluto partition and workspace variables.}
cwp : 1..2;       {Current working partition.}
csp : 0..255;     {Current symbol partition.}
cdp : 1..2;       {Current display partition.}
cpx : 0..639;     {Current position, X value.}
cpy : 0..287;     {current position, Y value.}
{Pluto miscellaneous variables.}
stat : byte;      {Pluto status variable.}
pat : 0..255;     {Line and arc pattern.}
{Single colour plane operators.}
wprot : 0..7;     {Write protect mask.}
rsel : 0..7;      {Read select mask.}
{The style variable.}
style : 0..255;
{Pluto parameter variables.}
X : xcoord;       {Absolute X, Y coordinates.}
Y : ycoord;
DX : xinc;        {16 bit X, Y displacements.}
DY : yinc;
dx : xincsh;      {8 bit X, Y displacements.}
dy : yincsh;
width : horiz;    {Width and height.}
height : vert;
n : byte;         {General 8 bit number.}
p : byte;         {Partition identifier.}
c : plutocol;     {General colour returned.}
{Parameters for the copy routine.}
xfrom, xto : xcoord;
yfrom, yto : ycoord;
pfrom, pto : byte;
{Parameters for the arc routine.}
arcxc, arcyc : INTEGER;
arcxe : xinc;
arcye : yinc;
{Parameter block for the pixel block commands.}
pixels : ARRAY [1..pixhigh,1..pixwide] OF plutocolour;
PROCEDURE sendword(w1, w2 : INTEGER);
{This sends two 16 bit values to the data port
address of the Pluto board. As I am using the
HSA-88B board and Hisoft Pascal 5, it has to do
a bit of messing round to get the 16 bits we
want from the 32 actually stored!}
BEGIN
  OUT(data,PEEK(ADDR(w1)+2,CHAR));
  OUT(data,PEEK(ADDR(w1)+3,CHAR));
  OUT(data,PEEK(ADDR(w2)+2,CHAR));
  OUT(data,PEEK(ADDR(w2)+3,CHAR))
END;
PROCEDURE senddxdy;
BEGIN
  OUT(data,CHR(dx)); OUT(data,CHR(dy))
END;
PROCEDURE setdefaults;
BEGIN
  ccol := 7; bcol := 0; fcol := 7; tcol := 7;
  pcol := 7; cwp := 1; csp := 255; cdp := 1;
  X := 0; Y := 0; pat := 240; wprot:= 0;
  rsel := 7; style := 128
END;
PROCEDURE pwait;
{This waits until the Pluto board is ready.}
BEGIN
  WHILE INP(status) < CHR(128) DO {nothing}
END;
PROCEDURE notyet;
BEGIN
```

```
  WRITELN('Called routine is not yet written!');
  HALT
END;
{
And now the actual routine!
---------------------------
}
PROCEDURE pluto(com : byte);
VAR
  i, j : INTEGER;
BEGIN
  pwait; {Wait until Pluto is ready.}
  {Send Pluto command.}
  OUT(data,CHR(com));
  {Now send parameters, if any.}
  CASE com OF
    allocp : BEGIN pwait; sendword(width,height);
                   OUT(data,CHR(n)) END;
    arc    : BEGIN pwait; sendword(arcxc,arcyc);
                   sendword(arcxe,arcye) END;
    copy   : BEGIN pwait; sendword(width,height);
                   OUT(data,CHR(pfrom));
                   sendword(xfrom,yfrom);
                   OUT(data,CHR(pto));
                   sendword(xto,yto) END;
    copyts : BEGIN pwait; OUT(data,CHR(n)) END;
    limage : BEGIN pwait; sendword(width,height);
                   FOR i := 1 TO height DO
                   FOR j := 1 TO width DO
                   OUT(data,CHR(pixels[i,j])) END;
    limagc : BEGIN pwait; notyet END;
    liner  : BEGIN pwait; sendword(DX,DY) END;
    liners : BEGIN pwait; senddxdy END;
    lineto : BEGIN pwait; sendword(X,Y) END;
    lsym   : BEGIN pwait; OUT(data,CHR(n));
                   FOR i := 1 TO height DO
                   FOR j := 1 TO width DO
                   OUT(data,CHR(pixels[i,j])) END;
    lsymc  : BEGIN pwait; notyet END;
    mover  : BEGIN pwait; sendword(DX,DY) END;
    movers : BEGIN pwait; senddxdy END;
    moveto : BEGIN pwait; sendword(X,Y) END;
    pinit  : BEGIN pwait; setdefaults END;
    plot   : BEGIN pwait; sendword(X,Y) END;
    plotr  : BEGIN pwait; sendword(DX,DY) END;
    plotrs : BEGIN pwait; senddxdy END;
    rfill  : BEGIN pwait; sendword(width,height) END;
    rimage : BEGIN pwait; sendword(width,height) END;
    rimagc : BEGIN pwait; sendword(width,height) END;
    rpix   : BEGIN pwait; sendword(X,Y) END;
    rpixr  : BEGIN pwait; sendword(DX,DY) END;
    rpixrs : BEGIN pwait; senddxdy END;
    rsym   : BEGIN pwait; notyet END;
    rsymc  : BEGIN pwait; notyet END;
    sbcol  : BEGIN pwait; OUT(data,CHR(bcol)) END;
    sccol  : BEGIN pwait; OUT(data,CHR(ccol)) END;
    scdp   : BEGIN pwait; OUT(data,CHR(cdp)) END;
    scsp   : BEGIN pwait; OUT(data,CHR(csp)) END;
    scwp   : BEGIN pwait; OUT(data,CHR(cwp)) END;
    sfcol  : BEGIN pwait; OUT(data,CHR(fcol)) END;
    sfpatr : BEGIN pwait; sendword(width,height);
                   OUT(data,CHR(p));
                   sendword(X,Y) END;
    sfpats : BEGIN pwait; OUT(data,CHR(p));
                   OUT(data,CHR(n)) END;
    spat   : BEGIN pwait; OUT(data,CHR(pat)) END;
    spcol  : BEGIN pwait; OUT(data,CHR(pcol)) END;
    srsel  : BEGIN pwait; OUT(data,CHR(rsel)) END;
    sstyle : BEGIN pwait; OUT(data,CHR(style)) END;
    stcol  : BEGIN pwait; OUT(data,CHR(tcol)) END;
    swprot : BEGIN pwait; OUT(data,CHR(wprot)) END
  END;
  {Now get return values if any.}
  CASE com OF
    allocp : BEGIN pwait; p := ORD(INP(data)) END;
    ibcol  : BEGIN pwait; bcol := ORD(INP(data)) END;
    iccol  : BEGIN pwait; ccol := ORD(INP(data)) END;
    icp    : BEGIN pwait; X := ORD(INP(data))+256*ORD(INP(data));
                   Y := ORD(INP(data))+256*ORD(INP(data)) END;
    icdp   : BEGIN pwait; cdp := ORD(INP(data)) END;
    icsp   : BEGIN pwait; csp := ORD(INP(data)) END;
    icwp   : BEGIN pwait; cwp := ORD(INP(data)) END;
    ifcol  : BEGIN pwait; fcol := ORD(INP(data)) END;
    ipat   : BEGIN pwait; pat := ORD(INP(data)) END;
    ipcol  : BEGIN pwait; pcol := ORD(INP(data)) END;
    irsel  : BEGIN pwait; rsel := ORD(INP(data)) END;
    istat  : BEGIN pwait; stat := ORD(INP(data)) END;
    istyle : BEGIN pwait; style := ORD(INP(data)) END;
    itcol  : BEGIN pwait; tcol := ORD(INP(data)) END;
    iwprot : BEGIN pwait; wprot := ORD(INP(data)) END;
    rimage : BEGIN pwait;
                   FOR i := 1 TO height DO
                   FOR j := 1 TO width DO
                   pixels[i,j] := ORD(INP(data)) END;
    rimagc : BEGIN pwait; notyet END;
    rpix   : BEGIN pwait; c := ORD(INP(data)) END;
    rpixr  : BEGIN pwait; c := ORD(INP(data)) END;
    rpixrs : BEGIN pwait; c := ORD(INP(data)) END;
    rsym   : BEGIN pwait;
                   FOR i := 1 TO height DO
                   FOR j := 1 TO width DO
                   pixels[i,j] := ORD(INP(data)) END;
    rsymc  : BEGIN pwait; notyet END
  END
END;
BEGIN {MAIN PROGRAM}
{You have to write this part for yourself, of
course, but it should be a lot easier now!}
END.
```

CP/M File Transfer Program Via The IEEE488 Bus.

By S. Wood

Below is a simple program written in BASIC-80 which, if present on two machines fitted with the EV Computing IEEE488 interface, will allow data files to be transfered between them at a reasonable rate. The speed loss is due to the amount of data conversion done in BASIC, but it is still faster than serial interface methods previously used. Shortly a machine code version with more features will be made available which will be able to transfer files very fast indeed (approaching 200 K bytes a second). The speed limitation then will be in the disk drives and not in the software or interface.

```
10 REM 488IO, Written by S.Wood. March 1984
20 WIDTH 255
30 B$=CHR$(2)
40 REM SENDER IS ADDRESS 1 & SYSTEM CONTROLLER
50 REM RECEIVER IS ADDRESS 2
60 INPUT "S)END OR R)ECEIVE";A$
70 IF A$="R" OR A$="r" THEN 390
80 IF A$="s" OR A$="S" THEN 110
90 IF A$<>"" THEN PRINT CHR$(7);:GOTO 60
100 END
```

```
110 REM SEND SECTION
120 PRINT B$;"pon,1/s":REM Initialise Sender as system controller
130 INPUT "File name";FS$
140 PRINT B$;"wrt,2 ";FS$:REM Send filename to be loaded across
150 OPEN "R",1,FS$,128
160 R=1:FIELD 1,128 AS X$
170 GET 1,1:REM Force dummy file read otherwise EOF function does not work
180 WHILE NOT EOF(1):REM Main sending loop
190 GET 1,R
200 R=R+1
210 S1$="":S2$="":REM Generate two ASCII-Hex strings 128 and 129 bytes long
220 FOR A=1 TO 64:REM the second string has an EOF marker attached.
230 S$=HEX$(ASC(MID$(X$,A,1))):IF LEN(S$)=1 THEN S$="0"+S$
240 S1$=S1$+S$
250 NEXT A
260 FOR A=65 TO 128
270 S$=HEX$(ASC(MID$(X$,A,1))):IF LEN(S$)=1 THEN S$="0"+S$
280 S2$=S2$+S$
290 NEXT A
300 IF EOF(1) THEN S$="A" ELSE S$="M"
310 S2$=S2$+S$
320 PRINT B$;"wrt,2 ";S1$
330 PRINT B$;"wrt,2 ";S2$
340 WEND
350 PRINT"Complete."
360 INPUT "Any more files to send";A$
370 IF A$="y" OR A$="Y" THEN RUN
380 END
390 REM RECEIVE SECTION
400 PRINT B$;"pon,2":REM Power-on the receiver as address 2
410 PRINT B$;:INPUT "red,1";FR$
420 PRINT"File ";FR$;" is being received."
430 R=1
440 OPEN "R",1,FR$,128
450 FIELD 1,128 AS X$
460 REM RE-ENTER HERE
470 TEMP$=""
480 PRINT B$;:INPUT "red,1";S1$
490 PRINT B$;:INPUT "red,1";S2$ :TEMP$=""
500 IF RIGHT$(S2$,1)="A" AND LEN(S2$)=129 THEN 610:REM END OF FILE
510 FOR A=1 TO 128 STEP 2
520 TEMP$=TEMP$+CHR$(VAL("&H"+MID$(S1$,A,2)))
530 NEXT A
540 FOR A=1 TO 128 STEP 2
550 TEMP$=TEMP$+CHR$(VAL("&H"+MID$(S2$,A,2)))
560 NEXT A
570 LSET X$=TEMP$
580 PUT 1,R:R=R+1
590 IF RIGHT$(S2$,1)="M" THEN 460:REM Not EOF yet
600 IF RIGHT$(S2$,1)<>"A" THEN PRINT"error":STOP
610 CLOSE 1
620 PRINT B$;"rdy":REM Read last line-feed from Bus
630 RUN
```

Dave Hunt's Bits

So 80-BUS is on the move again, and has caught me by totally by surprise as I've nothing ready to print. Do I hear cries of shame!!! [Ed. - No.]

They say that behind every great man is a woman calling the shots, and in the case of our editor, this seems to be the case. Now he's too modest and/or shy to make any comment, but if I say that shortly before the Christmas issue was due to be put to bed, a rather nice young lady entered his life .... suffice to say that the demands on his spare time (usually devoted to magazine preparation) was devoted, or, should I say, diverted elsewhere. Now, this being the case, if you assume that the reappearance of the mag. is indicative of a departure in his life, you would be wrong, perhaps he's feeling guilty having left his readership magazine-less for so long, or something; whatever, it has created a great spurt of energy, and not only is this issue in the last stages of preparation as I write, but the next issue as well.

The radio bits first....

Firstly AMTOR, I am now in possession of a number of reports on the development of the commercial system upon which AMTOR is based and also the various specifications which comprise the working details of AMTOR. My grateful thanks to those readers (several anonymous) who sent me the details. I confess that I'm somewhat intrigued by the two copies addressed directly to my home, as at the time my address hadn't been published anywhere (now QTHR in the 1984 callbook). Sadly, I'm not sure that I will be able to make use of the information (although all information is ultimately useful at sometime). As you may have guessed I have been investigating 'PACKET RADIO', or at least the bastardized form that the UK amateur radio regulations allow.

Literally a couple of days ago I was presented with the PACKET RADIO program by G8WJL and G6GIX, for the BBC computer. I swiped a BEEB and had a go, it certainly works well. I particularly liked the fact that no additional hardware was required, just a lead which plugs from the BEEB tape I/O socket into the mic, ptt and extension speaker sockets of the rig.

Despite the ease of setting up, I feel the program suffers from a number of minor deficiencies, niggles really. If the program is active, then it will read any packet information that it sees, so if there are two or three QSO's going on on the same frequency (don't forget the idea of packet is to allow just this), then the program writes the packets to the screen. This is fine if you are 'earwigging' and just want to know who's around. The problem is that if you start a QSO with a station, then (as far as I can see) this reporting of other QSO's continues, making it difficult to see who is talking to who. Worse, because of this reporting, it suggests that multiway QSO's are possible, fine until you try it. As a data anti-collision protocol is used with random timing between packets, there is no knowing which station is going to send first, so unless 'to' -> 'from' callsigns are included in each packet text then multiway QSO's becomes extremely confusing. A simple software toggle to turn the 'blurb' off once contact is established would be a very good idea.

A further off-shoot of the 'blurb' reporting is the ability to 'sort of' send a CQ, something which packet does not normally allow. If you program the originating callsign as your own and the destination callsign as CQCQ rather

than a legit callsign, followed by the message 'DE G6MFR', then any station in the 'earwigging' mode will see:

G6MFR: CQCQ DE G6MFR

The receiving station, say G2XXX, can then open a communications channel to you simply by programming the destination callsign as the callsign just received. So his channel becomes G2XXX -> G6MFR. G2XXX can then send a message (or even no message), say KN DE G2XXX, where upon you will see:

G2XXX: KN DE G2XXX

What's the problem? Well, as you were sending to a mythical station called CQCQ you can't go back to G2XXX until the packet program gives up trying to contact the station CQCQ (16 tries over the space of a couple of minutes), or without first having escaped from the program and then reRUNning it. This takes time and clears the screen so if you've got a bad memory like me, then you've forgotten who was calling in the first place.

The program needs to know when no-one else is transmitting to allow the anti-collision logic to work. This is done by manually opening the squelch and the consequent white noise is detected by the tape I/O (probably as random numbers) indicating that the program is free to transmit. When a strong blank carrier is up, or data is being received, the program inhibits transmit and goes to sleep for a random period before trying again. Now this has snags. A noisy incoming signal which doesn't quite trigger the tape I/O could be ignored or interpreted as a free channel allowing transmit (this sort of error occurs on signals worse than about 4 by 5). Secondly, white noise must be present to indicate the channel is free. This precludes any phase lock signal shaping circuits which could be used to reconstitute a noisy signal, thereby much improving the sensitivity of the system. Now the rig squelch circuit is much better at differentiating between signal and no signal than the computer, so a logical development would be to return the squelch line to the computer to detect a channel free condition. For sideband use, a phase lock shaper could be used and the out-of-lock line from this may be used to indicate a free channel.

... and BBC BASIC

It was to investigate my niggles and with a mind to convert the program to my machine that I started to poke around inside the program. Now I haven't paid much attention to the BEEB machine in the past, prefering my machine which I consider superior and, at least, I understand. Some people tell me the BEEB is the most fantastic machine since the invention of the rocket propelled roller skate, whilst others tell me that all BEEBs should be collected, along with their owners, put in a large box and dumped at the bottom of the Marianas Trench as an insidious danger to mankind. I know of one gentleman who made a considerable loss by trading in an almost new Gemini MultiBoard machine for a BEEB, whilst I know of another who can't wait to trade his BEEB for a Galaxy when he can rake up the necessary. So overall, on a statistical sample of two, 50% of BEEB owners are loonies whilst the other 50% have seen the light and should be helped as much as possible. Certainly the BEEB seems to me to be overpriced and has gained a sort of cult acceptance which it does not deserve, but then the same could be said for some of the other machines around, so ... there you go!!

Anyway on to BEEB 'BASIC', the quotes are deliberate, as whatever it is, it bears little relation to the original Dartmouth College Basic. In fact its closer relations would seem to be some sort of cross between Pascal and C with BASIC syntax. Perhaps a better name for it might be BEEB C-BasPas, anything, but not BASIC. As a language, it's very powerful, but encourages 'opaque programming' under the guise of 'structured programming'. In other words if the programmer cares to do the job properly with lashings of comment and everything laid out in a nice orderly fashion, then anything written in C-BasPas should be beautifully clear; on the other hand, if the programmer wanted make his program difficult to read (or is lazy like most of us), all he has to do is remove all the comment, jumble the order of the PROCedures and function calls, and the program becomes as clear as mud. The latter is more likely to be the case as the available memory in the BEEB is quite small (having subtracted the screen memory and sundry workspaces (a lot of which don't seem to do much)), as it's quite difficult to squeeze a program of any size into the available memory space without resorting to 'dirty' short cuts.

The BEEB is also provided with a large number of indirectly addressed operating system calls (driven from a number table, something like NAS-SYS) which the user is encouraged to use from the C-BasPas. The OS calls are reasonably documented in the manual, but rely on an understanding of the hardware and devices used, information which is sadly not provided. (I understand the Advanced User's manual guide goes into detail, but as the BEEB machine I have is on borrow, I don't feel inclined to buy this just to see what's happening).

The program uses the BEEB CUTS tape I/O but seems to address it directly, as the data sent appears to be in the form of a bit stream (as it should be) rather than ASCII characters with start and stop bits using the UART as I had hoped. Shades of nasty undocumented things going on in the ULA. All this will prove an interesting exercise on a Gemini, as I guess a couple of bits from a port are going to have to be hard wired to the tape I/O and a software UART written, either that or I'm going to have to design an SIO piggyback board.

So my original plan of quickly rewriting the PACKET program into something more generalized has had to go by the board. It's not so much an exercise in rewriting, but having to learn C-BasPas and the detail innards of the BEEB operating system and hardware, a time wasting exercise that I'm not too keen on. Oh well, I will carry on (unless someone else wants to have a go). Instead of something I judged would take a few evenings to do, it looks like it's going to take weeks. Still when (if) I've done it we'll publish it here so that Gemini's and Nascom's can take to the air in the PACKET mode.

As to my opinion of the BEEB after a couple of days acquaintance ... well let's say I wouldn't give more than fifty quid for one if someone was rash enough to pass one my way!! Torch have the right idea, demote the BEEB to a terminal for use on a CP/M system, it's about all it's good for. Pity Torch had to do their own thing with a non-standard CP/M system. Perhaps I might have a go at doing the same with a MultiBoard system, because overpriced as the BEEB is, it might just be cheaper than buying an SVC and keyboard; it could then replace these two, with high res. colour thrown in. Now all this brands me as a heretic with the BEEB cult. So I'd better keep quiet otherwise some dark night, someone might find me lying in a dark alley with a BEEB in my back.

BIOS 3 ... Where is it?

Some time ago (in fact about this time last year, early June) I wrote a piece called 'SYS is dead, long live ...', where I stated that due to alleged piracy of the SYS disk drive source for commercial purposes Gemini were likely to withdraw their permission for the publication of their disk drivers in the SYS source. At about the same time Gemini started to charge very high prices for their source code to bona fide users. The point of my piece was that Gemini were pricing their source code at prices which only pirates, who could take commercial advantage of their source code, could afford. This accusation on my part quite understandably upset Gemini, who it was aimed at, to the point that they have threatened to produce a product called BIOS 3 which will allow the Gemini user to update his BIOS to the latest spec. and include all Gemini permutations of drive/controller/video for himself. As the majority of SYS users were using SYS for just this purpose, the need for SYS would disappear. So far BIOS 3 has not appeared, so a gentle nudge is required. How about it Gemini!!!

An apology

Much to my surprise, my piece 'SYS is dead, long live ...' provoked a much more vehement reaction from an unexpected quarter. Due to the tendency of 80-BUS magazine to appear in two's, my published comments in Vol.2 Iss.4 could be linked with an unattributed editorial piece (which I hasten to add I did not write) in Vol.2 Iss.5, the following issue, which appeared about the same time. MAP80 Systems of Chertsey have taken very strong objection to these two pieces and claim that these have damaged their business. If MAP80 Systems think that my piece was directed at them and in consequence, damaged their business, then I can only apologise and unreservedly withdraw any such imputation. My piece made no allusions direct or indirect to any specific concern.

As far as the second piece, the editorial, is concerned, this was merely a statement of fact, nothing more nothing less. The details of the litigation were sub-judice at the time and as far as I know, still are. The fact that MAP80 Systems and Gemini are in litigation has no bearing on my piece, as this litigation started sometime after I wrote the piece, which after all, as stated above, was intended as a 'side-swipe' at Gemini.

The SVC Board

The new Gemini GM832 Super Video Controller Board has at long last escaped from Amersham. I'm not sure whether this was a mistake or not, as up to a week or two ago, these cards were rare-er than rocking horse manure (very good for plastic rose trees I'm told). Anyway one of these cards fell into my clutches for a couple of hours before being whipped away by the customer who had paid for it months before.

You don't get much time to do a full evaluation in two hours, but first impressions were good. There was none of the video patterning characteristic of the GM812 when in inverse video. The most immediate impression is speed. Using the TYPE command under CP/M revealed a staggering increase in speed with text hurtling up the screen at a phenomenal rate. A lot more than the 50% increase directly attributable to the use of a 6MHz Z80B. The screen/workspace RAM contention logic has been removed from software and placed in hardware, and this is one other factor increasing the speed. The nett result looks more like 150% faster, although I didn't set to with stop watch to time it or to read the documentation to see if any actual figure is quoted. The only problem

with this improved display speed is that fingers 2 and 4 on my left hand had to increase their reaction time by a proportional amount, as there was barely time to bash ^S before the thing you wanted to see disappeared off the screen. Perhaps a ZZzzz... command could be built into CCPZ to slow it up when not actually executing a program.

The next feature investigated (for its novelty) was the clock display. Once enabled, a changing clock display is written at a user predetermined cursor position on the screen. The default cursor position is the extreme top right hand corner of the screen, and this suits most software I had a chance to try it with. The internal clock generator is interrupt driven from the frame sync., which is divided by 50 to update the clock counter and redisplay the clock once a second. As the system clock is crystal controlled, it is accurate enough for most purposes although I think reprogramming the 6845 video controller could upset the clock accuracy. A very simple routine was written which read the time from the GM822 RTC and loaded the clock on the SVC. At the end of the two hour period there was a discepancy of 7 seconds between the RTC and the SVC clock.

A number of bits of software which were known to be to greater or lesser extent video card dependant were tried to see if the compatibility between the GM832 and the now discontinued GM812 were as claimed, and all were found to work as before. DISKPEN/GEMPEN required adjustment to the cursor speed patch byte, because the cursor flash rate went berserk, but in all other respects except the 48 wide mode (the new SVCs second screen mode is now 40 wide) DISKPEN/GEMPEN worked well. The increased speed of the card showed to advantage with WORDSTAR, screen rewrites being accomplished with very acceptable speed. My old CHARGEN program for reprogramming the character sets worked, but of course needs rewriting in the light of the SVC as there are now two programmable character sets (total 256 characters) instead of one.

The SVC has a number of new features including 256 by 256 pixel graphics with built in line and circle drawing software, selectable attributes for flashing characters with low, half and inverse intensities, inputs for serial keyboards, and many other things. As I said, I only laid hands on one for a very short period, so it is very difficult to be objective. I didn't find any nasty quirks and the board behaved impeccably. Perhaps now Gemini will loan me one to play with, and I'll be able to find out what it can really do. The only thing that worries me though is the very high degree of sophistication of this card, commensurate with its elevated price. In my experience, 99% of users were unaware of the potential of the old GM812 card, so is all this cleverness really necessary?

The Climax colour card

The Climax colour card is to be re-introduced. Gemini have acquired the rights to manufacture the card from Climax, and deliveries are to commence very shortly. This card was very popular with those owning it, and some very clever displays have been seen. But as the card has been unavailable for the best part of a year, interest in it has waned. With its welcome reappearance perhaps some new and clever software will start to be written. Certainly as the video format of the Climax card is the same as the high-res mode of the new SVC, there will be demand for a piece of software which will act as an interface between the two. Here is an opportunity for CC-SOFT, who have written some rather nice graphics software for Gemini gear in the past to come up with something. It would be nice to see two interfaces with common inputs

which would then either drive the SVC or the Climax from the same graphics programs.

With the advent of either black and white high-res or colour graphics perhaps some enterprising software writers will have a go at writing some arcade type games for the Gemini and Nascom hybrid machines. These would have to written for fun as there would be little money in it, but it is surprising how often we get asked about games for the Gemini. There are a number of the more intellectual type, Adventure, Chess, Planetfall, etc., but the arcade types tend to get overlooked. I know I personally have no patience with arcade games, both lacking in the necessary co-ordination to play them and also in the necessary patience to practice. I also realise for the cost of a Climax card (now Gemini GM837) I could buy a Spectrum and a whole bag full of Space Invaders tapes. But some people seem to want them and where there is a need someone will usually try to fill it.

Qwikdraw

Whilst still on the subject of colour graphics, the expensive Pluto card (now only available in the full 8MHz, extended monitor form) has gained a very versatile graphics package shortly to be made available by Gemini. Called Qwikdraw, and written primarily for use on the Gemini networks installed in the Manpower Services YTS training scheme, Qwikdraw is an easy to use graphics package with some very novel features. Input is either from the keyboard (using the cursor keys with selectable step rates) or, ultimately, from a bit pad. Drawing of graphical displays is quite easy with automatic circle and smoothed curve creation. Block and complex shape fills and colour floods are also catered for. An optional colour mixing package will allow up to 32 colours to be displayed, and a 'picture compiler' which can convert the stored picture format into a .COM file for immediate execution. Limited animation is also possible. Display is normally to a high resolution colour monitor or to a dot matrix printer or to a multicolour plotter. Very extensive 'help' facilities are provided, which are an education in themselves with animated graphics demonstrating the points queried. One interesting use demonstrated was the preparation of the cels used for overhead projectors in an educational environment. The cels being drawn on the plotter using oil based pens. Whilst not as versatile as the Nascom Lotti, it is extremely quick and easy to use with a powerful editor, it should find lots of applications outside the YTS scheme particularly in the education field.

Printer. RS232 interface. 180 cps. Bidirectional. Buffered. True Descenders. #80. Crawthorne (0344) 776894.

Nascom 2 with Nas-Sys 3, Naspen, 48K RAM and Castle interface - #300; additional 48K RAM B - #75; self-contained keyboard with lead - #35; graphics chip - #10; PSU - #25; Bricomp Real Time Clock/Calendar - #17; Nasbus EPROM/ROM card with Naspen, Debug, ZEAP - #45. 0532 740921

Colour card (Holmes/R&EW) - 16 colour + sprites. Also has 2 sound generators, 2 clock chips, 8 port A-D, CMOS RAM, CTC. Fully built, working with hardware and software. All ic's (socketed) except 1 off AY-3-8910. All xtals and backup battery. Also interface card to convert colour diff. signals to RGB, offers around #100. Tel C. Bowden 0209 860480 for details.

NASCOM ROM BASIC DIS–ASSEMBLED

PART 5

BY CARL LLOYD–PARKER


```
F012 E5        ARLDSV: PUSH    HL              ; Save code string address
F013 F5                PUSH    AF              ; A = 00 , Flags set = Z,N
F014 2AD810             LD      HL,(VAREND)     ; Start of arrays
F017 3E                 DEFB    (LD A,n)        ; Skip "ADD HL,DE"
F018 19        FNDARY: ADD     HL,DE           ; Move to next array start
F019 EB                 EX      DE,HL
F01A 2ADA10             LD      HL,(ARREND)     ; End of arrays
F01D EB                 EX      DE,HL           ; Current array pointer
F01E CD8AE6             CALL    CPDEHL          ; End of arrays found?
F021 CA4AF0             JP      Z,CREARY        ; Yes - Create array
F024 7E                 LD      A,(HL)          ; Get second byte of name
F025 B9                 CP      C               ; Compare with name given
F026 23                 INC     HL              ; Move on
F027 C22CF0             JP      NZ,NXTARY       ; Different - Find next array
F02A 7E                 LD      A,(HL)          ; Get first byte of name
F02B B8                 CP      B               ; Compare with name given
F02C 23        NXTARY: INC     HL              ; Move on
F02D 5E                 LD      E,(HL)          ; Get LSB of next array address
F02E 23                 INC     HL
F02F 56                 LD      D,(HL)          ; Get MSB of next array address
F030 23                 INC     HL
F031 C218F0             JP      NZ,FNDARY       ; Not found - Keep looking
F034 3AAC10             LD      A,(LCRFLG)      ; Found - Locate or Create it?
F037 B7                 OR      A
F038 C2B6E3             JP      NZ,DDERR        ; Create - ?DD Error
F03B F1                 POP     AF              ; Locate - Get number of dim'ns
F03C 44                 LD      B,H             ; BC Points to array dim'ns
F03D 4D                 LD      C,L
F03E CA54F7             JP      Z,POPHRT        ; Jump if array load/save
F041 96                 SUB     (HL)            ; Same number of dimensions?
F042 CA8AF0             JP      Z,FINDEL        ; Yes - Find element
F045 1E10      BSERR:  LD      E,BS            ; ?BS Error
F047 C3C1E3             JP      ERROR           ; Output error
```

```
F04A 110400     CREARY: LD      DE,4            ; 4 Bytes per entry
F04D F1                 POP     AF              ; Array to save or 0 dim'ns?
F04E CAA0E9             JP      Z,FCERR         ; Yes - ?FC Error
F051 71                 LD      (HL),C          ; Save second byte of name
F052 23                 INC     HL
F053 70                 LD      (HL),B          ; Save first byte of name
F054 23                 INC     HL
F055 4F                 LD      C,A             ; Number of dimensions to C
F056 CD8AE3             CALL    CHKSTK          ; Check if enough memory
F059 23                 INC     HL              ; Point to number of dimensions
F05A 23                 INC     HL
F05B 22C510             LD      (CUROPR),HL     ; Save address of pointer
F05E 71                 LD      (HL),C          ; Set number of dimensions
F05F 23                 INC     HL
F060 3AAC10             LD      A,(LCRFLG)      ; Locate of Create?
F063 17                 RLA                     ; Carry set = Create
F064 79                 LD      A,C             ; Get number of dimensions
F065 010B00     CRARLP: LD      BC,10+1         ; Default dimension size 10
F068 D26DF0             JP      NC,DEFSIZ       ; Locate - Set default size
F06B C1                 POP     BC              ; Get specified dimension size
F06C 03                 INC     BC              ; Include zero element
F06D 71         DEFSIZ: LD      (HL),C          ; Save LSB of dimension size
F06E 23                 INC     HL
F06F 70                 LD      (HL),B          ; Save MSB of dimension size
F070 23                 INC     HL
F071 F5                 PUSH    AF              ; Save num' of dim'ns an status
F072 E5                 PUSH    HL              ; Save address of dim'n size
F073 CDFFF8             CALL    MLDEBC          ; Multiply DE by BC to find
F076 EB                 EX      DE,HL           ; amount of mem needed (to DE)
F077 E1                 POP     HL              ; Restore address of dimension
F078 F1                 POP     AF              ; Restore number of dimensions
F079 3D                 DEC     A               ; Count them
F07A C265F0             JP      NZ,CRARLP       ; Do next dimension if more
F07D F5                 PUSH    AF              ; Save locate/create flag
F07E 42                 LD      B,D             ; MSB of memory needed
F07F 4B                 LD      C,E             ; LSB of memory needed
F080 EB                 EX      DE,HL
F081 19                 ADD     HL,DE           ; Add bytes to array start
F082 DAA2E3             JP      C,OMERR         ; Too big - Error
F085 CD93E3             CALL    ENFMEM          ; See if enough memory
F088 22DA10             LD      (ARREND),HL     ; Save new end of array
```



```
F08B 2B         ZERARY: DEC     HL              ; Back through array data
F08C 3600               LD      (HL),0          ; Set array element to zero
F08E CD8AE6             CALL    CPDEHL          ; All elements zeroed?
F091 C28BF0             JP      NZ,ZERARY       ; No - Keep on going
F094 03                 INC     BC              ; Number of bytes + 1
F095 57                 LD      D,A             ; A=0
F096 2AC510             LD      HL,(CUROPR)     ; Get address of array
F099 5E                 LD      E,(HL)          ; Number of dimensions
F09A EB                 EX      DE,HL           ; To HL
F09B 29                 ADD     HL,HL           ; Two bytes per dimension size
F09C 09                 ADD     HL,BC           ; Add number of bytes
F09D EB                 EX      DE,HL           ; Bytes needed to DE
F09E 2B                 DEC     HL
F09F 2B                 DEC     HL
F0A0 73                 LD      (HL),E          ; Save LSB of bytes needed
F0A1 23                 INC     HL
F0A2 72                 LD      (HL),D          ; Save MSB of bytes needed
F0A3 23                 INC     HL
F0A4 F1                 POP     AF              ; Locate / Create?
F0A5 DACCF0             JP      C,ENDDIM        ; A is 0 , End if create
F0A8 47         FINDEL: LD      B,A             ; Find array element
F0A9 4F                 LD      C,A
F0AA 7E                 LD      A,(HL)          ; Number of dimensions
F0AB 23                 INC     HL
F0AC 16                 DEFB    (LD D,n)        ; Skip "POP HL"
F0AD E1         FNDELP: POP     HL              ; Address of next dim' size
F0AE 5E                 LD      E,(HL)          ; Get LSB of dim'n size
F0AF 23                 INC     HL
F0B0 56                 LD      D,(HL)          ; Get MSB of dim'n size
F0B1 23                 INC     HL
F0B2 E3                 EX      (SP),HL         ; Save address - Get index
F0B3 F5                 PUSH    AF              ; Save number of dim'ns
F0B4 CD8AE6             CALL    CPDEHL          ; Dimension too large?
F0B7 D245F0             JP      NC,BSERR        ; Yes - ?BS Error
F0BA E5                 PUSH    HL              ; Save index
F0BB CDFFF8             CALL    MLDEBC          ; Multiply previous by size
F0BE D1                 POP     DE              ; Index supplied to DE
F0BF 19                 ADD     HL,DE           ; Add index to pointer
F0C0 F1                 POP     AF              ; Number of dimensions
F0C1 3D                 DEC     A               ; Count them
F0C2 44                 LD      B,H             ; MSB of pointer
F0C3 4D                 LD      C,L             ; LSB of pointer
F0C4 C2ADF0             JP      NZ,FNDELP       ; More - Keep going
F0C7 29                 ADD     HL,HL           ; 4 Bytes per element
F0C8 29                 ADD     HL,HL
F0C9 C1                 POP     BC              ; Start of array
F0CA 09                 ADD     HL,BC           ; Point to element
F0CB EB                 EX      DE,HL           ; Address of element to DE
F0CC 2AD010     ENDDIM: LD      HL,(NXTOPR)     ; Get code string address
F0CF C9                 RET
```

```
FOD0 2ADA10     FRE:    LD      HL,(ARREND)     ; Start of free memory
FOD3 EB                 EX      DE,HL           ; To DE
FOD4 210000             LD      HL,0            ; End of free memory
FOD7 39                 ADD     HL,SP           ; Current stack value
FOD8 3AAD10             LD      A,(TYPE)        ; Dummy argument type
FODB B7                 OR      A
FODC CAECFO             JP      Z,FRENUM        ; Numeric - Free variable space
FODF CD53F3             CALL    GSTRCU          ; Current string to pool
FOE2 CD53F2             CALL    GARBGE          ; Garbage collection
FOE5 2A5A10             LD      HL,(STRSPC)     ; Bottom of string space in use
FOE8 EB                 EX      DE,HL           ; To DE
FOE9 2AC310             LD      HL,(STRBOT)     ; Bottom of string space
FOEC 7D         FRENUM: LD      A,L             ; Get LSB of end
FOED 93                 SUB     E               ; Subtract LSB of beginning
FOEE 4F                 LD      C,A             ; Save difference if C
FOEF 7C                 LD      A,H             ; Get MSB of end
FOF0 9A                 SBC     A,D             ; Subtract MSB of beginning
FOF1 41         ACPASS: LD      B,C             ; Return integer AC
FOF2 50         ABPASS: LD      D,B             ; Return integer AB
FOF3 1E00               LD      E,0
FOF5 21AD10             LD      HL,TYPE         ; Point to type
FOF8 73                 LD      (HL),E          ; Set type to numeric
FOF9 0690               LD      B,80H+16        ; 16 bit integer
FOFB C32AF8             JP      RETINT          ; Return the integr

FOFE 3AAB10     POS:    LD      A,(CURPOS)      ; Get cursor position
F101 47         PASSA:  LD      B,A             ; Put A into AB
F102 AF                 XOR     A               ; Zero A
F103 C3F2F0             JP      ABPASS          ; Return integer AB

F106 CD89F1     DEF:    CALL    CHEKFN          ; Get "FN" and name
F109 CD7BF1             CALL    IDTEST          ; Test for illegal direct
F10C 0170EA             LD      BC,DATA         ; To get next statement
F10F C5                 PUSH    BC              ; Save address for REturn
F110 D5                 PUSH    DE              ; Save address of function ptr
F111 CD90E6             CALL    CHKSYN          ; Make sure "(" follows
F114 28                 DEFB    "("
F115 CD2DEF             CALL    GETVAR          ; Get argument variable name
F118 E5                 PUSH    HL              ; Save code string address
F119 EB                 EX      DE,HL           ; Argument address to HL
F11A 2B                 DEC     HL
F11B 56                 LD      D,(HL)          ; Get first byte of arg name
F11C 2B                 DEC     HL
F11D 5E                 LD      E,(HL)          ; Get second byte of arg name
F11E E1                 POP     HL              ; Restore code string address
F11F CD44ED             CALL    TSTNUM          ; Make sure numeric argument
F122 CD90E6             CALL    CHKSYN          ; Make sure ")" follows
F125 29                 DEFB    ")"
F126 CD90E6             CALL    CHKSYN          ; Make sure "=" follows
F129 B4                 DEFB    ZEQUAL          ; "=" token
F12A 44                 LD      B,H             ; Code string address to BC
F12B 4D                 LD      C,L
F12C E3                 EX      (SP),HL         ; Save code str , Get FN ptr
F12D 71                 LD      (HL),C          ; Save LSB of FN code string
F12E 23                 INC     HL
F12F 70                 LD      (HL),B          ; Save MSB of FN code string
```



```
F133 CD89F1     DOFN:   CALL    CHEKFN          ; Make sure FN follows
F136 D5                 PUSH    DE              ; Save function pointer address
F137 CD09EE             CALL    EVLPAR          ; Evaluate expression in "()"
F13A CD44ED             CALL    TSTNUM          ; Make sure numeric result
F13D E3                 EX      (SP),HL         ; Save code str , Get FN ptr
F13E 5E                 LD      E,(HL)          ; Get LSB of FN code string
F13F 23                 INC     HL
F140 56                 LD      D,(HL)          ; Get MSB of FN code string
F141 23                 INC     HL
F142 7A                 LD      A,D             ; And function DEFined?
F143 B3                 OR      E
F144 CAB9E3             JP      Z,UFERR         ; No - ?UF Error
F147 7E                 LD      A,(HL)          ; Get LSB of argument address
F148 23                 INC     HL
F149 66                 LD      H,(HL)          ; Get MSB of argument address
F14A 6F                 LD      L,A             ; HL = Arg variable address
F14B E5                 PUSH    HL              ; Save it
F14C 2ADE10             LD      HL,(FNRGNM)     ; Get old argument name
F14F E3                 EX      (SP),HL         ; Save old , Get new
F150 22DE10             LD      (FNRGNM),HL     ; Set new argument name
F153 2AE210             LD      HL,(FNARG+2)    ; Get LSB,NLSB of old arg value
F156 E5                 PUSH    HL              ; Save it
F157 2AE010             LD      HL,(FNARG)      ; Get MSB,EXP of old arg value
F15A E5                 PUSH    HL              ; Save it
F15B 21E010             LD      HL,FNARG        ; HL = Value of argument
F15E D5                 PUSH    DE              ; Save FN code string address
F15F CD6BF8             CALL    FPTHL           ; Move FPREG to argument
F162 E1                 POP     HL              ; Get FN code string address
F163 CD41ED             CALL    GETNUM          ; Get value from function
F166 2B                 DEC     HL              ; DEC 'cos GETCHR INCs
F167 CD36E8             CALL    GETCHR          ; Get next character
F16A C2ADE3             JP      NZ,SNERR        ; Bad character in FN - Error
F16D E1                 POP     HL              ; Get MSB,EXP of old arg
F16E 22E010             LD      (FNARG),HL      ; Restore it
F171 E1                 POP     HL              ; Get LSB,NLSB of old arg
F172 22E210             LD      (FNARG+2),HL    ; Restore it
F175 E1                 POP     HL              ; Get name of old arg
F176 22DE10             LD      (FNRGNM),HL     ; Restore it
F179 E1                 POP     HL              ; Restore code string address
F17A C9                 RET

F17B E5         IDTEST: PUSH    HL              ; Save code string address
F17C 2A5C10             LD      HL,(LINEAT)     ; Get current line number
F17F 23                 INC     HL              ; -1 means direct statement
F180 7C                 LD      A,H
F181 B5                 OR      L
F182 E1                 POP     HL              ; Restore code string address
F183 C0                 RET     NZ              ; Return if in program
F184 1E16               LD      E,ID            ; ?ID Error
F186 C3C1E3             JP      ERROR
```

```
F189 CD90E6     CHEKFN: CALL    CHKSYN          ; Make sure FN follows
F18C A7                 DEFB    ZFN             ; "FN" token
F18D 3E80               LD      A,80H
F18F 32CB10             LD      (FORFLG),A      ; Flag FN name to find
F192 B6                 OR      (HL)            ; FN name has bit 7 set
F193 47                 LD      B,A             ; in first byte of name
F194 CD32EF             CALL    GTFNAM          ; Get FN name
F197 C344ED             JP      TSTNUM          ; Make sure numeric function

F19A CD44ED     STR:    CALL    TSTNUM          ; Make sure it's a number
F19D CDB8F9             CALL    NUMASC          ; Turn number into text
F1A0 CDCEF1             CALL    CRTST           ; Create string entry for it
F1A3 CD53F3             CALL    GSTRCU          ; Current string to pool
F1A6 01AEF3             LD      BC,TOPOOL       ; Save in string pool
F1A9 C5                 PUSH    BC              ; Save address on stack

F1AA 7E         SAVSTR: LD      A,(HL)          ; Get string length
F1AB 23                 INC     HL
F1AC 23                 INC     HL
F1AD E5                 PUSH    HL              ; Save pointer to string
F1AE CD29F2             CALL    TESTR           ; See if enough string space
F1B1 E1                 POP     HL              ; Restore pointer to string
F1B2 4E                 LD      C,(HL)          ; Get LSB of address
F1B3 23                 INC     HL
F1B4 46                 LD      B,(HL)          ; Get MSB of address
F1B5 CDC2F1             CALL    CRTMST          ; Create string entry
F1B8 E5                 PUSH    HL              ; Save pointer to MSB of addr
F1B9 6F                 LD      L,A             ; Length of string
F1BA CD46F3             CALL    TOSTRA          ; Move to string area
F1BD D1                 POP     DE              ; Restore pointer to MSB
F1BE C9                 RET

F1BF CD29F2     MKTMST: CALL    TESTR           ; See if enough string space
F1C2 21BF10     CRTMST: LD      HL,TMPSTR       ; Temporary string
F1C5 E5                 PUSH    HL              ; Save it
F1C6 77                 LD      (HL),A          ; Save length of string
F1C7 23                 INC     HL
F1C8 23         SVSTAD: INC     HL
F1C9 73                 LD      (HL),E          ; Save LSB of address
F1CA 23                 INC     HL
F1CB 72                 LD      (HL),D          ; Save MSB of address
F1CC E1                 POP     HL              ; Restore pointer
F1CD C9                 RET
```



```
F1CE 2B         CRTST:  DEC     HL              ; DEC - INCed after
F1CF 0622       QTSTR:  LD      B,'"'           ; Terminating quote
F1D1 50                 LD      D,B             ; Quote to D
F1D2 E5         DTSTR:  PUSH    HL              ; Save start
F1D3 0EFF               LD      C,-1            ; Set counter to -1
F1D5 23         QTSTLP: INC     HL              ; Move on
F1D6 7E                 LD      A,(HL)          ; Get byte
F1D7 0C                 INC     C               ; Count bytes
F1D8 B7                 OR      A               ; End of line?
F1D9 CAE4F1             JP      Z,CRTSTE        ; Yes - Create string entry
F1DC BA                 CP      D               ; Terminator D found?
F1DD CAE4F1             JP      Z,CRTSTE        ; Yes - Create strig entry
F1E0 B8                 CP      B               ; Terminator B found?
F1E1 C2D5F1             JP      NZ,QTSTLP       ; No - Keep looking
F1E4 FE22       CRTSTE: CP      '"'             ; End with '"'?
F1E6 CC36E8             CALL    Z,GETCHR        ; Yes - Get next character
F1E9 E3                 EX      (SP),HL         ; Starting quote
F1EA 23                 INC     HL              ; First byte of string
F1EB EB                 EX      DE,HL           ; To DE
F1EC 79                 LD      A,C             ; Get length
F1ED CDC2F1             CALL    CRTMST          ; Create string entry
F1F0 11BF10     TSTOPL: LD      DE,TMPSTR       ; Temporary string
F1F3 2AB110             LD      HL,(TMSTPT)     ; Temporary string pool pointer
F1F6 22E410             LD      (FPREG),HL      ; Save address of string ptr
F1F9 3E01               LD      A,1
F1FB 32AD10             LD      (TYPE),A        ; Set type to string
F1FE CD6EF8             CALL    DETHL4          ; Move string to pool
F201 CD8AE6             CALL    CPDEHL          ; Out of string pool?
F204 22B110             LD      (TMSTPT),HL     ; Save new pointer
F207 E1                 POP     HL              ; Restore code string address
F208 7E                 LD      A,(HL)          ; Get next code byte
F209 C0                 RET     NZ              ; Return if pool OK
F20A 1E1E               LD      E,ST            ; ?ST Error
F20C C3C1E3             JP      ERROR           ; String pool overflow

F20F 23         PRNUMS: INC     HL              ; Skip leading space
F210 CDCEF1     PRS:    CALL    CRTST           ; Create string entry for it
F213 CD53F3     PRS1:   CALL    GSTRCU          ; Current string to pool
F216 CD62F8             CALL    LOADFP          ; Move string block to BCDE
F219 1C                 INC     E               ; Length + 1
F21A 1D         PRSLP:  DEC     E               ; Count characters
F21B C8                 RET     Z               ; End of string
F21C 0A                 LD      A,(BC)          ; Get byte to output
F21D CD9BE6             CALL    OUTC            ; Output character in A
F220 FE0D               CP      CR              ; Return?
F222 CC86EB             CALL    Z,DONULL        ; Yes - Do nulls
F225 03                 INC     BC              ; Next byte in string
F226 C31AF2             JP      PRSLP           ; More characters to output
```

```
F229 B7          TESTR:  OR      A               ; Test if enough room
F22A OE                  DEFB    (LD C,n)        ; No garbage collection done
F22B F1          GRBDON: POP     AF              ; Garbage collection done
F22C F5                  PUSH    AF              ; Save status
F22D 2A5A10              LD      HL,(STRSPC)     ; Bottom of string space in use
F230 EB                  EX      DE,HL           ; To DE
F231 2AC310              LD      HL,(STRBOT)     ; Bottom of string area
F234 2F                  CPL                     ; Negate length (Top down)
F235 4F                  LD      C,A             ; -Length to BC
F236 06FF                LD      B,-1            ; BC = -ve length of string
F238 09                  ADD     HL,BC           ; Add to bottom of space in use
F239 23                  INC     HL              ; Plus one for 2's complement
F23A CD8AE6              CALL    CPDEHL          ; Below string RAM area?
F23D DA47F2              JP      C,TESTOS        ; Tidy up if not done else err
F240 22C310              LD      (STRBOT),HL     ; Save new bottom of area
F243 23                  INC     HL              ; Point to first byte of string
F244 EB                  EX      DE,HL           ; Address to DE
F245 F1          POPAF:  POP     AF              ; Throw away status push
F246 C9                  RET

F247 F1          TESTOS: POP     AF              ; Garbage collect been done?
F248 1E1A                LD      E,OS            ; ?OS Error
F24A CAC1E3              JP      Z,ERROR         ; Yes - Not enough string space
F24D BF                  CP      A               ; Flag garbage collect done
F24E F5                  PUSH    AF              ; Save status
F24F 012BF2              LD      BC,GRBDON       ; Garbage collection done
F252 C5                  PUSH    BC              ; Save for RETurn
F253 2AAF10      GARBGE: LD      HL,(LSTRAM)     ; Get end of RAM pointer
F256 22C310      GARBLP: LD      (STRBOT),HL     ; Reset string pointer
F259 210000              LD      HL,0
F25C E5                  PUSH    HL              ; Flag no string found
F25D 2A5A10              LD      HL,(STRSPC)     ; Get bottom of string space
F260 E5                  PUSH    HL              ; Save bottom of string space
F261 21B310              LD      HL,TMSTPL       ; Temporary string pool
F264 EB          GRBLP:  EX      DE,HL
F265 2AB110              LD      HL,(TMSTPT)     ; Temporary string pool pointer
F268 EB                  EX      DE,HL
F269 CD8AE6              CALL    CPDEHL          ; Temporary string pool done?
F26C 0164F2              LD      BC,GRBLP        ; Loop until string pool done
F26F C2B8F2              JP      NZ,STPOOL       ; No - See if in string area
F272 2AD610              LD      HL,(PROGND)     ; Start of simple variables
F275 EB          SMPVAR: EX      DE,HL
F276 2AD810              LD      HL,(VAREND)     ; End of simple variables
F279 EB                  EX      DE,HL
F27A CD8AE6              CALL    CPDEHL          ; All simple strings done?
F27D CA8BF2              JP      Z,ARRLP         ; Yes - Do string arrays
F280 7E                  LD      A,(HL)          ; Get type of variable
F281 23                  INC     HL
F282 23                  INC     HL
F283 B7                  OR      A               ; "S" flag set if string
F284 CDBBF2              CALL    STRADD          ; See if string in string area
F287 C375F2              JP      SMPVAR          ; Loop until simple ones done
```



```
F28A C1          GNXARY: POP     BC              ; Scrap address of this array
F28B EB          ARRLP:  EX      DE,HL
F28C 2ADA10              LD      HL,(ARREND)     ; End of string arrays
F28F EB                  EX      DE,HL
F290 CD8AE6              CALL    CPDEHL          ; All string arrays done?
F293 CAE1F2              JP      Z,SCNEND        ; Yes - Move string if found
F296 CD62F8              CALL    LOADFP          ; Get array name to BCDE
F299 7B                  LD      A,E             ; Get type of array
F29A E5                  PUSH    HL              ; Save address of num of dim'ns
F29B 09                  ADD     HL,BC           ; Start of next array
F29C B7                  OR      A               ; Test type of array
F29D F28AF2              JP      P,GNXARY        ; Numeric array - Ignore it
F2A0 22C510              LD      (CUROPR),HL     ; Save address of next array
F2A3 E1                  POP     HL              ; Get address of num of dim'ns
F2A4 4E                  LD      C,(HL)          ; BC = Number of dimensions
F2A5 0600                LD      B,0
F2A7 09                  ADD     HL,BC           ; Two bytes per dimension size
F2A8 09                  ADD     HL,BC
F2A9 23                  INC     HL              ; Plus one for number of dim'ns
F2AA EB          GRBARY: EX      DE,HL
F2AB 2AC510              LD      HL,(CUROPR)     ; Get address of next array
F2AE EB                  EX      DE,HL
F2AF CD8AE6              CALL    CPDEHL          ; Is this array finished?
F2B2 CA8BF2              JP      Z,ARRLP         ; Yes - Get next one
F2B5 01AAF2              LD      BC,GRBARY       ; Loop until array all done
F2B8 C5          STPOOL: PUSH    BC              ; Save return address
F2B9 F680                OR      80H             ; Flag string type
F2BB 7E          STRADD: LD      A,(HL)          ; Get string length
F2BC 23                  INC     HL
F2BD 23                  INC     HL
F2BE 5E                  LD      E,(HL)          ; Get LSB of string address
F2BF 23                  INC     HL
F2C0 56                  LD      D,(HL)          ; Get MSB of string address
F2C1 23                  INC     HL
F2C2 F0                  RET     P               ; Not a string - Return
F2C3 B7                  OR      A               ; Set flags on string length
F2C4 C8                  RET     Z               ; Null string - Return
F2C5 44                  LD      B,H             ; Save variable pointer
F2C6 4D                  LD      C,L
F2C7 2AC310              LD      HL,(STRBOT)     ; Bottom of new area
F2CA CD8AE6              CALL    CPDEHL          ; String been done?
F2CD 60                  LD      H,B             ; Restore variable pointer
F2CE 69                  LD      L,C
F2CF D8                  RET     C               ; String done - Ignore
F2D0 E1                  POP     HL              ; Return address
F2D1 E3                  EX      (SP),HL         ; Lowest available string area
F2D2 CD8AE6              CALL    CPDEHL          ; String within string area?
F2D5 E3                  EX      (SP),HL         ; Lowest available string area
F2D6 E5                  PUSH    HL              ; Re-save return address
F2D7 60                  LD      H,B             ; Restore variable pointer
F2D8 69                  LD      L,C
F2D9 D0                  RET     NC              ; Outside string area - Ignore
F2DA C1F1F1              POP     BC,AF,AF        ; Get return , Throw 2 away
F2DD E5                  PUSH    HL              ; Save variable pointer
F2DE D5                  PUSH    DE              ; Save address of current
F2DF C5                  PUSH    BC              ; Put back return address
F2E0 C9                  RET                     ; Go to it
```

```
F2E1 D1E1      SCNEND: POP     DE,HL           ; Addresses of strings
F2E3 7D                LD      A,L             ; HL = 0 if no more to do
F2E4 B4                OR      H
F2E5 C8                RET     Z               ; No more to do - Return
F2E6 2B                DEC     HL
F2E7 46                LD      B,(HL)          ; MSB of address of string
F2E8 2B                DEC     HL
F2E9 4E                LD      C,(HL)          ; LSB of address of string
F2EA E5                PUSH    HL              ; Save variable address
F2EB 2B                DEC     HL
F2EC 2B                DEC     HL
F2ED 6E                LD      L,(HL)          ; HL = Length of string
F2EE 2600              LD      H,0
F2F0 09                ADD     HL,BC           ; Address of end of string+1
F2F1 50                LD      D,B             ; String address to DE
F2F2 59                LD      E,C
F2F3 2B                DEC     HL              ; Last byte in string
F2F4 44                LD      B,H             ; Address to BC
F2F5 4D                LD      C,L
F2F6 2AC310            LD      HL,(STRBOT)     ; Current bottom of string area
F2F9 CD7CE3            CALL    MOVSTR          ; Move string to new address
F2FC E1                POP     HL              ; Restore variable address
F2FD 71                LD      (HL),C          ; Save new LSB of address
F2FE 23                INC     HL
F2FF 70                LD      (HL),B          ; Save new MSB of address
F300 69                LD      L,C             ; Next string area+1 to HL
F301 60                LD      H,B
F302 2B                DEC     HL              ; Next string area address
F303 C356F2            JP      GARBLP          ; Look for more strings

F306 C5E5      CONCAT: PUSH    BC,HL           ; Save prec' opr & code string
F308 2AE410            LD      HL,(FPREG)      ; Get first string
F30B E3                EX      (SP),HL         ; Save first string
F30C CDD1ED            CALL    OPRND           ; Get second string
F30F E3                EX      (SP),HL         ; Restore first string
F310 CD45ED            CALL    TSTSTR          ; Make sure it's a string
F313 7E                LD      A,(HL)          ; Get length of second string
F314 E5                PUSH    HL              ; Save first string
F315 2AE410            LD      HL,(FPREG)      ; Get second string
F318 E5                PUSH    HL              ; Save second string
F319 86                ADD     A,(HL)          ; Add length of second string
F31A 1E1C              LD      E,LS            ; ?LS Error
F31C DAC1E3            JP      C,ERROR         ; String too long - Error
F31F CDBFF1            CALL    MKTMST          ; Make temporary string
F322 D1                POP     DE              ; Get second string to DE
F323 CD57F3            CALL    GSTRDE          ; Move to string pool if needed
F326 E3                EX      (SP),HL         ; Get first string
F327 CD56F3            CALL    GSTRHL          ; Move to string pool if needed
F32A E5                PUSH    HL              ; Save first string
F32B 2AC110            LD      HL,(TMPSTR+2)   ; Temporary string address
F32E EB                EX      DE,HL           ; To DE
F32F CD3DF3            CALL    SSTSA           ; First string to string area
F332 CD3DF3            CALL    SSTSA           ; Second string to string area
F335 2166ED            LD      HL,EVAL2        ; Return to evaluation loop
F338 E3                EX      (SP),HL         ; Save return,get code string
F339 E5                PUSH    HL              ; Save code string address
F33A C3F0F1            JP      TSTOPL          ; To temporary string to pool
```



```
F33D E1        SSTSA:  POP     HL              ; Return address
F33E E3                EX      (SP),HL         ; Get string block,save return
F33F 7E                LD      A,(HL)          ; Get length of string
F340 23                INC     HL
F341 23                INC     HL
F342 4E                LD      C,(HL)          ; Get LSB of string address
F343 23                INC     HL
F344 46                LD      B,(HL)          ; Get MSB of string address
F345 6F                LD      L,A             ; Length to L
F346 2C        TOSTRA: INC     L               ; INC - DECed after
F347 2D        TSALP:  DEC     L               ; Count bytes moved
F348 C8                RET     Z               ; End of string - Return
F349 0A                LD      A,(BC)          ; Get source
F34A 12                LD      (DE),A          ; Save destination
F34B 03                INC     BC              ; Next source
F34C 13                INC     DE              ; Next destination
F34D C347F3            JP      TSALP           ; Loop until string moved

F350 CD45ED    GETSTR: CALL    TSTSTR          ; Make sure it's a string
F353 2AE410    GSTRCU: LD      HL,(FPREG)      ; Get current string
F356 EB        GSTRHL: EX      DE,HL           ; Save DE
F357 CD71F3    GSTRDE: CALL    BAKTMP          ; Was it last tmp-str?
F35A EB                EX      DE,HL           ; Restore DE
F35B C0                RET     NZ              ; No - Return
F35C D5                PUSH    DE              ; Save string
F35D 50                LD      D,B             ; String block address to DE
F35E 59                LD      E,C
F35F 1B                DEC     DE              ; Point to length
F360 4E                LD      C,(HL)          ; Get string length
F361 2AC310            LD      HL,(STRBOT)     ; Current bottom of string area
F364 CD8AE6            CALL    CPDEHL          ; Last one in string area?
F367 C26FF3            JP      NZ,POPHL        ; No - Return
F36A 47                LD      B,A             ; Clear B (A=0)
F36B 09                ADD     HL,BC           ; Remove string from str' area
F36C 22C310            LD      (STRBOT),HL     ; Save new bottom of str' area
F36F E1        POPHL:  POP     HL              ; Restore string
F370 C9                RET

F371 2AB110    BAKTMP: LD      HL,(TMSTPT)     ; Get temporary string pool top
F374 2B                DEC     HL              ; Back
F375 46                LD      B,(HL)          ; Get MSB of address
F376 2B                DEC     HL              ; Back
F377 4E                LD      C,(HL)          ; Get LSB of address
F378 2B                DEC     HL              ; Back
F379 2B                DEC     HL              ; Back
F37A CD8AE6            CALL    CPDEHL          ; String last in string pool?
F37D C0                RET     NZ              ; Yes - Leave it
F37E 22B110            LD      (TMSTPT),HL     ; Save new string pool top
F381 C9                RET
```

```
F382 0101F1     LEN:    LD      BC,PASSA        ; To return integer A
F385 C5                 PUSH    BC              ; Save address
F386 CD50F3     GETLEN: CALL    GETSTR          ; Get string and its length
F389 AF                 XOR     A
F38A 57                 LD      D,A             ; Clear D
F38B 32AD10             LD      (TYPE),A        ; Set type to numeric
F38E 7E                 LD      A,(HL)          ; Get length of string
F38F B7                 OR      A               ; Set status flags
F390 C9                 RET

F391 0101F1     ASC:    LD      BC,PASSA        ; To return integer A
F394 C5                 PUSH    BC              ; Save address
F395 CD86F3     GTFLNM: CALL    GETLEN          ; Get length of string
F398 CAA0E9             JP      Z,FCERR         ; Null string - Error
F39B 23                 INC     HL
F39C 23                 INC     HL
F39D 5E                 LD      E,(HL)          ; Get LSB of address
F39E 23                 INC     HL
F39F 56                 LD      D,(HL)          ; Get MSB of address
F3A0 1A                 LD      A,(DE)          ; Get first byte of string
F3A1 C9                 RET

F3A2 3E01       CHR:    LD      A,1             ; One character string
F3A4 CDBFF1             CALL    MKTMST          ; Make a temporary string
F3A7 CD87F4             CALL    MAKINT          ; Make it integer A
F3AA 2AC110             LD      HL,(TMPSTR+2)   ; Get address of string
F3AD 73                 LD      (HL),E          ; Save character
F3AE C1         TOPOOL: POP     BC              ; Clean up stack
F3AF C3F0F1             JP      TSTOPL          ; Temporary string to pool
```



```
F3B2 CD37F4     LEFT:   CALL    LFRGNM          ; Get number and ending ")"
F3B5 AF                 XOR     A               ; Start at first byte in string
F3B6 E3         RIGHT1: EX      (SP),HL         ; Save code string,Get string
F3B7 4F                 LD      C,A             ; Starting position in string
F3B8 E5         MID1:   PUSH    HL              ; Save string block address
F3B9 7E                 LD      A,(HL)          ; Get length of string
F3BA B8                 CP      B               ; Compare with number given
F3BB DAC0F3             JP      C,ALLFOL        ; All following bytes required
F3BE 78                 LD      A,B             ; Get new length
F3BF 11                 DEFB    (LD DE,nn)      ; Skip "LD C,0"
F3C0 0E00       ALLFOL: LD      C,0             ; First byte of string
F3C2 C5                 PUSH    BC              ; Save position in string
F3C3 CD29F2             CALL    TESTR           ; See if enough string space
F3C6 C1                 POP     BC              ; Get position in string
F3C7 E1                 POP     HL              ; Restore string block address
F3C8 E5                 PUSH    HL              ; And re-save it
F3C9 23                 INC     HL
F3CA 23                 INC     HL
F3CB 46                 LD      B,(HL)          ; Get LSB of address
F3CC 23                 INC     HL
F3CD 66                 LD      H,(HL)          ; Get MSB of address
F3CE 68                 LD      L,B             ; HL = address of string
F3CF 0600               LD      B,0             ; BC = starting address
F3D1 09                 ADD     HL,BC           ; Point to that byte
F3D2 44                 LD      B,H             ; BC = source string
F3D3 4D                 LD      C,L
F3D4 CDC2F1             CALL    CRTMST          ; Create a string entry
F3D7 6F                 LD      L,A             ; Length of new string
F3D8 CD46F3             CALL    TOSTRA          ; Move string to string area
F3DB D1                 POP     DE              ; Clear stack
F3DC CD57F3             CALL    GSTRDE          ; Move to string pool if needed
F3DF C3F0F1             JP      TSTOPL          ; Temporary string to pool

F3E2 CD37F4     RIGHT:  CALL    LFRGNM          ; Get number and ending ")"
F3E5 D1                 POP     DE              ; Get string length
F3E6 D5                 PUSH    DE              ; And re-save
F3E7 1A                 LD      A,(DE)          ; Get length
F3E8 90                 SUB     B               ; Move back N bytes
F3E9 C3B6F3             JP      RIGHT1          ; Go and get sub-string
```

```
F3EC EB        MID:    EX      DE,HL       ; Get code string address
F3ED 7E                LD      A,(HL)      ; Get next byte "," or ")"
F3EE CD3CF4            CALL    MIDNUM      ; Get number supplied
F3F1 04                INC     B           ; Is it character zero?
F3F2 05                DEC     B
F3F3 CAA0E9            JP      Z,FCERR     ; Yes - Error
F3F6 C5                PUSH    BC          ; Save starting position
F3F7 1EFF              LD      E,255       ; All of string
F3F9 FE29              CP      ")"         ; Any length given?
F3FB CA05F4            JP      Z,RSTSTR    ; No - Rest of string
F3FE CD90E6            CALL    CHKSYN      ; Make sure "," follows
F401 2C                DEFB    ","
F402 CD84F4            CALL    GETINT      ; Get integer 0-255
F405 CD90E6    RSTSTR: CALL    CHKSYN      ; Make sure ")" follows
F408 29                DEFB    ")"
F409 F1                POP     AF          ; Restore starting position
F40A E3                EX      (SP),HL     ; Get string,save code string
F40B 01B8F3            LD      BC,MID1     ; Continuation of MID$ routine
F40E C5                PUSH    BC          ; Save for return
F40F 3D                DEC     A           ; Starting position-1
F410 BE                CP      (HL)        ; Compare with length
F411 0600              LD      B,0         ; Zero bytes length
F413 D0                RET     NC          ; Null string if start past end
F414 4F                LD      C,A         ; Save starting position-1
F415 7E                LD      A,(HL)      ; Get length of string
F416 91                SUB     C           ; Subtract start
F417 BB                CP      E           ; Enough string for it?
F418 47                LD      B,A         ; Save maximum length available
F419 D8                RET     C           ; Truncate string if needed
F41A 43                LD      B,E         ; Set specified length
F41B C9                RET                 ; Go and create string

F41C CD86F3    VAL:    CALL    GETLEN      ; Get length of string
F41F CA33F6            JP      Z,RESZER    ; Result zero
F422 5F                LD      E,A         ; Save length
F423 23                INC     HL
F424 23                INC     HL
F425 7E                LD      A,(HL)      ; Get LSB of address
F426 23                INC     HL
F427 66                LD      H,(HL)      ; Get MSB of address
F428 6F                LD      L,A         ; HL = String address
F429 E5                PUSH    HL          ; Save string address
F42A 19                ADD     HL,DE       ;
F42B 46                LD      B,(HL)      ; Get end of string+1 byte
F42C 72                LD      (HL),D      ; Zero it to terminate
F42D E3                EX      (SP),HL     ; Save string end,get start
F42E C5                PUSH    BC          ; Save end+1 byte
F42F 7E                LD      A,(HL)      ; Get starting byte
F430 CD1AF9            CALL    ASCTFP      ; Convert ASCII string to FP
F433 C1                POP     BC          ; Restore end+1 byte
F434 E1                POP     HL          ; Restore end+1 address
F435 70                LD      (HL),B      ; Put back original byte
F436 C9                RET
```



```
F437 EB        LFRGNM: EX      DE,HL       ; Code string address to HL
F438 CD90E6            CALL    CHKSYN      ; Make sure ")" follows
F43B 29                DEFB    ")"
F43C C1        MIDNUM: POP     BC          ; Get return address
F43D D1                POP     DE          ; Get number supplied
F43E C5                PUSH    BC          ; Re-save return address
F43F 43                LD      B,E         ; Number to B
F440 C9                RET

F441 CD87F4    INP:    CALL    MAKINT      ; Make it integer A
F444 323F10            LD      (INPORT),A  ; Set input port
F447 CD3E10            CALL    INPSUB      ; Get input from port
F44A C301F1            JP      PASSA       ; Return integer A

F44D CD71F4    POUT:   CALL    SETIO       ; Set up port number
F450 C30610            JP      OUTSUB      ; Output data and return

F453 CD71F4    WAIT:   CALL    SETIO       ; Set up port number
F456 F5                PUSH    AF          ; Save AND mask
F457 1E00              LD      E,0         ; Assume zero if none given
F459 2B                DEC     HL          ; DEC 'cos GETCHR INCs
F45A CD36E8            CALL    GETCHR      ; Get next character
F45D CA67F4            JP      Z,NOXOR     ; No XOR byte given
F460 CD90E6            CALL    CHKSYN      ; Make sure "," follows
F463 2C                DEFB    ","
F464 CD84F4            CALL    GETINT      ; Get integer 0-255 to XOR with
F467 C1        NOXOR:  POP     BC          ; Restore AND mask
F468 CD3E10    WAITLP: CALL    INPSUB      ; Get input
F46B AB                XOR     E           ; Flip selected bits
F46C A0                AND     B           ; Result non-zero?
F46D CA68F4            JP      Z,WAITLP    ; No = keep waiting
F470 C9                RET

F471 CD84F4    SETIO:  CALL    GETINT      ; Get integer 0-255
F474 323F10            LD      (INPORT),A  ; Set input port
F477 320710            LD      (OTPORT),A  ; Set output port
F47A CD90E6            CALL    CHKSYN      ; Make sure "," follows
F47D 2C                DEFB    ","
F47E C384F4            JP      GETINT      ; Get integer 0-255 and return

F481 CD36E8    FNDNUM: CALL    GETCHR      ; Get next character
F484 CD41ED    GETINT: CALL    GETNUM      ; Get a number from 0 to 255
F487 CD85E9    MAKINT: CALL    DEPINT      ; Make sure value 0 - 255
F48A 7A                LD      A,D         ; Get MSB of number
F48B B7                OR      A           ; Zero?
F48C C2A0E9            JP      NZ,FCERR    ; No - Error
F48F 2B                DEC     HL          ; DEC 'cos GETCHR INCs
F490 CD36E8            CALL    GETCHR      ; Get next character
F493 7B                LD      A,E         ; Get number to A
F494 C9                RET
```

```
                        ; << NO REFERENCE TO THIS SECTION OF CODE >>
                        ; << Set up another program area (can be in ROM) >>

F495 2A5E10             LD      HL,(BASTXT)     ; Get start of program text
F498 22D610             LD      (PROGND),HL     ; Set more variable space
F49B 210080             LD      HL,8000H        ; Address of new program
F49E 5E                 LD      E,(HL)          ; Get LSB of new RAM end
F49F 23                 INC     HL
F4A0 56                 LD      D,(HL)          ; Get MSB of new RAM end
F4A1 23                 INC     HL
F4A2 23                 INC     HL              ; Null at start of program
F4A3 225E10             LD      (BASTXT),HL     ; New program text area 8003H
F4A6 EB                 EX      DE,HL           ; New RAM end to HL
F4A7 22AF10             LD      (LSTRAM),HL     ; Set new RAM end
F4AA 225A10             LD      (STRSPC),HL     ; Clear string space
F4AD 01F2E7             LD      BC,RUNCNT       ; Execution driver loop
F4B0 C5                 PUSH    BC              ; Save for return
F4B1 C3C5E4             JP      RUNFST          ; Clear variables and continue

F4B4 C356FD     RUART:  JP      GUART           ; Get a byte from UART

F4B7 CDBAF4     WUART2: CALL    WUART           ; Send 2 Bytes to UART
F4BA F5         WUART:  PUSH    AF              ; Save byte
F4BB C5                 PUSH    BC              ; Save BC
F4BC 4F                 LD      C,A             ; Byte to C
F4BD CD68FD             CALL    SUART           ; Send byte to UART
F4C0 C1                 POP     BC              ; Restore BC
F4C1 F1                 POP     AF              ; Restore byte
F4C2 C9                 RET

F4C3 0601       CSAVE:  LD      B,1             ; Flag "CSAVE"
F4C5 FEAE               CP      ZTIMES          ; "*" token? ("CSAVE*")
F4C7 CABBE8             JP      Z,ARRSV1        ; Yes - Array save
F4CA CD5AED             CALL    EVAL            ; Evaluate expression
F4CD E5                 PUSH    HL              ; Save code string address
F4CE CD95F3             CALL    GTFLNM          ; Get file name
F4D1 D5                 PUSH    DE              ; Save file name
F4D2 CDC8FC             CALL    CASFFW          ; Turn on motor and wait
F4D5 D1                 POP     DE              ; Restore file name
F4D6 3ED3               LD      A,11010011B     ; Header byte
F4D8 CDBAF4             CALL    WUART           ; Send byte to UART
F4DB CDB7F4             CALL    WUART2          ; Send byte twice more
F4DE 1A                 LD      A,(DE)          ; Get file name
F4DF CDBAF4             CALL    WUART           ; Send it to UART
F4E2 00                 NOP
F4E3 00                 NOP
F4E4 00                 NOP
F4E5 21D610             LD      HL,PROGND       ; Start of program information
F4E8 220C0C             LD      (ARG1),HL       ; Save for monitor save routine
F4EB 2AD610             LD      HL,(PROGND)     ; End of program information
F4EE 220E0C             LD      (ARG2),HL       ; Save for monitor save routine
F4F1 CD73FE             CALL    SAVE            ; Save program to tape
F4F4 CDD8FC             CALL    ARET            ; Not much there!
F4F7 E1                 POP     HL              ; Restore code string address
F4F8 C9                 RET
```



```
F4F9 7E         CLOAD:  LD      A,(HL)          ; Get byte after "CLOAD"
F4FA FEAE               CP      ZTIMES          ; "*" token? ("CLOAD*")
F4FC CAB9E8             JP      Z,ARRLD1        ; Yes - Array load
F4FF CDD1FF             CALL    SMOTOR          ; Start motor and get "?"
F502 D69E               SUB     ZPRINT          ; "?" ("PRINT" token) Verify?
F504 CA09F5             JP      Z,FLGVER        ; Yes - Flag "verify"
F507 AF                 XOR     A               ; Flag "load"
F508 01                 DEFB    (LD BC,nn)      ; Skip "CPL" and "INC HL"
F509 2F         FLGVER: CPL                     ; Flag "verify"
F50A 23                 INC     HL              ; Skip over "?"
F50B F5                 PUSH    AF              ; Save verify flag
F50C 2B                 DEC     HL              ; DEC 'cos GETCHR INCs
F50D CD36E8             CALL    GETCHR          ; Get next character
F510 3E00               LD      A,0             ; Any file will do
F512 CA1CF5             JP      Z,ANYNAM        ; No name given - Any will do
F515 CD5AED             CALL    EVAL            ; Evaluate expression
F518 CD95F3             CALL    GTFLNM          ; Get file name
F51B 1A                 LD      A,(DE)          ; Get first byte of name
F51C 6F         ANYNAM: LD      L,A             ; Save name to find
F51D F1                 POP     AF              ; Get verify flag
F51E F5                 PUSH    AF              ; And re-save
F51F B7                 OR      A               ; Verify of load?
F520 67                 LD      H,A
F521 22E410             LD      (FPREG),HL      ; Save nam of file to find
F524 CCBAE4             CALL    Z,CLRPTR        ; Load - Clear pointers
F527 2AE410             LD      HL,(FPREG)      ; Get name of program to find
F52A EB                 EX      DE,HL           ; Name to DE
F52B 0603       CLOAD1: LD      B,3             ; 3 Header bytes
F52D CDB4F4     CLOAD2: CALL    RUART           ; Get a byte from UART
F530 D6D3               SUB     11010011B       ; Header byte?
F532 C22BF5             JP      NZ,CLOAD1       ; Look for header
F535 05                 DEC     B               ; Count header bytes
F536 C22DF5             JP      NZ,CLOAD2       ; More to find?
F539 CDB4F4             CALL    RUART           ; Get name of file
F53C CD74F5             CALL    FILFND          ; Display "file X found"
F53F 1C                 INC     E               ; Any file name given?
F540 1D                 DEC     E
F541 CA48F5             JP      Z,THSFIL        ; No - This file will do
F544 BB                 CP      E               ; Has file been found?
F545 C22BF5             JP      NZ,CLOAD1       ; No - Look for another
F548 00         THSFIL: NOP
F549 00                 NOP
F54A 00                 NOP
F54B F1                 POP     AF              ; Get verify flag
F54C B7                 OR      A               ; Load or verify?
F54D C25CF5             JP      NZ,CLOADV       ; Verify program
F550 CD88FE             CALL    MONLD           ; Use monitor to load program
F553 2AD610             LD      HL,(PROGND)     ; Get end of program
F556 CD93E3             CALL    ENFMEM          ; See if enough memory
F559 C35FF5             JP      CLOADE          ; "Ok" and set up pointers

F55C CDAAFE     CLOADV: CALL    MONVE           ; Use monitor to verify program
F55F 214BE3     CLOADE: LD      HL,OKMSG        ; "Ok" message
F562 CD10F2             CALL    PRS             ; Output string
F565 CDD8FC             CALL    ARET            ; Not a lot there!
F568 C37CE4             JP      SETPTR          ; Set up line pointers
```

```
F56B 219DF5     OUTBAD: LD      HL,BAD          ; "Bad" message
F56E CD10F2             CALL    PRS             ; Output string
F571 C3E1E3             JP      ERRIN           ; In line message

F574 C5         FILFND: PUSH    BC              ; <- Save
F575 E5                 PUSH    HL              ; <- all
F576 D5                 PUSH    DE              ; <- the
F577 F5                 PUSH    AF              ; <- registers
F578 218EF5             LD      HL,FILE         ; "File" message
F57B CD10F2             CALL    PRS             ; Output string
F57E F1                 POP     AF              ; Get file name
F57F F5                 PUSH    AF              ; And re-save
F580 CDD9FC             CALL    CONMON          ; Output file name to screen
F583 2194F5             LD      HL,FOUND        ; "Found" message
F586 CD10F2             CALL    PRS             ; Output string
F589 F1                 POP     AF              ; <- Restore
F58A D1                 POP     DE              ; <- all
F58B E1                 POP     HL              ; <- the
F58C C1                 POP     BC              ; <- registers
F58D C9                 RET

F58E 46696C65   FILE:   DEFB    "File ",0
F594 20466F75   FOUND:  DEFB    " Found",CR,LF,0
F59D 42616400   BAD:    DEFB    "Bad",0,0,0

F5A3 CD8BE9     PEEK:   CALL    DEINT           ; Get memory address
F5A6 1A                 LD      A,(DE)          ; Get byte in memory
F5A7 C301F1             JP      PASSA           ; Return integer A

F5AA CD41ED     POKE:   CALL    GETNUM          ; Get memory address
F5AD CD8BE9             CALL    DEINT           ; Get integer -32768 to 3276
F5B0 D5                 PUSH    DE              ; Save memory address
F5B1 CD90E6             CALL    CHKSYN          ; Make sure "," follows
F5B4 2C                 DEFB    ","
F5B5 CD84F4             CALL    GETINT          ; Get integer 0-255
F5B8 D1                 POP     DE              ; Restore memory address
F5B9 12                 LD      (DE),A          ; Load it into memory
F5BA C9                 RET

F5BB 2191FA     ROUND:  LD      HL,HALF         ; Add 0.5 to FPREG
F5BE CD62F8     ADDPHL: CALL    LOADFP          ; Load FP at (HL) to BCDE
F5C1 C3CDF5             JP      FPADD           ; Add BCDE to FPREG
```



```
F5C4 CD62F8     SUBPHL: CALL    LOADFP          ; FPREG = -FPREG + number at HL
F5C7 21                 DEFB    (LD HL,nn)      ; Skip "POP BC" and "POP DE"
F5C8 C1         PSUB:   POP     BC              ; Get FP number from stack
F5C9 D1                 POP     DE
F5CA CD3CF8     SUBCDE: CALL    INVSGN          ; Negate FPREG
F5CD 78         FPADD:  LD      A,B             ; Get FP exponent
F5CE B7                 OR      A               ; Is number zero?
F5CF C8                 RET     Z               ; Yes - Nothing to add
F5D0 3AE710             LD      A,(FPEXP)       ; Get FPREG exponent
F5D3 B7                 OR      A               ; Is this number zero?
F5D4 CA54F8             JP      Z,FPBCDE        ; Yes - Move BCDE to FPREG
F5D7 90                 SUB     B               ; BCDE number larger?
F5D8 D2E7F5             JP      NC,NOSWAP       ; No - Don't swap thm
F5DB 2F                 CPL                     ; Two's complement
F5DC 3C                 INC     A               ;  FP exponent
F5DD EB                 EX      DE,HL
F5DE CD44F8             CALL    STAKFP          ; Put FPREG on stack
F5E1 EB                 EX      DE,HL
F5E2 CD54F8             CALL    FPBCDE          ; Move BCDE to FPREG
F5E5 C1                 POP     BC              ; Restore number from stack
F5E6 D1                 POP     DE
F5E7 FE19       NOSWAP: CP      24+1            ; Second number insignificant?
F5E9 D0                 RET     NC              ; Yes - First number is result
F5EA F5                 PUSH    AF              ; Save number of bits to scale
F5EB CD79F8             CALL    SIGNS           ; Set MSBs & sign of result
F5EE 67                 LD      H,A             ; Save sign of result
F5EF F1                 POP     AF              ; Restore scaling factor
F5F0 CD92F6             CALL    SCALE           ; Scale BCDE to same exponent
F5F3 B4                 OR      H               ; Result to be positive?
F5F4 21E410             LD      HL,FPREG        ; Point to FPREG
F5F7 F20DF6             JP      P,MINCDE        ; No - Subtract FPREG from CDE
F5FA CD72F6             CALL    PLUCDE          ; Add FPREG to CDE
F5FD D253F6             JP      NC,RONDUP       ; No overflow - Round it up
F600 23                 INC     HL              ; Point to exponent
F601 34                 INC     (HL)            ; Increment it
F602 CABCE3             JP      Z,OVERR         ; Number overflowed - Error
F605 2E01               LD      L,1             ; 1 bit to shift right
F607 CDA8F6             CALL    SHRT1           ; Shift result right
F60A C353F6             JP      RONDUP          ; Round it up

F60D AF         MINCDE: XOR     A               ; Clear A and carry
F60E 90                 SUB     B               ; Negate exponent
F60F 47                 LD      B,A             ; Re-save exponent
F610 7E                 LD      A,(HL)          ; Get LSB of FPREG
F611 9B                 SBC     A,E             ; Subtract LSB of BCDE
F612 5F                 LD      E,A             ; Save LSB of BCDE
F613 23                 INC     HL
F614 7E                 LD      A,(HL)          ; Get NMSB of FPREG
F615 9A                 SBC     A,D             ; Subtract NMSB of BCDE
F616 57                 LD      D,A             ; Save NMSB of BCDE
F617 23                 INC     HL
F618 7E                 LD      A,(HL)          ; Get MSB of FPREG
F619 99                 SBC     A,C             ; Subtract MSB of BCDE
F61A 4F                 LD      C,A             ; Save MSB of BCDE
F61B DC7EF6     CONPOS: CALL    C,COMPL         ; Overflow - Make it positive
```

```
F61E 68          BNORM:  LD      L,B             ; L = Exponent
F61F 63                  LD      H,E             ; H = LSB
F620 AF                  XOR     A
F621 47          BNRMLP: LD      B,A             ; Save bit count
F622 79                  LD      A,C             ; Get MSB
F623 B7                  OR      A               ; Is it zero?
F624 C240F6              JP      NZ,PNORM        ; No - Do it bit at a time
F627 4A                  LD      C,D             ; MSB = NMSB
F628 54                  LD      D,H             ; NMSB= LSB
F629 65                  LD      H,L             ; LSB = VLSB
F62A 6F                  LD      L,A             ; VLSB= 0
F62B 78                  LD      A,B             ; Get exponent
F62C D608                SUB     8               ; Count 8 bits
F62E FEE0                CP      -24-8           ; Was number zero?
F630 C221F6              JP      NZ,BNRMLP       ; No - Keep normalising
F633 AF          RESZER: XOR     A               ; Result is zero
F634 32E710      SAVEXP: LD      (FPEXP),A       ; Save result as zero
F637 C9                  RET

F638 05          NORMAL: DEC     B               ; Count bits
F639 29                  ADD     HL,HL           ; Shift HL left
F63A 7A                  LD      A,D             ; Get NMSB
F63B 17                  RLA                     ; Shift left with last bit
F63C 57                  LD      D,A             ; Save NMSB
F63D 79                  LD      A,C             ; Get MSB
F63E 8F                  ADC     A,A             ; Shift left with last bit
F63F 4F                  LD      C,A             ; Save MSB
F640 F238F6      PNORM:  JP      P,NORMAL        ; Not done - Keep going
F643 78                  LD      A,B             ; Number of bits shifted
F644 5C                  LD      E,H             ; Save HL in EB
F645 45                  LD      B,L
F646 B7                  OR      A               ; Any shifting done?
F647 CA53F6              JP      Z,RONDUP        ; No - Round it up
F64A 21E710              LD      HL,FPEXP        ; Point to exponent
F64D 86                  ADD     A,(HL)          ; Add shifted bits
F64E 77                  LD      (HL),A          ; Re-save exponent
F64F D233F6              JP      NC,RESZER       ; Underflow - Result is zero
F652 C8                  RET     Z               ; Result is zero
F653 78          RONDUP: LD      A,B             ; Get VLSB of number
F654 21E710      RONDB:  LD      HL,FPEXP        ; Point to exponent
F657 B7                  OR      A               ; Any rounding?
F658 FC65F6              CALL    M,FPROND        ; Yes - Round number up
F65B 46                  LD      B,(HL)          ; B = Exponent
F65C 23                  INC     HL
F65D 7E                  LD      A,(HL)          ; Get sign of result
F65E E680                AND     10000000B       ; Only bit 7 needed
F660 A9                  XOR     C               ; Set correct sign
F661 4F                  LD      C,A             ; Save correct sign in number
F662 C354F8              JP      FPBCDE          ; Move BCDE to FPREG
```



```
F665 1C          FPROND: INC     E               ; Round LSB
F666 C0                  RET     NZ              ; Return if ok
F667 14                  INC     D               ; Round NMSB
F668 C0                  RET     NZ              ; Return if ok
F669 0C                  INC     C               ; Round MSB
F66A C0                  RET     NZ              ; Return if ok
F66B 0E80                LD      C,80H           ; Set normal value
F66D 34                  INC     (HL)            ; Increment exponent
F66E C0                  RET     NZ              ; Return if ok
F66F C3BCE3              JP      OVERR           ; Overflow error

F672 7E          PLUCDE: LD      A,(HL)          ; Get LSB of FPREG
F673 83                  ADD     A,E             ; Add LSB of BCDE
F674 5F                  LD      E,A             ; Save LSB of BCDE
F675 23                  INC     HL
F676 7E                  LD      A,(HL)          ; Get NMSB of FPREG
F677 8A                  ADC     A,D             ; Add NMSB of BCDE
F678 57                  LD      D,A             ; Save NMSB of BCDE
F679 23                  INC     HL
F67A 7E                  LD      A,(HL)          ; Get MSB of FPREG
F67B 89                  ADC     A,C             ; Add MSB of BCDE
F67C 4F                  LD      C,A             ; Save MSB of BCDE
F67D C9                  RET

F67E 21E810      COMPL:  LD      HL,SGNRES       ; Sign of result
F681 7E                  LD      A,(HL)          ; Get sign of result
F682 2F                  CPL                     ; Negate it
F683 77                  LD      (HL),A          ; Put it back
F684 AF                  XOR     A
F685 6F                  LD      L,A             ; Set L to zero
F686 90                  SUB     B               ; Negate exponent,set carry
F687 47                  LD      B,A             ; Re-save exponent
F688 7D                  LD      A,L             ; Load zero
F689 9B                  SBC     A,E             ; Negate LSB
F68A 5F                  LD      E,A             ; Re-save LSB
F68B 7D                  LD      A,L             ; Load zero
F68C 9A                  SBC     A,D             ; Negate NMSB
F68D 57                  LD      D,A             ; Re-save NMSB
F68E 7D                  LD      A,L             ; Load zero
F68F 99                  SBC     A,C             ; Negate MSB
F690 4F                  LD      C,A             ; Re-save MSB
F691 C9                  RET

F692 0600        SCALE:  LD      B,0             ; Clear underflow
F694 D608        SCALLP: SUB     8               ; 8 bits (a whole byte)?
F696 DAA1F6              JP      C,SHRITE        ; No - Shift right A bits
F699 43                  LD      B,E             ; <- Shift
F69A 5A                  LD      E,D             ; <- right
F69B 51                  LD      D,C             ; <- eight
F69C 0E00                LD      C,0             ; <- bits
F69E C394F6              JP      SCALLP          ; More bits to shift
```

```
F6A1 C609       SHRITE: ADD     A,8+1           ; Adjust count
F6A3 6F                 LD      L,A             ; Save bits to shift
F6A4 AF         SHRLP:  XOR     A               ; Flag for all done
F6A5 2D                 DEC     L               ; All shifting done?
F6A6 C8                 RET     Z               ; Yes - Return
F6A7 79                 LD      A,C             ; Get MSB
F6A8 1F         SHRT1:  RRA                     ; Shift it right
F6A9 4F                 LD      C,A             ; Re-save
F6AA 7A                 LD      A,D             ; Get NMSB
F6AB 1F                 RRA                     ; Shift right with last bit
F6AC 57                 LD      D,A             ; Re-save it
F6AD 7B                 LD      A,E             ; Get LSB
F6AE 1F                 RRA                     ; Shift right with last bit
F6AF 5F                 LD      E,A             ; Re-save it
F6B0 78                 LD      A,B             ; Get underflow
F6B1 1F                 RRA                     ; Shift right with last bit
F6B2 47                 LD      B,A             ; Re-save underflow
F6B3 C3A4F6             JP      SHRLP           ; More bits to do

F6B6 00000081   UNITY:  DEFB    000H,000H,000H,081H     ; 1.00000

F6BA 03         LOGTAB: DEFB    3                       ; Table used by LOG
F6BB AA561980           DEFB    0AAH,056H,019H,080H     ; 0.59898
F6BF F1227680           DEFB    0F1H,022H,076H,080H     ; 0.96147
F6C3 45AA3882           DEFB    045H,0AAH,038H,082H     ; 2.88539

F6C7 CD13F8     LOG:    CALL    TSTSGN          ; Test sign of value
F6CA B7                 OR      A
F6CB EAA0E9             JP      PE,FCERR        ; ?FC Error if <= zero
F6CE 21E710             LD      HL,FPEXP        ; Point to exponent
F6D1 7E                 LD      A,(HL)          ; Get exponent
F6D2 013580             LD      BC,8035H        ; BCDE = SQR(1/2)
F6D5 11F304             LD      DE,04F3H
F6D8 90                 SUB     B               ; Scale value to be < 1
F6D9 F5                 PUSH    AF              ; Save scale factor
F6DA 70                 LD      (HL),B          ; Save new exponent
F6DB D5                 PUSH    DE              ; Save SQR(1/2)
F6DC C5                 PUSH    BC
F6DD CDCDF5             CALL    FPADD           ; Add SQR(1/2) to value
F6E0 C1                 POP     BC              ; Restore SQR(1/2)
F6E1 D1                 POP     DE
F6E2 04                 INC     B               ; Make it SQR(2)
F6E3 CD69F7             CALL    DVBCDE          ; Divide by SQR(2)
F6E6 21B6F6             LD      HL,UNITY        ; Point to 1.
F6E9 CDC4F5             CALL    SUBPHL          ; Subtract FPREG from 1
F6EC 21BAF6             LD      HL,LOGTAB       ; Coefficient table
F6EF CD5BFB             CALL    SUMSER          ; Evaluate sum of series
F6F2 018080             LD      BC,8080H        ; BCDE = -0.5
F6F5 110000             LD      DE,0000H
F6F8 CDCDF5             CALL    FPADD           ; Subtract 0.5 from FPREG
F6FB F1                 POP     AF              ; Restore scale factor
F6FC CD8EF9             CALL    RSCALE          ; Re-scale number
F6FF 013180     MULLN2: LD      BC,8031H        ; BCDE = Ln(2)
F702 111872             LD      DE,7218H
F705 21                 DEFB    (LD HL,nn)      ; Skip "POP BC" and "POP DE"
```



```
F706 C1         MULT:   POP     BC              ; Get number from stack
F707 D1                 POP     DE
F708 CD13F8     FPMULT: CALL    TSTSGN          ; Test sign of FPREG
F70B C8                 RET     Z               ; Return zero if zero
F70C 2E00               LD      L,0             ; Flag add exponents
F70E CDD1F7             CALL    ADDEXP          ; Add exponents
F711 79                 LD      A,C             ; Get MSB of multiplier
F712 32F610             LD      (MULVAL),A      ; Save MSB of multiplier
F715 EB                 EX      DE,HL
F716 22F710             LD      (MULVAL+1),HL   ; Save rest of multiplier
F719 010000             LD      BC,0            ; Partial product (BCDE) = zero
F71C 50                 LD      D,B
F71D 58                 LD      E,B
F71E 211EF6             LD      HL,BNORM        ; Address of normalise
F721 E5                 PUSH    HL              ; Save for return
F722 212AF7             LD      HL,MULT8        ; Address of 8 bit multiply
F725 E5E5               PUSH    HL,HL           ; Save for NMSB,MSB
F727 21E410             LD      HL,FPREG        ; Point to number
F72A 7E         MULT8:  LD      A,(HL)          ; Get LSB of number
F72B 23                 INC     HL              ; Point to NMSB
F72C B7                 OR      A               ; Test LSB
F72D CA56F7             JP      Z,BYTSFT        ; Zero - shift to next byte
F730 E5                 PUSH    HL              ; Save address of number
F731 2E08               LD      L,8             ; 8 bits to multiply by
F733 1F         MUL8LP: RRA                     ; Shift LSB right
F734 67                 LD      H,A             ; Save LSB
F735 79                 LD      A,C             ; Get MSB
F736 D244F7             JP      NC,NOMADD       ; Bit was zero - Don't add
F739 E5                 PUSH    HL              ; Save LSB and count
F73A 2AF710             LD      HL,(MULVAL+1)   ; Get LSB and NMSB
F73D 19                 ADD     HL,DE           ; Add NMSB and LSB
F73E EB                 EX      DE,HL           ; Leave sum in DE
F73F E1                 POP     HL              ; Restore MSB and count
F740 3AF610             LD      A,(MULVAL)      ; Get MSB of multiplier
F743 89                 ADC     A,C             ; Add MSB
F744 1F         NOMADD: RRA                     ; Shift MSB right
F745 4F                 LD      C,A             ; Re-save MSB
F746 7A                 LD      A,D             ; Get NMSB
F747 1F                 RRA                     ; Shift NMSB right
F748 57                 LD      D,A             ; Re-save NMSB
F749 7B                 LD      A,E             ; Get LSB
F74A 1F                 RRA                     ; Shift LSB right
F74B 5F                 LD      E,A             ; Re-save LSB
F74C 78                 LD      A,B             ; Get VLSB
F74D 1F                 RRA                     ; Shift VLSB right
F74E 47                 LD      B,A             ; Re-save VLSB
F74F 2D                 DEC     L               ; Count bits multiplied
F750 7C                 LD      A,H             ; Get LSB of multiplier
F751 C233F7             JP      NZ,MUL8LP       ; More - Do it
F754 E1         POPHRT: POP     HL              ; Restore address of number
F755 C9                 RET

F756 43         BYTSFT: LD      B,E             ; Shift partial product left
F757 5A                 LD      E,D
F758 51                 LD      D,C
F759 4F                 LD      C,A
F75A C9                 RET
```

AUNT AGATHA'S AGONY COLUMN

By David Parkinson

GM809, GM829 compatibility/upgrading

This is a question that has cropped up several times recently. The answer is that the Gemini GM829 FDC/SASI board can be regarded as a GM809HL. With the 'HL' level of trim you gain software controlled 5.25"/8" switching together with a SASI interface. Other than that the products are identical (ports, software interface etc). For some one currently running a system with GM809, upgrading is a matter of a) getting a GM829; b) Checking the straps; c) Plugging it in. That's all there is to it.

The software controllable 5.25"/8" switching of GM829 is quite useful - even if you don't have 8" drives connected to your system. This is because the Western Digital Floppy disk controller used on GM809/GM829 is limited in the maximum rate it can step the head between tracks on the attached drives. When set for 5.25" drives it can only achieve a 6ms stepping rate, but the modern Japanese drives (e.g. TEAC FD55Es & Fs) can be stepped at 3ms/step. Running them at the slower rate results in reduced performance and a 'graunching' noise from the drive (non destructive!). However, with GM829, whenever a SEEK is required, the 5.25"/8" control can be flipped to 8". The main effect of this is to double the clock frequency to the Controller chip, which results in all stepping times being halved. Thus drives can now be stepped as fast as 3ms/step, leading to increased performance and much quieter stepping. (Once the seek is complete, the control bit is obviously flipped back to the 5.25" setting before doing the read/write operation.)

BASIC mathematics

A letter from Phil Dunglinson on the topic of a BASIC program that doesn't work provides me with my next topic. The listing is shown below:

```
10 FOR N3=1 TO 9
20 FOR N2=0 TO 9
30 FOR N1=0 TO 9
40 A= N1^3 + N2^3 + N3^3
50 B= N1 + N2*10 + N3*100
60 IF A=B THEN 80
70 GOTO 90
80 PRINT B
90 NEXT N1
100 NEXT N2
110 NEXT N3
120 END
```

It should print out the results of 153,370,371,407 but doesn't. Can anyone see why not?

The answer is simple, and reminds me of that old adage about not blindly accepting the answer that comes out of a computer. Just because your computer tells you that 2<>2 or 3=2 does not necessarily mean that it is true. Remember that your computer is an idiot and tries to do exactly what you tell it. It can do mundane operations very quickly, but it does have limitations and this example highlights twoof them.

Point 1: Derived Arithmetic Functions.

To us normal Human Beings N1^3 (N1 raised to the power 3) in line 40 of the above program means N1*N1*N1. However to Nascom BASIC it means: EXP(3*LOG(N1)), where both the LOG() and the EXP() are calculated by evaluating polynomial approximations of the form:

$$C0 + C1*X + C2*X^2 + C3*X^3 + \ldots + CN*X^N$$

X is the value passed to the approximation routine, and C0,C1,..CN are constants whose value depend upon the approximation required (EXP, LOG, SQR etc). Usually X has to be scaled to lie within a certain range for the approximation to be valid. The accuracy of the approximation obviously depends upon the degree of the polynomial, and for some functions a polynomial of low degree (of only 3 say) can be suprisingly accurate. From a computational speed

point of view, the shorter the polynomial, the faster it can be evaluated. For those who wish to pursue matters further various books can be found covering the topic. One fairly comprehensive book I have encountered is [1].

So point one is that derived functions such as "^", LOG, EXP, etc are written for the general case and have approximate results. (They may be accurate, but not necessarily exact.) They do not recognise specific cases (such as the exponent in a "^" expression being a small integer) and do not adjust their algorithms accordingly. This means that the instruction in line 60 (IF A=B THEN...) is almost certainly going to be false. If A=2 and B=1.999999999..... then they are not equal in the eyes of the computer although an engineer would happily accept them as such. (Mind you a Scientist may not, but that leads on to the old joke...) Therefore line 60 has to be rephrased as - IF A EQUALS B FOR ALL PRACTICAL PURPOSES THEN .... This can best be done by coding it as IF ABS(A-B)<1E-4 THEN.... Here we have said if the two values are within 1/10,000 of each other then take them as equal. After making this change the program above will run successfully.

Alternatively the program can be recoded to make the calculation more accurate. By recoding line 40 as ... N1*N1*N1 + N2*N2.... we replace the approximation by an accurate calculation. (Accurate in this case as we are dealing with reasonably sized INTEGERS. In other circumstances - e.g. N1 etc being REAL numbers like 2.345 and 7.916 - there would be rounding errors and possible dynamic range problems affecting the accuracy of the result.) This change also has the side effect of speeding up the program as the two multiplications are faster than the ^ function. In making this change to line 40, line 60 can be left as IF A=B... (but remember the caveat above).

Point 2: Binary Arithmetic

While we are on the topic of computer accuracy I'll just mention one other point. Computers that use binary arithmetic cannot hold most decimal fractions accurately. (e.g. 2.67 might be held as 'a number very close to 2.67'.) This is why any serious financial program always uses BCD (Binary Coded Decimal) arithmetic - where 2.67 IS 2.67 - rather than pure binary arithmetic. That way the books generally balance exactly rather than approximately as the arithmetic exactly matches the human 'pencil & paper' mode. Details of BCD algorithms can be found in [2].

Software Testing

The example above highlights another important point that we are all frequently guilty of, and that is inadequate testing of programs. This program is not perhaps the best example as it does not process any external data in order to produce its result. The important point though is that Phil Dunglinson knew what the program should do (in its present form) and when the correct results didn't emerge he knew there was a bug to find. Software testing is an art. (Think of the unlimited character combinations possible in the source input file for an assembler or compiler.) There are various books on the topic that you can read if you are interested [3][4]. I don't intend to cover software testing here, but one thing to remember is that any program that processes data, as well as producing correct output from correct input, must not accept incorrect input without complaining, or crash when presented with the unexpected.

Just to give a few examples of what I've encountered: First a minor bug illustrating what can happen to a program presented with the unexpected. With C/80 version 2.0 the compiler carried on compiling a source file past the end-of-file marker if the file ended in a TAB character rather than the usual CR/LF pair. (Easy enough to end up with a TAB if you use an on-screen editor.)

Next a minor bug but slightly more serious. Old habits die hard, and in some C/80 source files I entered a few hexadecimal constants with a trailing 'H' (e.g. 0xAH rather than the correct 0xA). The compiler accepted these with out comment and interpreted the 'H' as part of the Hex number so I ended up with an incorrectly evaluated constant. Finally my latest encounter has been with an unforgiveable bug from Microsoft in an 8086 cross assembler. Being relatively inexperienced in 8086 assembly language programming my initial programs contained some glaring errors like loading a segment register with an immediate 16-bit value. (Although you can load normal registers with immediate data, the segment registers can only be loaded with data from another register, or from memory.) The Microsoft assembler accepted my illegal instructions without complaint, and proceeded to generate an opcode for a completely different instruction. It was only by manually dis-assembling the opcodes on the assembly listing that I found the error. (Luckily, as I was just starting to familiarise myself with the 8086, the program was only some 25 lines long and it didn't long to find the error.)

One final comment on this topic - if you do find a bug REPORT IT, not by complaining to your friends, but to the Author. Document the bug thoroughly, for example by writing a four or five line program to illustrate it, and send the print-outs in with your comments. DON'T ASSUME SOMEBODY ELSE MUST HAVE ALREADY REPORTED IT. You might find that your report vanishes into a black hole, or, as I did with the Software Toolworks, get a polite note back and find the bugs corrected in the next release of the software (C/80 version 3.0).

Do-it-yourself Electrocution.(c.f. Richard Beal in the last issue!)

Next a request from Mr E.Jones for an article covering the general principles of converting a domestic TV set to a monitor, and interfacing a NASCOM to it. These sort of articles tend to be few and far between, and I assume that the reason for this is that no magazine editor wishes to be sued by the relatives of a late hobbyist who attempted to follow the article. VOLTAGES INSIDE A TV SET OR MONITOR ARE LETHAL. If you do not know what you are doing LEAVE THE BACK ON THE SET. If you think you know what you are doing I still advise you to leave the back on the set. With the proliferation of home computers and video recorders that has occured in recent years it should be possible to pick up a cheap black-and-white monitor, or to buy a TV set that already has a video input socket on the rear. For those who want to progress further I offer the following suggestions/observations.

(1) Before doing anything buy a copy of the service sheet or service manual for the set in question.

(2) Look at the power supply section of the diagram. Many sets directly rectify the mains input and then use the filtered DC supply either directly, (old Valved sets), or via a switch-mode step-down power supply (newer integrated sets). Certainly in the first case, and possibly in the second, you will find that one side of the TV chassis is connected directly to the NEUTRAL of the mains supply, (or directly to the LINE side if the plug has been wired incorrectly). In this case DO NOT PROCEED FURTHER unless you can isolate the TV supply from the mains, or provide a suitable barrier at the video interface into the TV. (A small black-and-white portable offering mains/battery operation will almost certainly have a mains transformer providing the required isolation.)

(3) The signal present at the VIDEO pin on the Nascom 2 circuit board is a convential composite video signal containing a mixture of Video and the horizontal and vertical sync pulses. This is similar to the signal present at the output of the detector following the IF stages in the TV. I haven't looked

at any TV circuit diagrams recently, but with transistorised sets this point can be found relatively easily. With a modern set full of ICs you may find that the point you're after lies within an IC. You need to inject the composite video signal (of the appropriate amplitude and polarity) at a point just before the video and sync signals are separated.
(4) ALWAYS SWITCH OFF AND ALLOW TIME FOR HIGH VOLTAGES TO DISCHARGE BEFORE MAKING ANY CHANGE TO YOUR CIRCUITS.
(5) If you must make any internal adjustments on a live set always use only one hand, keeping the other well out of the way (like in your pocket). DON'T GROPE ROUND THE BACK OF THE SET. If you need to see the screen while you make the adjustment sit behind the set and use a mirror. ALWAYS KEEP AN EYE ON THE HAND MAKING THE ADJUSTMENTS. As well as high voltages there are usually HOT resistors in the set. If you touch one of those inadvertantly who knows what your reflex action may make you come in contact with, and you may end up with far more than a burnt hand.

CP/M Users Group

Some time ago I mentioned the CP/M Users Group. The CP/M Users Group (UK) publishes a magazine (quarterly?) that varies in content. It normally contains news, reviews and comment. In the last issue (March 1984) over half the magazine was a listing of the new UK and SIG/M disks that have recently been added to the library. Currently a years subscription costs £7.50. The CP/M Users Group can be found at 72 Mill Road, Hawley, Dartford, Kent DA2 7RZ.

Using a Nascom as a counter-timer.

Next on the pile is a rather confused letter from someone in Southampton who had better remain nameless. (Anyway we reckon the name was a pseudonym.) He is wanting to use the interrupt line on a Nascom to measure the duration and frequency of a pulse stream. For some obscure reason (not stated) he wants to use interrupt mode 1. This effectively executes a 'RST 38' in response to an interrupt, vectoring to address 38H. However Nas-Sys has its "RDEL" routine at that address which is not much use in measuring anything! As a result he has got tied up in knots trying to get a RAM based version of Nas-Sys going. He is trying to do this so that he can patch the RST 38 location to jump to his own routine. I'll ignore the mire he has got into with his attempt to get Nas-Sys into RAM and point out the large errors he has made and a possible approach to the problem.

1) The interrupt problem.

Use Mode 2. There is nothing mysterious about it, it just needs a little thought as there are two levels of indirection to go through before you arrive at your service routine which can be anywhere in memory. If you must use mode 1 why not use NMI instead of INT? In response to an NMI, Nas-Sys 3 vectors via a JP stored in RAM at 0C7D. (i.e. At address 66 - the NMI execution address - Nas-Sys has a JP 0C7D. It intialises 0C7D to a JP <register display routine>.) Thus anyone wanting to use the NMI for another purpose can change the JP stored in the workspace area so that control passes to their routine, rather than back into Nas-Sys. (If necessary you connect your interrupt signal via a 2-input OR gate (e.g. 74LS32) to the NMI input. By connecting the other input of the OR gate to a spare bit on the PIO you can have a 'maskable' NMI).

2) Why there is no point in doing it anyway.

To go back to his original requirement. "To input a stream of variable length pulses direct into the Z80 pin 16 /INT, and thereafter analyse their lengths & groupings by program". He has started off by making a fundamental

error. The INT input to the Z80 is level sensitive. i.e. While the /INT signal is low an interrupt will occur immediately interrupts are enabled. I cannot conceive of how the /INT pin could be reasonably used to measure the duration of pulses. (The only thing that comes to mind is a service routine that starts EI, NOP, followed by code that counts the number of return addresses that have been pushed onto the stack by the successive interrupts - hardly a practical proposition.) The situation with the NMI input is virtually the same. The only difference is that it is an edge triggered input. The NMI is generated on the high-to-low transition of the signal on this pin and the low-to-high transition has no effect. Therefore a program could collect statistics of the frequency of occurrence of the pulses, but could not determine anything about their duration. Neither pin is suitable for the measurement of pulse widths.

3) Try the PIO.

The obvious canditate for this task is the PIO. It can be set into mode 3 (control mode) and the levels present on one or more of the external inputs can be monitored by the control software. But before anything can be done we need to know something about the characteristics of the pulses that are to be monitored. Obviously if they are very short (of the order of luS) and very frequent there is no way that the Z80 can be used for this purpose. Pulses of milliseconds or seconds duration can be measured relatively easily, but a few calculations should be done to determine the relative accuracy obtainable, and whether it is acceptable.

A simple timing loop could be:

```
      ld   hl,0       ; Clear counter
loop: inc  hl         ; Bump counter
      in   a,(pio)    ; Read port
      rlca            ; Put bit into carry
      jp   c,loop     ; Loop if still ON
      ....            ; Check count & save data
```

With this there is obviously an uncertainty equal to the software loop time in determining when the pulse goes OFF. (It could go OFF just before or just after the IN instruction is executed. In the latter case another complete loop is executed although the time difference between the two cases could only be nanoseconds.) Similarly there can be a similar uncertainty in recognising the ON instant. The basic unit measurement here is the 'LOOP' (above) and we can measure to an accuracy of +/- 2 LOOPS.

The above is a very simplified view of matters as there is also the problem of measuring the intervals between pulses. (Note also the above loop assumes that a 16-bit counter is adequate as the loop counter. For long pulses this is unlikely to be so, and the loop will have to be expanded to include an overflow check of the counter. However, as the pulse is longer, we can accept a greater absolute error as the relative error will still probably be small.)

The only thing to do is to sit down and write some of the software and start counting clock cycles. You may find it advantageous to add dummy instructions into some conditional paths so that all routes from point A in the software to point B take about the same amount of time. That way a software timer-counter can be updated with a fixed number to compensate for the execution of a particular service routine. (e.g. The routine that takes the last pulse measurement and updates a histogram table based upon the pulse width in 'loop counts'.)

Life may be easier if you use an I/O board which includes a CTC (Counter Timer Circuit).

NASCOM(?) 1.5(?)
Finally a letter from Steve Waites which I reproduce below:

"Back in the dark annals of history I had a perfectly good and working Nascom 1 + RAM A card. Simple but effective. Then I left to work in America so the poor old computer got left behind. Two years later I returned, and found that the old Nascom had pined itself away to an early demise. Checking the boards I found several of the 2102 RAMs and a few TTL chips had gone to the great semiconductor heaven. In desperation, and because I only had two weeks in England, I grabbed all the relavent software and made the decision to rebuild and upgrade. Several months of hard effort later I now welcome the Nascom 1.5. Why 1.5? Well its no longer a Nascom 1, nor is it quite a Nascom 2. My additions are an enhanced RS232 port, 18K of CMOS RAM on the main board plus 4118 RAMs for the video and user areas. Oh yes it also has a sound chip, a battery backed RAM area, graphics, and the memory is fully decoded in 2K blocks by a 74LS154.

"So far so good - everything works wonderfully - now to my question. Is there anyway that I can get my ZEAP (level 2.1 in ROM) to output continuously to the UART port as I usually use my computer with a slave CRT connected to the RS232 interface rather than the internal video. Also, but not quite so important, I have the Bits & PCs programmers toolkit. It won't accept commands from the remote CRT keyboard. Both Nas-Sys and BASIC have no problems in this respect, so what is the difference?"

Perhaps some of you ZEAP hackers can write in with a solution for Steve. I believe that ZEAP writes directly into the bottom line of the NASCOM display. This is done for speed, as outputing a character at a time via Nas-Sys would have an impact on the already slow assembly speed of ZEAP. How much of ZEAP's output goes this way I have no idea. As for the Bits & PCs toolkit, this uses its own input and output tables, and also utilises UIN and UOUT. I assume that a small error must exist in this somewhere which prevents it scanning the serial input port. Perhaps some enterprising person who is using the toolkit can send in a suitable patch that can be published in a subsequent issue of the NEWS?

Ta Ta.

References:
1. HART J.F, CHENEY E.W. et al,"Computer Approximations", Pub: John Wiley 1978
2. SCHMID H., "Decimal Computation", Pub: John Wiley 1974
3. MYERS G.J., "The Art of Software Testing", Pub: John Wiley 1979.
4. BRUCE R.C., "Software debugging for Microcomputers", Reston Publ. Co. 1980

Arfon Speech Board - #50. 16K RAM A board - #15. 32K RAM B board - #40. Easicomp PSG board (minus AY-3-8910 chip) - #10. Machine Code Programming for the Nascom Book - #2. Phone Kevin on 0224-36160 (evenings).

IBM Selectric KBD printer, ex. 2741, with hardware/sofware interface for Nascom & Nas-Sys. Uses 8 bits of PIO. Previously IBM maintained, in excellent working order. Sensible offers please to Ian on Ipswich (0473) 831353.

Polydos File Update

By M. J. R. Gibbs

This is a program for directly updating files on disk for a Gemini GM809/GM815 system with Polydos 2.0.

This allows the user to update copies of a file stored on several disks without having to enter all the normal commands which can cause problems especialy if you are like me and sometimes forget to use the 'NEW' command (I have spent many hours trying to sort out the chaos that this causes). The required program/data is loaded into RAM between #1000 and #C000 either by using the Polydos 'Read' command or by Assembling the program directly into RAM.

This program is loaded into RAM at #0C80 and executed at #0C80, the user is asked for the file name, file extension and the RAM location for the new code. The Program asks you to insert the disks and press the 'Enter' key, the disk directory is read and the filename checked and, should it exist, the file is replaced with the new version. The old file is completely overwritten and this means that the number of sectors used by the old file are replaced by the program stored in RAM. The user should be careful in ensuring that the new version is not going to require more sectors on disk than the original otherwise the whole of the new program will not be saved. I normally save programs that are common to several disks with a few sectors more than required to allow for expansion, this does not normally cause a problem as I have found that the maximum number of fifty files means that the disk is not often full before the directory is used up. This program can update a disk with a full directory because the directory is left unaltered.

Should the program not exist on the disk inserted a suitable message is printed out telling the user and the disk is left unaltered.

The Program informs the user of the start disk sector and the number of sectors updated and the RAM location used for the update. Should there be any disk errors then a message is printed out and the program repeats the inital menu asking the user to insert a new disk. The one thing that I have found is that it is easy to forget to remove any write protect tabs and this does not cause a problem as you simply take the disk out remove the tab and reinsert the disk (do not forget to replace the tab afterwards).

A word of warning when testing this program especially when entering it from the dump listing, which is always prone to errors, you are strongly advised to test it on a copy of an existing disk because it can overwrite irreplaceable portions of disk if anything goes wrong.

I suggest that you proceed as follows:-

1) Enter the program and save it on disk using Polydos. Use the 'Read' routine to load a file from the disk to say #1000 in RAM.
2) Examine and make a note of the number of sectors that this file uses.
3) Use SZAP to examine the next file on the disk and make a note of the first few bytes.
4) Change the first few bytes of the file loaded into RAM with the NAS-SYS 'M' (modify command).
5) Run this program to update the disk and check the user information for the correct sector location, number of sectors and RAM location.
6) Use SZAP to verify that the original file has been updated correctly. Also that the file following is not corrupted.

Below is a full Assembly listing, a sorted symbol table and a dump of the program using a modified version of the disk dump published in Vol.1 iss. 2 of 80-BUS NEWS (the numbers on the left hand side are the RAM locations).

```
0F35                    ;*************************
0F35                    ;*  DISK UPDATE ROUTINE  *
0F35                    ;* --------------------- *
0F35                    ;*                       *
0F35                    ;* M.J.R GIBBS 29-12-82  *
0F35                    ;*************************
0018        NASSYS EQU  #18
0028        PRS    EQU  #28
0030        ROUT   EQU  #30
0030                                ;NASSYS COMMANDS
007B        BLINK  EQU  #7B
0068        B2HEX  EQU  #68
0065        CRT    EQU  #65
0063        INLIN  EQU  #63
0064        NUM    EQU  #64
005B        RETNAS EQU  #5B
0066        TBCD3  EQU  #66
005D        TDEL   EQU  #5D
005D                                ;DISK COMMANDS
0080        ZDSIZE EQU  #80         ;DISK SIZE
0081        ZDRD   EQU  #81         ;READ
0082        ZDWR   EQU  #82         ;WRITE
0083        ZRDIR  EQU  #83         ;READ DIRECTORY
0084        ZWDIR  EQU  #84         ;WRITE DIRECTORY
0085        ZCFS   EQU  #85         ;CONVET FILE SPECIFIER
0087        ZENTER EQU  #87         ;UPDATE DIRECTORY
0088        ZCOV   EQU  #88         ;CALL OVERLAY
0089        ZCOVR  EQU  #89         ;CALL OVERLAY RESTORE
008A        ZCKER  EQU  #8A         ;CHECK FOR ERRORS
008B        ZCBRK  EQU  #8B         ;CHECK FOR BREAK
008C        ZCFMA  EQU  #8C         ;ABORT COMMAND MODE
0086        ZLOOK  EQU  #86         ;LOOKUP FILE DIRECTORY
0063        ZINLIN EQU  #63         ;MONITOR INPUT LINE
0063                                ;DISK LOCATIONS
C055        S1FCB  EQU  #C055
C069        S2FCB  EQU  #C069
C001        DDRV   EQU  #C001       ;DIRECTORY DRIVE
C414        NXTSEC EQU  #C414       ;NEXT FREE SECTOR
C414                                ;DUMMY FCB
C055              ORG   S1FCB
C055              LOAD  0
C055        FNAM   DS   8           ;FILE   NAME
C05D        FEXT   DS   2           ;FILE   EXTENSION
C05F        FSFL   DS   1           ;SYSTEM FLAG
C060        FUFL   DS   1           ;USER   FLAG
C061        FSEC   DS   2           ;SECTOR ADDRESS
C063        FNSC   DS   2           ;NUMBER SECTORS
C065        FLDA   DS   2           ;LOAD   ADDRESS
C067        FEXA   DS   2           ;EXEC   ADDRESS
C067                                ;SCREEN ADDRESSES
C067                                ;
080A        LINE1  EQU  #080A
0C29        CURSOR EQU  #0C29
```



```
0C21        NUMV   EQU  #0C21
000C        CLEAR  EQU  #0C
000D        CR     EQU  #0D
001B        ESC    EQU  #1B
001B                                ;
0C80              ORG   #0C80
0C80              LOAD  #8000
0C80                                ;
0C80                                ;--- DISK UPDATE ---
0C80                                ;
0C80 CD230E UPDATE CALL  HEAD       ;WRITE HEADINGS
0C83 EF            RST   PRS
0C84 4469736B
0C88 20557064
0C8C 61746520
0C90 46696C65
0C94 204E616D
0C98 65202E2E
0C9C 2E3A2D20
0CA0 00            DB    'Disk Update File Name ...:- ',0
0CA1 DF            RST   NASSYS
0CA2 63            DB    ZINLIN        ;GET REPLY
0CA3 211C00        LD    HL,28
0CA6 19            ADD   HL,DE         ;HL = A(START OF REPLY)
0CA7 1155C0        LD    DE,FNAM       ;DE = A(S1FCB)
0CAA 010800        LD    BC,8
0CAD EDB0          LDIR                ;LOAD NAME
0CAF EF            RST   PRS
0CB0 0D            DB    CR
0CB1 46696C65
0CB5 20457874
0CB9 656E7369
0CBD 6F6E202E
0CC1 2E2E2E2E
0CC5 2E2E2E2E
0CC9 2E3A2D20
0CCD 00            DB    'File Extension .........:- ',0
0CCE DF            RST   NASSYS
0CCF 63            DB    INLIN
0CD0 211C00        LD    HL,28
0CD3 19            ADD   HL,DE         ;HL = A(EXT)
0CD4 115DC0        LD    DE,FEXT       ;DE = A(FEXT)
0CD7 010200        LD    BC,2
0CDA EDB0          LDIR                ;LOAD FEXT
0CDC EF            RST   PRS
0CDD 0D            DB    CR
0CDE 52414D20
0CE2 53746172
0CE6 74204164
0CEA 64726573
0CEE 73202E2E
0CF2 2E2E2E2E
0CF6 2E3A2D20
```

```
0CFA 00              DB    'RAM Start Address .....:= ',0
0CFB 2A290C          LD    HL,(CURSOR) ;HL = CURSOR LOCN
0CFE 22290C   ERR10  LD    (CURSOR),HL ;RESET CURSOR
0D01 E5              PUSH  HL          ;KEEP CURSOR LOCN
0D02 DF              RST   NASSYS
0D03 63              DB    INLIN       ;GET  REPLY
0D04 D1              POP   DE          ;DECODE REPLY
0D05 E5              PUSH  HL          ;NUM  MODS HL
0D06 DF              RST   NASSYS
0D07 64              DB    NUM         ;SORT OUT HEX NUMBER
0D08 E1              POP   HL          ;RESTORE HL
0D09 38F3            JR    C,ERR10     ;ERROR??
0D0B 2A210C          LD    HL,(NUMV)   ;HL = NUMBER
0D0E 22350F          LD    (RAMPOS),HL ;KEEP REPLY
0D0E                                   ;
0D0E                                   ;MAIN LOOP FOR LOADING FILES
0D0E                                   ;
0D11 CD6B0E   LOOP   CALL  HEAD2
0D14 DF              RST   NASSYS
0D15 7B              DB    BLINK
0D16 FE1B            CP    ESC
0D18 CA1C0E          JP    Z,DSKEND    ;STOP
0D1B 3EFF            LD    A,#FF
0D1D 3201C0          LD    (DDRV),A    ;FORCE DIRECTORY READ
0D20 0E00            LD    C,0         ;DIRECTORY DRIVE 0
0D22 DF              RST   NASSYS
0D23 83              DB    ZRDIR       ;READ DIRECTORY
0D24 20EB            JR    NZ,LOOP     ;ERROR MAYBE NO DISK
0D26 2155C0          LD    HL,S1FCB    ;HL = A(FCB)
0D29 0630            LD    B,#30       ;LOAD & LOCKED FILES
0D2B DF              RST   NASSYS
0D2C 86              DB    ZLOOK
0D2D 2829            JR    Z,LOADIT
0D2F EF              RST   PRS
0D30 0D0D            DB    CR,CR
0D32 20202020
0D36 2020203C
0D3A 3D3D3D3D
0D3E 2046494C
0D42 45204E4F
0D46 5420464F
0D4A 554E4420
0D4E 3D3D3D3D
0D52 3E00            DB    '       <==== FILE NOT FOUND ====>',0
0D54 DF              RST   NASSYS      ;WAIT TO SHOW RESULT
0D55 5D              DB    TDEL
0D56 18B9            JR    LOOP
0D56                                   ;
0D56                                   ;
0D56                                   ;ALL  OK LOAD DISK
0D56                                   ;SHOW ATTRIBUTES FOR CHECK
0D56                                   ;
0D58 CD230E   LOADIT CALL  HEAD        ;LOAD DISK
0D5B 2A61C0          LD    HL,(FSEC)   ;HL = SECTOR LOCATION
```



```
0D5E EF              RST   PRS
0D5F 20202020
0D63 20202044
0D67 69736B20
0D6B 53656374
0D6F 6F72202E
0D73 2E2E2E2E
0D77 2E2E2E3A
0D7B 3D2000          DB    '       Disk Sector .....:= ',0
0D7E DF              RST   NASSYS
0D7F 66              DB    TBCD3
0D80 2A350F          LD    HL,(RAMPOS) ;HL = RAM POSN-
0D83 EF              RST   PRS
0D84 0D0D            DB    CR,CR
0D86 20202020
0D8A 20202046
0D8E 726F6D20
0D92 52616D20
0D96 2E2E2E2E
0D9A 2E2E2E2E
0D9E 2E2E2E3A
0DA2 3D2000          DB    '       From RAM ..........:= ',0
0DA5 DF              RST   NASSYS
0DA6 66              DB    TBCD3
0DA7 3A63C0          LD    A,(FNSC)    ;A  = NUMBER OF SECTORS
0DAA 47              LD    B,A         ;B  = NUMBER OF SECTORS
0DAB EF              RST   PRS
0DAC 0D0D            DB    CR,CR
0DAE 20202020
0DB2 2020204E
0DB6 756D6265
0DBA 72206F66
0DBE 20536563
0DC2 746F7273
0DC6 202E2E3A
0DCA 3D202020
0DCE 00              DB    '       Number of Sectors ..:=   ',0
0DCF 78              LD    A,B         ;A  = NUMBER OF SECTORS
0DD0 DF              RST   NASSYS
0DD1 68              DB    B2HEX
0DD2 C5              PUSH  BC          ;WAIT TO SHOW
0DD3 DF              RST   NASSYS
0DD4 5D              DB    TDEL
0DD5 C1              POP   BC          ;RECOVER BC
0DD6 0E00            LD    C,0         ;C  = DISK DRIVE
0DD8 ED5B61C0        LD    DE,(FSEC)   ;DE = A(SECTOR)
0DDC 2A350F          LD    HL,(RAMPOS) ;HL = A(RAM)
0DDF DF              RST   NASSYS
0DE0 82              DB    ZDWR        ;WRITE IT OUT
0DE1 2003            JR    NZ,ERRORD
0DE3 C3110D          JP    LOOP        ;NEXT
0DE6 F5       ERRORD PUSH  AF          ;KEEP ERROR NUMBER
0DE7 EF              RST   PRS
```

```
0DE8 0D0D4572
0DEC 726F7220
0DF0 3D3D3D3E
0DF4 2000           DB    CR,CR,"Error ===> ",00
0DF6 F1             POP   AF
0DF7 DF             RST   NASSYS
0DF8 88             DB    ZCOV          ;LOAD ERROR OVERLAY
0DF9 456D7367       DB    "Emsg"        ;PRINT MESSAGE
0DFD EF             RST   PRS
0DFE 0D0D           DB    CR,CR
0E00 20202020
0E04 20202050
0E08 72657373
0E0C 2022456E
0E10 74657222
0E14 202000         DB          Press "Enter"  ',0
0E17 DF             RST   NASSYS
0E18 7B             DB    BLINK
0E19 C3110D         JP    LOOP
0E1C 3EFF     DSKEND LD   A,#FF         ;FORCE DIR READ
0E1E 3201C0         LD    (DDRV),A
0E21 DF             RST   NASSYS        ;END
0E22 5B             DB    RETNAS        ;
0E23 EF       HEAD  RST   PRS           ;--- SUBROUTINE HEAD ---
0E24 0C             DB    CLEAR
0E25 20202020
0E29 20202020
0E2D 20202020
0E31 20444953
0E35 4B205550
0E39 44415445
0E3D 20555449
0E41 4C495459       DB                    DISK UPDATE UTILITY'
0E45 0D             DB    CR
0E46 20202020
0E4A 20202020
0E4E 20202020
0E52 202D2D2D
0E56 2D2D2D2D
0E5A 2D2D2D2D
0E5E 2D2D2D2D
0E62 2D2D2D2D       DB                    ----------'
0E66 0D0D0D00       DB    CR,CR,CR,0
0E6A C9             RET                 ;
0E6B CD230E   HEAD2 CALL  HEAD          ;--- SUBROUTINE HEAD2 ---
0E6E EF             RST   PRS
0E6F 20202020
0E73 20202046
0E77 696C6520
0E7B 4E616D65
0E7F 202E2E2E
0E83 2E2E2E2E
0E87 2E2E2E3A
```



```
0E8B 2D2000         DB          File Name ..........:- ',0
0E8E 2155C0         LD    HL,S1FCB
0E91 0608           LD    B,8
0E93 7E       HEAD10 LD   A,(HL)        ;PRINT FILE NAME
0E94 F7             RST   ROUT
0E95 23             INC   HL            ;INDEX ALONG
0E96 10FB           DJNZ  HEAD10
0E98 EF             RST   PRS
0E99 2E00           DB    '.',0
0E9B 0602           LD    B,2
0E9D 7E       HEAD20 LD   A,(HL)        ;PRINT FILE EXT
0E9E F7             RST   ROUT
0E9F 23             INC   HL
0EA0 10FB           DJNZ  HEAD20
0EA2 EF             RST   PRS
0EA3 0D0D           DB    CR,CR
0EA5 20202020
0EA9 2020204C
0EAD 6F616465
0EB1 64204672
0EB5 6F6D2052
0EB9 414D202E
0EBD 2E2E2E3A
0EC1 2D2000         DB    '       Loaded From RAM ....:- ',0
0EC4 2A350F         LD    HL,(RAMPOS)
0EC7 DF             RST   NASSYS
0EC8 66             DB    TBCD3         ;PRINT IT
0EC9 EF             RST   PRS
0ECA 0D0D           DB    CR,CR
0ECC 20202020
0ED0 20202049
0ED4 6E736572
0ED8 74204469
0EDC 736B2069
0EE0 6E746F20
0EE4 44726976
0EE8 6520302E       DB    '       Insert Disk into Drive 0.'
0EEC 0D             DB    CR
0EED 20202020
0EF1 20202050
0EF5 72657373
0EF9 2022456E
0EFD 74657222
0E01 20776865
0E05 6E207265
0E09 6164792E       DB    '       Press "Enter" when ready.'
0F0D 0D             DB    CR
0F0E 20202020
0F12 20202050
0F16 72657373
0F1A 20224573
0F1E 63222020
0F22 20746F20
```

```
Z2 ASSEMBLY LISTING ..... PAGE  7
-----------------------------------

  0F26 5465726D
  0F2A 696E6174
  0F2E 652E3A2D
  0F32 20              DB    '       Press "Esc"   to Terminate.:- "
  0F33 00              DB    0
  0F34 C9              RET
  0F35 0100      RAMPOS DW    1
```

```
Z2 ASSEMBLY LISTING ..... PAGE  8
-----------------------------------

  B2HEX   0068 BLINK  007B CLEAR  000C CR      000D
  CRT     0065 CURSOR 0C29 DDRV   C001 DSKEND 0E1C
  ERR10   0CFE ERRORD 0DE6 ESC    001B FEXA    C067
  FEXT    C05D FLDA   C065 FNAM   C055 FNSC    C063
  FSEC    C061 FSFL   C05F FUFL   C060 HEAD    0E23
  HEAD10  0E93 HEAD2  0E6B HEAD20 0E9D INLIN   0063
  LINE1   080A LOADIT 0D58 LOOP   0D11 NASSYS  0018
  NUM     0064 NUMV   0C21 NXTSEC C414 PRS     0028
  RAMPOS  0F35 RETNAS 005B ROUT   0030 S1FCB   C055
  S2FCB   C069 TBCD3  0066 TDEL   005D UPDATE  0C80
  ZCBRK   008B ZCFMA  008C ZCFS   0085 ZCKER   008A
  ZCOV    0088 ZCOVR  0089 ZDRD   0081 ZDSIZE  0080
  ZDWR    0082 ZENTER 0087 ZINLIN 0063 ZLOOK   0086
  ZRDIR   0083 ZWDIR  0084
```

```
DUMP V1.0       UPDATE V1.0                                    PAGE 01

0C80  CD 23 0E EF 44 69 73 6B  20 55 70 64 61 74 65 20  .#..Disk Update
0C90  46 69 6C 65 20 4E 61 6D  65 20 2E 2E 2E 3A 2D 20  File Name ...:-
0CA0  00 DF 63 21 1C 00 19 11  55 C0 01 08 00 ED B0 EF  ..c!....U.......
0CB0  0D 46 69 6C 65 20 45 78  74 65 6E 73 69 6F 6E 20  .File Extension
0CC0  2E 2E 2E 2E 2E 2E 2E 2E  2E 2E 3A 2D 20 00 DF 63  ..........:- ..c
0CD0  21 1C 00 19 11 5D C0 01  02 00 ED B0 EF 0D 52 41  !....]........RA
0CE0  4D 20 53 74 61 72 74 20  41 64 64 72 65 73 73 20  M Start Address
0CF0  2E 2E 2E 2E 2E 2E 2E 3A  2D 20 00 2A 29 0C 22 29  .......:- .*).")
0D00  0C E5 DF 63 D1 E5 DF 64  E1 38 F3 2A 21 0C 22 35  ...c...d.8.*!."5
0D10  0F CD 6B 0E DF 7B FE 1B  CA 1C 0E 3E FF 32 01 C0  ..k..{.....>.2..
0D20  0E 00 DF 83 20 EB 21 55  C0 06 30 DF 86 28 29 EF  .... .!U..0..().
0D30  0D 0D 20 20 20 20 20 20  20 3C 3D 3D 3D 3D 20 46  ..       <==== F
0D40  49 4C 45 20 4E 4F 54 20  46 4F 55 4E 44 20 3D 3D  ILE NOT FOUND ==
0D50  3D 3D 3E 00 DF 5D 18 B9  CD 23 0E 2A 61 C0 EF 20  ==>..]...#.*a..
0D60  20 20 20 20 20 20 44 69  73 6B 20 53 65 63 74 6F        Disk Secto
0D70  72 20 2E 2E 2E 2E 2E 2E  2E 2E 3A 3D 20 00 DF 66  r .........:= ..f
0D80  2A 35 0F EF 0D 0D 20 20  20 20 20 20 20 46 72 6F  *5....        Fro
0D90  6D 20 52 61 6D 20 2E 2E  2E 2E 2E 2E 2E 2E 2E 2E  m Ram ..........
0DA0  2E 3A 3D 20 00 DF 66 3A  63 C0 47 EF 0D 0D 20 20  .:= ..f:c.G...
0DB0  20 20 20 20 20 4E 75 6D  62 65 72 20 6F 66 20 53       Number of S
0DC0  65 63 74 6F 72 73 20 2E  2E 3A 3D 20 20 20 00 78  ectors ..:=    .x
0DD0  DF 68 C5 DF 5D C1 0E 00  ED 5B 61 C0 2A 35 0F DF  .h..]....[a.*5..
0DE0  82 20 03 C3 11 0D F5 EF  0D 0D 45 72 72 6F 72 20  . .........Error
0DF0  3D 3D 3D 3E 20 00 F1 DF  88 45 6D 73 67 EF 0D 0D  ===> ....Emsg...
0E00  20 20 20 20 20 20 20 50  72 65 73 73 20 22 45 6E         Press "En
0E10  74 65 72 22 20 20 00 DF  7B C3 11 0D 3E FF 32 01  ter"  ..{...>.2.
0E20  C0 DF 5B EF 0C 20 20 20  20 20 20 20 20 20 20 20  ..[..
0E30  20 20 44 49 53 4B 20 55  50 44 41 54 45 20 55 54    DISK UPDATE UT
0E40  49 4C 49 54 59 0D 20 20  20 20 20 20 20 20 20 20  ILITY.
0E50  20 20 20 2D 2D 2D 2D 2D  2D 2D 2D 2D 2D 2D 2D 2D     ----------
0E60  2D 2D 2D 2D 2D 2D 0D 0D  0D 00 C9 CD 23 0E EF 20  ------.....#..
0E70  20 20 20 20 20 20 46 69  6C 65 20 4E 61 6D 65 20        File Name
0E80  2E 2E 2E 2E 2E 2E 2E 2E  2E 2E 3A 2D 20 00 21 55  ..........:- .!U
0E90  C0 06 08 7E F7 23 10 FB  EF 2E 00 06 02 7E F7 23  ...~.#.....~.#
0EA0  10 FB EF 0D 0D 20 20 20  20 20 20 20 20 4C 6F 61 64  .....        Load
0EB0  65 64 20 46 72 6F 6D 20  52 41 4D 20 2E 2E 2E 2E  ed From RAM ....
0EC0  3A 2D 20 00 2A 35 0F DF  66 EF 0D 0D 20 20 20 20  :- .*5..f...
0ED0  20 20 20 49 6E 73 65 72  74 20 44 69 73 6B 20 69     Insert Disk i
0EE0  6E 74 6F 20 44 72 69 76  65 20 30 2E 0D 20 20 20  nto Drive 0..
0EF0  20 20 20 20 50 72 65 73  73 20 22 45 6E 74 65 72      Press "Enter
0F00  22 20 77 68 65 6E 20 72  65 61 64 79 2E 0D 20 20  " when ready..
0F10  20 20 20 20 20 50 72 65  73 73 20 22 45 73 63 22       Press "Esc"
0F20  20 20 20 74 6F 20 54 65  72 6D 69 6E 61 74 65 2E     to Terminate.
0F30  3A 2D 20 00 C9 01 00 00  00 00 00 00 00 00 00 00  :- ............
0F40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0F50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0F60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0F70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

REVIEW OF COMPASS, ZAP, AND RAVEN ASSEMBLERS

By Stephen Weir

COMPASS (not to be confused with COMPAS Pascal), ZAP and RAVEN are tape based assemblers for use with the Nascom range of microcomputers.

COMPASS

COMPASS stands for COMPression ASSembler and is written by Level 9, the same people who write those excellent adventures. As you would expect therefore, COMPASS is a good quality piece of software with no bugs. It comes on a TDK cassette with a neatly printed label together with a user manual bound in the usual "Level 9 blue". On the tape there is the assembler itself within a relocator program followed by a program to convert ZEAP files to COMPASS files should you wish to change over to COMPASS from ZEAP but still use your ZEAP files. Finally, there is the source for the ZEAP convertor so that you can see how it works and also use it to try out the editing facilities.

The relocator is very useful as you can relocate COMPASS to the top of your RAM and then you have the rest of RAM from 1000h upwards free. Once relocated, COMPASS takes up about 7K. On cold start it is necessary to specify start addresses for both source file and object code (this is a default value only and can be changed at any time by an ORG directive in the source). It is also possible to specify the symbol table and workspace area. So, providing you have enough memory it should be possible to arrange things so that you can always keep the required area of memory free for the object code.

When you are testing the assembled program while the assembler is still in RAM it is often the case that the program will not do as intended and may corrupt part of the assembler. On return to COMPASS , however, a checksum routine is carried out so that if COMPASS has been corrupted a message is displayed. It is then necessary to reload COMPASS.

As I have already mentioned, COMPASS compresses the source code. It does this by using a 1-byte code for each keyword. Assembly speed is good and Level 9 claim 3000 lines per minute, but since I can't count that fast, I'll just take their word for it! During assembly, any errors are displayed on the listing and assembly is resumed (unlike ZAP80 and RAVEN which abort assembly at the first error).

Lines are entered with line numbers and editing is carried out using the NAS-SYS screen-editing. The line-numbering system adopted by Level 9 is a little strange to say the least. Firstly, numbers are in HEX and they appear on the far left of the assembly listing next to the address field. Two columns of Hex numbers make reading a little confusing. Secondly, it is not possible to enter lines in increments other than one but it is possible to insert lines using the Insert command, which also automatically numbers the inserted lines and renumbers all succeeding lines.

Commands are entered using single characters and include listing (number of lines listed at a time may be set), string search, assemble (with listing options), and tape read, write and verify which merely use the NAS-SYS routines and therefore do not allow file names. NAS-SYS restart instructions and subroutine calls are not supported. For listing to a printer the NAS-SYS 'X' or 'U' commands have to be used.

The manual is mainly just a guide to the facilities available and does not attempt to teach the use of an assembler. This is also true of the other two manuals. However, there is some useful information given which should allow you to add extra commands to COMPASS.

ZAP

I ordered my copy of ZAP from an old copy of 80-BUS NEWS where the price quoted was #15. When it arrived, I was pleasantly surprised to find a refund of #8.50 since the price had since been reduced to #6.50. It comes on a good quality BASF tape and the manual is just 11 A4 pages stapled together. ZAP takes up about 7K and also compresses the source code, so a minimum of 16K RAM is sufficient. On cold starting you are greeted with a cheery message and a report on the amount of free memory.

The editing facilities are the worst feature of this assembler and are similar to what you find on most BASICs of a few years vintage. In particular there is no pause control on listing to the screen, so you can only list in blocks of 14 lines. Also, there is no string search/change command which, together with the poor listing facilities, makes editing rather tedious. Once you have suceeded in getting the required line on the screen it can be edited using the NAS-SYS screen-editing commands. All commands must be entered in full (usually four letters). This is far less convenient than single key entry. Filenames are not used when saving the source on tape.

There are a couple of useful commands however, such as OBEY "c" where c can be any NAS-SYS command, so you can use NAS-SYS commands without having to leave the assembler. DUMP will write the assembled object code to tape and if you put a RET instruction at the end of the program, RUN will execute the program and return control to ZAP when it is finished.

ZAP also compresses the source by using 1-byte codes for mnemonics, labels and macros, and by removing all unnecessary spaces. There is no space between the line number and the label and only one space is left between label and mnemonic, and mnemonic and comment. This makes it impossible to lay out the source in neat columns for readability, since the resultant listing is always a scruffy mess!

Conditional assembly is possible by enclosing the source within an IF (expression) . . . FI statement. If the expression is evaluated to true (non-zero) then the code within the block is assembled, otherwise it is ignored and assembly continues from the line following the FI. Macros are also supported - a macro is just a group of instructions which is given a suitable name. When the macro name appears in the source, the assembler inserts the machine code which makes up the macro. A macro would normally be used where a small group of instructions are used many times, but where a subroutine would not be appropriate. For example a macro to push registers at the start of each subroutine. Parameters may be passed to the macro so that the same macro may operate on different data, which may be registers, labels, expressions, etc. In the ZAP assembly listing the macro appears in full with all the mnemonics, so that it is still readable by anyone not familiar with macros. Labels and macros may be listed at any time but if you have deleted any from the source during programming, they still appear in the tables and it is imposible to get rid of them! Assembly is aborted at the first error, however the error messages given are reasonably full and clear.

Multistatement lines may be used with mnemonics separated by colons. The NAS-SYS restart mnemonics are supported (although BRKPT and RDEL are incorrectly represented by BREAK and KDEL respectively). In addition, "for convenience" as the manual puts it, there are alternative mnemonics for some of the more common instructions. e.g. CLA (clear acc) for XOR A, JSR (jump to subroutine) for CALL. I really don't see the point of including alternatives since their use will only lead to lack of standardisation and the Zilog mnemonics are quite logical and clear enough. Another unusual, but more interesting feature is the inclusion of the Z80 unknown opcodes, or most of them anyway.

RAVEN

At £30, RAVEN is the most expensive of the three and about the same price as ZEAP. However, when you see the sort of things it can do, I'm sure that you will agree that it is an excellent piece of software and quite reasonably priced at that. The TDK tape has the assembler itself on one side and the Z80 macro library on the other (see later for an explanation of the macro library). The manual is a loose-leaf ring-binder and has been printed directly by a dot-matrix printer. Unfortunately, the paper used is a bit thin and the holes are punched close to the edge so after a few hours' use the holes tear and you literally have a loose-leaf manual! Raven takes up about 16K of memory and it does not perform any text compression (apart from the use of TABs to save spaces), so 32K RAM is needed even for a fairly small program.

It is possible to interface RAVEN to your own system, in particular a printer routine, optional form feeds, cursor character code and repeat speed, and finally the start address for the source code. The adapted version may then be saved on tape and used as the working copy. The many facilities available on this assembler are well explained with clear examples. The only ambiguity I came across was the procedure for attaching your own printer routine. It didn't work for me so I had to devise my own method.

The editor is one of the best features of RAVEN. It is a full screen editor and I have found it to be far superior to the usual line-based editor that is supplied. The screen is best thought of as a 48 character by 15 line window on the text file, which is moved around the file by the cursor keys. Because any part of the file may be viewed at any time just by using the cursor keys, line numbers are not necessary. Lines can be up to 255 characters long but this is probably too long for most uses. An option of 80 (or 132) characters would be useful since this would be the same as most printers.

There is a comprehensive range of commands, selected by a single letter:- change and insert text, find and replace strings, delete and copy lines or blocks of text. As well as read, write and verifying of named files, it is possible to join a file from tape to one already in memory so that you can build up the source program from library files (for example). Tabs may be positioned anywhere along the 255 character line width and you may have as many as you wish. If you forget the function of a particular key, the Help command will give a short description. However it is usually necessary to consult the manual anyway since most commands have several options which require further input. It was while I was playing around with the help facility that I discovered a command that was not mentioned in the manual, this was to "update tabs". I've been trying it out but it doesn't seem to do anything useful!

The top line (line 16) is used to display information to the user. Cursor position is given by column number (1 to 255) but I would also like to have seen line position so that you would have a better idea of what part of the file you are at. This display may be changed to give the length of the file in bytes. Other information is displayed during tape transfers, and string search/replace etc.

Whereas assemblers for micros usually have one fixed instruction set, RAVEN does not have any instruction set built in at all. Before it can be used as an assembler it has to be supplied with the instruction set for the particular CPU. All instructions and most psuedo-ops are defined as macros. The list of macros defining the instruction set is treated just like any other source file, however for normal use it can be incorporated into the assembler semi-permanently so that it is loaded into RAM along with the assembler. The instruction set supplied is, of course, for the Z80 but you could just as easily replace it with say a 6502 set (although you would have to write this

yourself). RAVEN then becomes a cross-assembler for the 6502. Other uses for this powerful macro facility would be to add extra instructions to the Z80, e.g. LD (HL),HL etc. Or if you are involved in a particular field of interest such as robotics, music or graphics, then you could write your own personal programming language where each program instruction would in fact be a macro name. RAVEN then effectively becomes a compiler for your new language.

One notable ommission from the Z80 library was the NAS-SYS restart instructions. They can be easily added by writing a few extra macros using the methods described very clearly in the manual. The comma in the jump conditional instructions was missing, and as I was used to putting it in, I altered the macros accordingly. Another much more serious error was only discovered after spending many hours puzzling over apparently perfect programs which would just keep on crashing. The problem was that the assembler was allowing relative jumps up to +/- 255 instead of +129 to -126. So, for instance, if the jump was between +130 and +255 the error would not be detected and the byte put into the object code would in fact work out to be a negative jump and the program would therefore crash. I managed to overcome this problem by rewriting the macro definition dealing with relative jumps. Being rather surprised that this problem had not been spotted sooner, I wrote to the author, so he is now aware of it.

At first you might think that there has to be a macro definition for every possible permutation of instruction (about 750 I think), but RAVEN provides facilities for much more concise definitions so that each instruction "type" (e.g. 8-bit register load) may be defined by one macro. The parameters supplied to the macro determine the machine code generated. It is necesary to read the manual to fully understand the method, but I will give an example which may give some idea. e.g. the macro to define 8-bit register to register loads:

```
DEFMAC ("LD*,*",R8,R8)
       DB 40H + #0*8 + #1
END.
```

"LD , " is the name of the macro. The asterixes show the position that the parameters must be in the macro name (separated by a comma in this case). R8 stipulates that the parameters must belong to a previously defined set, R8, which is a set of the 8-bit register names A to L and A is assigned a value of 7, B a value of 0 etc. DB is the define byte pseudo-op. #0 and #1 refer to the first and second parameters respectively. The expression thus generates the required bit pattern according to the registers specified (see the Z80 Technical Manual for a breakdown of the bit patterns). e.g. for LD A,B the expression is evaluated like so:

```
                40H:     01000000
(Reg A = 7)  #0*8:     00111000
(Reg B = 0)  #1 :      00000000
                         --------
             Total     01111000 = 78H
```

There are also several PASCAL-like control structures for the generation of loops within macros. These may be used for the production of data tables. e.g. a macro to clear a block of memory of any size:

```
DEFMAC ("CLEAR * BYTES", NUM)    ;NUM is a set of integers
                                 ; 0 - 65535
        COUNTER = #0             ;Loop counter set to
                                 ;number of bytes
        $WHILE COUNTER > 0
        DB 0
        COUNTER = COUNTER - 1
        END
END.
```

So the macro call CLEAR 24 BYTES will set the next 24 bytes to zero. Notice that there may be spaces within the macro name thus increasing legibility. Other constructs are IF THEN .. ELSE , and $REPEAT .. UNTIL. Expression handling is comprehensive with 18 operators and 4 number bases to work with.

As well as the usual assembler options you can also specify user defined options for such things as conditional assembly. i.e. the value of the boolean argument in an IF .. THEN block may be set by an option at assembly time. Assembly stops at the first error with a single word error message which is not always very helpful. RAVEN is rather slower than the other two assemblers but this is to be expected due to its design. I had the idea to add all the EQUATES that I use (e.g. port addresses, video ram addresses, keyboard control codes) to the permanent macro library to save having to put any in at the beginning of a program. While this worked perfectly OK, the macro library was by now so large that assembly time was painfully long. But I am using a 2 MHz NASCOM 1 so the speed is obviously half of what it could be.

Conclusion

So there we are, three more assemblers for the NASCOM. It would be a little unfair to say which is best since prices vary. Each has its good points - COMPASS is very fast, ZAP offers a macro/conditional assembler at a very low price, and RAVEN can be used very much depending on your own imagination.

The three assemblers are supplied by:

COMPASS	v1.3	Level 9 Computing 229 Hughendon Rd High Wycombe Bucks. HP13 5PG	£12.00
ZAP	v1.6	Syrtis Software 23 Quantock Rd Bridgewater Somerset TA6 7EG	£6.50
RAVEN		P. Harvey 30 Jericho St Oxford OX2 6BU	£30.00

RANDOM RUMOURS (& TRUTHS?) — By S. Monger

Ah well, I've never claimed to be perfect, have I? In the last issue I mentioned the Gemini GM886 board containing an iAPX186. Well it seems that this idea was dropped a while ago, and in fact it is the GM888 that you will see in coming months. What is the difference - well the GM886 board was to be a board interfaced to the 80-BUS via a couple of I/O ports (like the GM812 IVC, GM832 SVC and IO828 Pluto). It contained its own 256K RAM and consequently would have been very fast, and expensive! It would also have had the disadvantage of not being able to drive any I/O peripherals directly, such as Pluto, and would have to have passed on the task via the Z80 on the host processor board, consequently slowing down the operation quite considerably. After a little research Gemini discovered that they could produce a board that would drive the 80-BUS directly, and thus be able to use all the I/O on the bus. More importantly, from a cost viewpoint, the board would be able to have NO memory on it, and would therefore almost certainly be under #200. Of course in interfacing to the 80-BUS the processor used has had to be changed to one with an external 8-bit interface, and so the 8MHz 8088 has been chosen.

So how do you use it? Well the system boots as normal under the Z80, you then flip a bit of an 80-BUS I/O port, the 8088 puts out a bus request, the Z80 finishes the instruction that it was executing, and then the 8088 takes over control of the entire system. If at any stage you wish to pass control back to the Z80 then the 8088 flips the I/O bit back, and the Z80 carries on where it left off. In theory, therefore, you can write code that is of mixed type, and switch to and fro between processors - must be of some use!

The GM888 board will also contain another of those blasted Real Time Clock thingies, with battery back-up. This serves two purposes. First of all certain 16 bit operating systems allow time and date stamping of files, and secondly the actual chip chosen provides oodles of interrupts if required, and this is necessary for task switching with Concurrent-DOS (previously called Concurrent-CP/M) and presumably with Multi-tasking MS-DOS. There is also a socket for the 8087 high speed arithmetic co-processor. Board availability? Don't know!!

And how about the extra memory that the 8088 can support? Well Gemini have launched a 256K RAM board that supports the 80-BUS extended address lines. It has several modes of operation:

1) 4 pages of 64K from address 0 (for use with Nascom 2s and Gemini GM811s).
2) 4 pages of 64K from any extended address (EA).
3) 1 page (any) of 256K from any extended address.
4) 3 pages of 64K from 0 and 1 page of 64K from EA1 (allows GM813 users to have 4 pages of 64K at 0 without contention.

In addition to the above there is a 'Common Area' mode which puts a common area of memory (selectable between 4K and 8K) in all four pages.

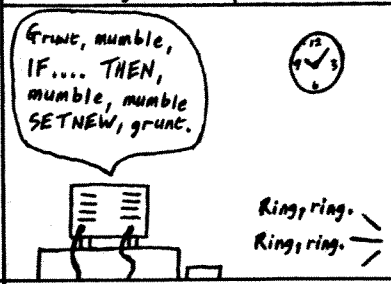
With all these modes it is possible to set up virtually any permutation of page mode and extended addressing that any one may want, and systems with the GM813 or GM888 will accept up to 8 boards = 2Mbytes! (4 pages of 512K.) And the price of all this flexibility ? - #325 + VAT. Available now!!

World Domination!

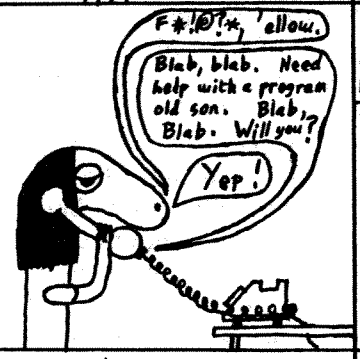
With a distinct lack of signs of activity from Lucas, Gemini continues its 80-BUS domination campaign! The Climax vector graphics colour board has been taken over by Gemini, as the GM837. It is now only available in its fully populated form (modulated and R-G-B outputs) and the price has been reduced to #165 + VAT. Gemini has also taken over the IO Research 8-bit A-D board, now the GM824, and the price remains the same at #125 + VAT. One 80-BUS price has risen, the Belectra arithmetic board. It is now a totally horrendous #268 + VAT!!

Laurence lends a helping hand.

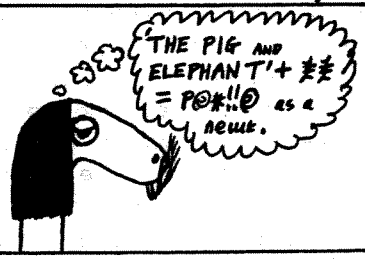
Laurence's train of thought is derailed by the telephone.



In the interest of good taste, this section of profane verbalage has been censored by the editor.



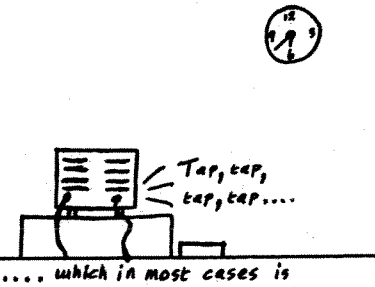
Laurence decides to lumber Heloise Fortran with the problem while he attends to more important things....

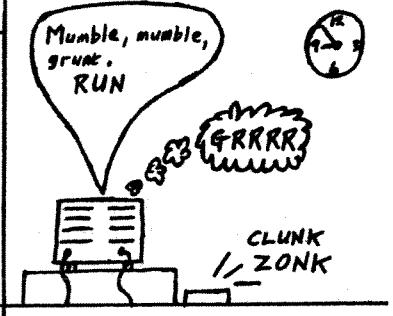


.... But.

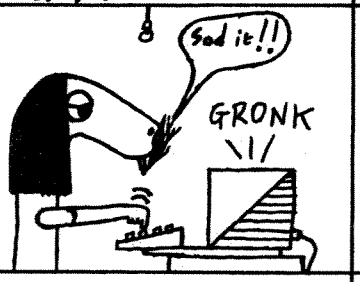
Like all good programmers, our hero thinks the problem through first....

.... then starts writing the program.

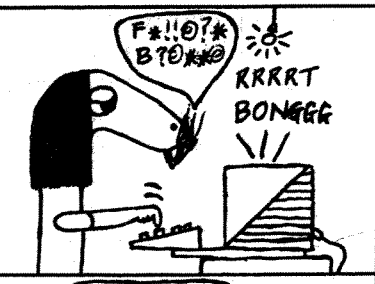




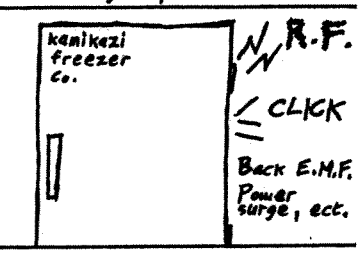
Part of the fun of computing is de-bugging your software....



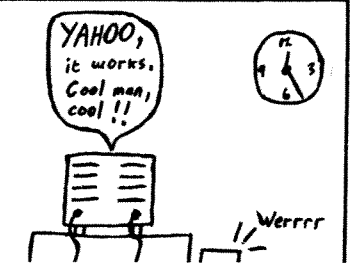
.... which in most cases is easier said than done!

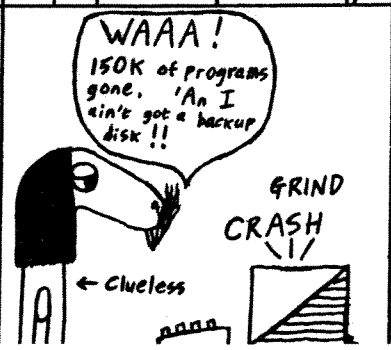


Meanwhile in the kitchen, the refrigerator's thermostat decides it's time to go 'ape'.



Laurence is unaware of the mains borne mayhem heading his way.





It's always a good idea to make backup copies of your disks in case you get caught out like Laurence.

By D.G. Richards.
Tonyrefail. Mid. Glamorgan.
1983.

ADS

Ram-A board, 32k RAM, 2716 conversion (no EPROMs). With INMC mods, but still won't work at 4MHz without waits (dunno why...) #45 ono. HS1N twin-drive digital cassette system. As reviewed 80-Bus News Vol 1 iss. 4. With original ROMs (2708s, assembled for #D000 and ports #F8 to #FF) or my enhanced OS (2716 - assembled for #D000, and ports #78 to #7F); 4118s, connecting leads and some tapes. #170 ono. Adrian Perkins. Tel. Bracknell (0344) 485816

Nascom 1, port select corrected, card frame, buffer PCB, 32K RAM fully debugged, PSU, 4800 baud cassette interface, TTY interface, keyboard, Harris terminal with 12" Motorola VDU, TTY printer, BASIC & assembler on tape. All documentation. #140 the lot or prepared to split. Jim Taylor Boldon 36215.

Floppy disk system. GM815/2, GM809 etc. Twin Pertec drives (350K per drive) in cabinet with power supply. Double density controller card, CP/MZ 2.2 operating system, and lots of software including Pascal, BASIC and C languages, Utilities, Adventure, Games and Business software. Totally reliable system. #450. Also, twin single sided drives in cabinet with power supply and Henelec Singler Density Controller. Including all above software and Vers. 2.2 operating system. Haggle around #275. Call John on 01-380-0191 between 10am and 7.30pm.

TRANSDATA MODEM CARDS.

Brand new, tested and aligned 300 BAUD answer and originate acoustic telephone modem cards manufactured by TRANSDATA for their model 317 acoustic modem. Full CCITT specs. Requires only a few components to complete (no case available). Computer interface is 300 BAUD RS232 I/O. The power supplies of +12 volts at 180mA and -12 volts at 170mA are required. Card size 350mm x 105mm. May be directly connected using the GEC LTU 11 coupler (see below). Full circuit description, drawings and connection data supplied. Suitable for use with computers fitted with 300 BAUD RS232 I/O. Software for Nascom & Gemini published in the last issue.

Card only at 29.95 inc. VAT. (26.04 + VAT)
Supplied with LEDs connector, switches and transducers at 34.95 inc. VAT. (30.39 + VAT)
Supplied with LEDs connector, switches and GEC LTU 11 directcoupler at 45.95 inc.VAT.(39.96+ VAT)
Carriage & packing 1.00

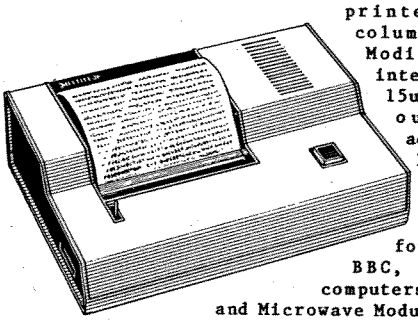
BIG BIG BIG POWER SUPPLIES

Ex-equipment GOULD power supplies for those who need a lot of power.

Type 1 +5 volts at 20 amps. 7" x 4" x 3.5" o.a.
Switch mode. 220-240 V AC in.
39.95 inc. VAT. (34.73 + VAT)
Carriage & Packing 2.00

Type 2 +12 volts at 4 amps, +5 volts at 40 amps
-5 volts at 1 amp, -12 volts at 1 amp
14.5 x 6" x 3.5" o.a. (including fan)
Thyristor switch mode. 220-240 V AC in
79.95 inc. VAT. (69.52 + VAT)
Carriage & packing 4.00

MULTITECH MPF-II PRINTER



The Multitech MPF-II thermal matrix line printer features 40 column print width. Modified Centronics interface (requires 15uS strobe or use our interface adaptor). 150 - 180 lines per minute. Bidirectional 7 x 10 matrix. 4.4" paper. Suitable for Nascom, Gemini, BBC, Dragon, Tandy computers, etc, and Tona and Microwave Modules RTTY readers. Unrepeatable value at 49.95 inc VAT. (40.43 + VAT.)

Interface adaptor with leads 14.95 inc. VAT (13.00 + VAT). Paper 2.95 per roll inc. VAT. UK Carriage & packing 1.00

GEC LTU 11 Mk II PRESTEL MODEM CARD

Manufactured for the GEC stand alone PRESTEL unit, computer interface is TTL logic level at 75 BAUD transmit and 1200 BAUD receive. (Suitable for use with most computers fitted with speed selectable UARTs or twin independant serial I/O at TTL levels.) Output level is at -13dBm 600 ohms, input at -36dBm, for connection to the GEC LTU 11 direct coupler. Power requirement +5 volts at approx. 250 mA. Designed for direct connection to the GEC LTU 11 direct coupler, to provide line connect and autodial facilities. Card size 266mm x 140mm. Supplied with full circuit description and connection data.

Card only at 14.95 inc. VAT. (13.00 + VAT)
Supplied with GEC LTU 11 direct coupler at 27.95 inc.VAT. (24.30 + VAT)
Carriage & packing 1.00

THE 69SD5 HALL EFFECT KEYBOARD

Compact, 64 key + 5 function keys, Hall effect keyboard with reprogrammable (2716) ASCII output decoder EPROM. Steel key frame for good rigidity. Negative going strobe. Requires +5 volt and -12 volt supplies. 29.95 inc. VAT (26.04 + VAT). Carriage & Packing 1.00.

STC NOVATEL PRESTEL TERMINAL

50 ONLY AT THIS PRICE

Features
7" diag. green screen, 240V AC mains operated, detachable key pad, robust case 14" x 12" x 7½" cassette recorder data store facility, video output for other monitors, security lock, etc.

Ideal for travel agents, offices, stock market, educational, home and all Prestel information!

Fully tested 'as new' and guaranteed
£149.95 inc. VAT
(£130.39+VAT) (UK C/P + ins. £3.55)

ORDER BY POST OR 'PHONE OR CALL IN AND SEE FOR YOURSELF

GEC LTU 11 DIRECT TELEPHONE COUPLER

Telephone direct coupler, with isolation transformer to the rigorous 1978 (5KV) spec. (not the more recent, 2KV, spec.). Relays are provided for line select and autodial. Power supply (for line control relays) +5 volts at 200 mA. TTL logic inputs for the line control, tone input to the unit at -13dBm, output at -35dBm. Supplied in a plastic case 195mm x 108mm x 65mm complete with PO type 96 5-pole plug on lead. Suitable for use with the GEC LTU 11 Mk II and the TRANSDATA modem cards. Full connection details and circuits are supplied.

Supplied complete at 14.95 inc. VAT. (13.00 + VAT)
Carriage & packing 1.00

ITT COMPUTER CASES

Professional computer case with keyboard cutout. 18" x 15.5" x 4.5" (front slopes). Ideal for single board computers like the Nascom or Gemini Multiboard (3 cards max.) Very heavy guage (.25") plastic with metal base. Attractive silver gray finish. 27.50 inc. VAT. (23.91 + VAT). UK Carriage & Packing 2.10

HENRY'S INCREDIBLE CP/M UTILITIES DISK

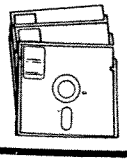
All the things you ever wanted: The disk cataloguing and file dating suites. File compare, string and byte search utilities, ASCII file compression suite, system independant disk repair utilities and all sorts of other goodies. Most are true CP/M system independant utilities and will work with any CP/M system. Supplied on most popular 5.25" formats (please state when ordering) at 17.25 inc. VAT.
Carriage & packing 50p

CCPZ

Replaces the CCP in CP/M and provides hierarchical file searches through user areas (invaluable when using more than one user area on high capacity drives). New commands for getting and executing files, and lots more. Now available rewritten as an M80 .MAC file using Z80 mnemonics as well as the earlier .ASM file, with two recently discovered bugs fixed. Supplied in most popular 5.25" formats (please state when ordering) at 11.50 inc. VAT.
Carriage & packing 50p

MDIS THE INTELLIGENT DISASSEMBLER

MDIS is a CP/M disassembler useable on most standard CP/M machines, switchable for either 8080 or Z80 mnemonics. Command syntax is designed to be similar to the Microsoft M80 assembler, and redirected input from .DAT files is allowed. The output files to printer or disk may include the disassembled code or in M80 format and output label types are differentiated by different letter prefixes. A powerful package at a price which is a lot less than its competitors. Supplied in most popular 5.25" disk formats (state type when ordering) at 57.50 inc. VAT.
Carriage & packing 50p



ORDER BY POST OR TELEPHONE OR CALL IN AND SEE FOR YOURSELF

HENRY'S COMPUTER SHOP
404/406 Edgware Road, London, W2 1ED
Telephone: 01-402 6822
OPEN 6 DAYS A WEEK

Credit Sales available ask for details.
Official orders welcome.
VISA AMERICAN EXPRESS Buy it with Access
Order by Post with CHEQUES/ACCESS/ VISA or you can telephone your orders.