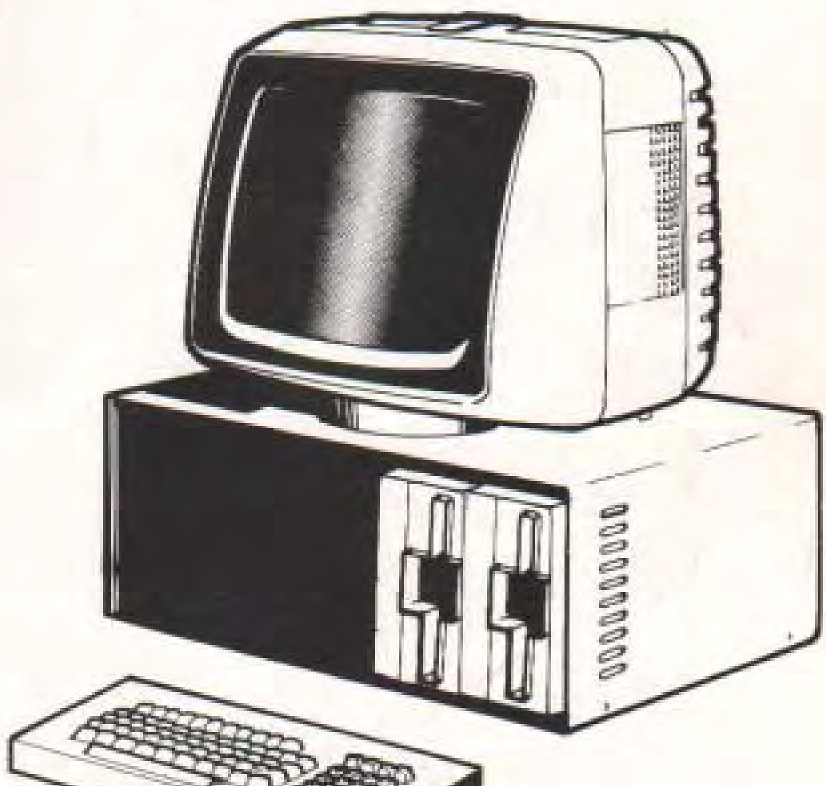
80-BUS NEWS

SEPTEMBER – OCTOBER 1983 VOL.2 ISSUE 5

- USING RANDOM ACCESS FILES
- REVIEWS – ARITHMETIC BOARD
  - 12 K BASIC
  - GEMINI GALAXY 2



£1.50

The Magazine for
NASCOM & GEMINI USERS

September — October 1983 **80-BUS NEWS** Volume 2. Issue 5.

CONTENTS

 The Editor welcomes articles and listings submitted for publication. Material is accepted on an all rights basis unless otherwise agreed. Published by Interface Data Ltd and printed by Excelsior Photoprinting Ltd., High Wycombe.

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,
Interface Data Ltd.,
Woodside Road,
Amersham, Bucks, HP6 5EW.

Editorial and Letters to the Editor

There was no Editorial last issue owing to lack of time (I assumed you would prefer a mag. without my bit, rather than no mag.!), and this time I thought I would restrict myself to answering a couple of letters that raise quite a few interesting points between them.

Expanding Range.

It is nice to see that the range of BUS compatible products is still increasing, although it is now very difficult for the average Hobbyist to keep up with many of the more sophisticated features. I have a hybrid Nascom/Gemini system at home, and I look after two Nascom 2s and a Gemini Galaxy 2 at my place of work.

I look forward to the new EPROM programmer that is soon to be available. Looking into the future, can we expect CP/M 3 to be available, (including a version for N2/Pertec Drives as well)? Will Richard Beal produce even more powerful SYS's to enable us to implement CP/M 3 on a variety of hardware such as Gemini GM833 RAM-DISKs and all the combinations of Keyboards and Disk Drives? A little further ahead, are there any plans for a Z800 CPU card ?

I am very puzzled by the extremely high prices that Gemini are asking for some source listings. As an amateur 'hacker' I have disassembled SIMON for my own use. This took a lot of time and I would have been very happy to have paid £10-£15 for a listing. I think that quite a few other people would feel the same. People who wish to 'rip it off' will either pay the high price, or disassemble it anyway. Can there be any justification for not making CBIOS listings available at a sensible price to the benefit of the author and end user? One of the things that helped make the Nascom system superior was that operating system listings were provided FREE. Many of the Nascom and Gemini users are still enthusiasts rather than 'plastic box' users and they enjoy modifying and adapting. Their enthusiasm for a good product must often favourably affect sales. That enthusiasm will fade if essential information is not easily available to those that need it. It is nice to see that most suppliers of cards still provide a circuit diagram and hardware description, even with ready made cards. Let us hope that this will continue. It would be nice if system software were reasonably priced too.

Yours sincerely, C.Bowden, Truro, Cornwall.

[Ed. - There are quite a few interesting points raised by this letter and so I shall try to comment on them.

EPROM Programmer.

First of all, "What EPROM programmer?" I'd be interested to hear about this as I am not aware that anything is imminent and I do like to keep in touch with things.

CP/M 3.

Well, I do not pretend to be an expert on this (in fact I don't pretend to be an expert on anything!) but I shall make a few statements, if you'll accept that there may be some inaccuracy in them. Firstly, the official name is CP/M Plus. The reason for this is quite simple - CP/M 2 was an upgrade of CP/M 1 and was always intended to be a replacement. CP/M Plus, on the other hand, is intended to be an option to CP/M 2 for those wanting the additional facilities that it provides, and CP/M 2 continues to be available. I have NOT seen any real signs that CP/M Plus is being made available by many manufacturers, and that is probably because of three major reasons - all cost! Firstly you may have noticed that a copy of CP/M 2 goes for about £120 on average, whereas a CP/M Plus will set you back about £260. Secondly, as far as the licensee (i.e. the manufacturer) is concerned he can NOT get any price concession on a CP/M Plus license from Digital Research unless he 'trades-in' his CP/M 2 license. This 'all-or-nothing' situation means selling ONLY CP/M Plus's with his systems (and having to put up his prices to cover it), or having to buy a CP/M Plus license as well (and having to put up his prices to cover it)! Finally, CP/M Plus is available in two versions - one for systems with 64K RAM or less, the other for more than 64K. From what I know the first of these is of little practical benefit, as you don't get the main features that CP/M Plus is all about - cache buffering etc; you do however lose more TPA (available RAM, to you and me!) because of Plus's greater size. With the latter version you get hit by cost again - the system needs additional hardware (e.g. more RAM, plus a Real Time Clock) to support all the extra features, and hence must cost more. I am sure that CP/M Plus has some features that we would all love, but, in my opinion, it either needs Digital Research to alter its pricing philosophy, or YOU, the customer, to say to Nascom, Gemini, or who-ever "Of course I'd buy lots more boards and things, if only I could have CP/M Plus." Personally I don't think it'll catch on - and I very rarely make such statements in print, as I would hate to have my own quotes flung back at me at some future date! There is one possible exception to my opinion - see below.

SYS.

As to the question of CP/M Plus SYSs, my understanding of Plus is that it is constructed in such a way that a well written BIOS can have additional facilities added WITHOUT having to have the BIOS source code, or in fact any other 'tools' than those supplied with Plus. For example, manufacturer X brings out a 1 gigabyte bubble memory board. With it he supplies a disk that contains code that is easily linked by the user to his CP/M Plus, providing instand giga-memory! Too easy to be true? Not according to an emminent Nascom/Gemini systems software writer - not named in case he's wrong!

Z800.

Z800 CPU card? Fantastic idea! One snag! Contact Zilog USA, find someone who knows what's going on, and ask him when there will be real production devices around in anything more than handfuls. If he says 1984 you've got a salesman, not a realist; try again! The Z800 sounds a super device (in fact 'Z800' is a generic term referring to any of the four quite different versions that there will eventually be). For those who haven't heard of the Z800, and without going into any detail, it is a super-charged, turbo-charged, fuel-injected Z80 with cache buffer, extended addressing range, and additional instructions. Further description can wait a few months, maybe even a year! Going back to my comments on CP/M Plus, it is in fact the Z800 that causes me to add the qualifier. The combination could be extremely powerful and fast. But we must dream, and w a i t . . .

Source listings.

I am sure that your comments on pricing will be noted, although the important thing to note is that the listings are available if you want them, even though it's at a price! Since Lucas bought Nascom I don't think that any of the new products have come with listings - for example, the AVC hasn't - see Dave Hunt's review in the last issue, nor has their disk system. Gemini supplies sources of one or two minor utilities and demos, and used to supply BIOS listings. My knowledge of other manufacturers is limited, but I am not aware of any that DOES supply sources, other than MAP80 - and that is currently the subject of High Court proceedings which are being brought against MAP80 Systems by Gemini Microcomputers for alleged copying by MAP80 of large portions of the Gemini BIOS. Selling source listings instead of giving them away obviously doesn't guarantee that they won't still be copied, but the

situation can be much more closely controlled (a condition of sale could be a signed acceptance of copyright terms), and the supplier gets at least some payment for his effort! It is also sad (in certain ways), but relevant, that the 'average' Nascom/Gemini customer is nowadays tending to be an industrial user rather than the 'old style' Nascom hobbyist who used to spend many hours pouring over machine code listings of the Nasbug monitor. Today's 'hobbyist' tends to play games on a Spectrum, with maybe a smattering of BASIC work. But I digress, and would just end up with a little 'thought for the day' (I claim copyright unless anyone can establish 'prior art'!):
"Excellent software costs a lot more to produce than mediocre software, but has one major fault - it costs no more to copy."]

Is he serius?

I would like to congritulat all the contribtures to the first three issues of the news I have seen (it's abusotlay amnasingly funay) and has a rare clarity on technical matters (most of the time) it was worth beying my system for it alone. Is it possible to obtain back coppes of the issues of volume 1 and the odd coppies of INMC?

I would lick to add my name to the handfull of peopel willing to post a disk as suggested by DARK'S DIAR'Y, my veray new and first system consits of the following GM810, GM813, GM812, GM829, GM664, NM I/O complete (excpt for prom SPD/1-L(-H) which Nascom & CIEL failed to supply). GM657, GM555, GM554, GM523, GM808 and a modfied tv. As this is my first computer I have no diy programs so far but I am keen to try to introduce a spelling option for GEMPEN.

I have been soled on the Z80 and 80-BUS for the last two years since the offer of £150 to spend on a project put me on the road excpyt, it took till JULAY 83 to wrase the cash and get moving. Ideas for articules:-

1 CP/M filenames and dirictory for a beginer (whats all the extra stuf on the master disk),
2 using NM I/O board with the GM813,
3 a list of games and programs,
4 an introduction to COMAL-80 (a book suggetion please).
5 instrusions on how best to look after the hardwere espialay the diskes and drive.

Yous sincerlay, David Barrington, Edinburgh.

[Ed. - Yes, I definitely agree that you would find a spelling option for Gempen useful! But if you insist on writing the dictionary for it yourself then please get someone else to check it! However, turning to several points that you have raised (wrased?):-

Back Issues. All 80-BUS magazines are available as back issues - that is Vol. 1 Issues 1-4 and Vol. 2 Issues 1-5 (so far). In addition most of the INMC80 mags. are available as well as a Compilation Issue of 'The Best Of' INMC.

Circle of Iron. Doctor Dark's Circle of Iron is intended to consist of a group of users posting a disk around in a circle, adding their programs to it, and copying any of the ones already on it that they want. This is for **your** programs, NOT **borrowed** ones. The intention is that there will be several circles, for the various disk formats. The whole thing is very bravely being organised by Chris Blackmore, who says that the response so far is nothing short of dismal! No takers for FREE programs? Anyone interested should write to Chris directly at 27 Laburnum Street, Taunton, Somerset, TA1 1LB, saying what disk format you use.

The Above System. Before continuing, I shall describe the system outlined above for those unfamiliar with the list of numbers given. It is a Gemini based system consisting of CPU/RAM, IVC and FDC boards with one 800K Micropolis drive, Nascom I/O board, Gemini extended keyboard, CP/M, Compas 1, Gempen, Gemzap, Gemdebug and Comal-80.

The Article Suggestions.

1. The simple answer is 'read the manuals' where you will find details of all the files on the master disk. Certain ones (FORMAT.COM, BACKUP.COM, CONFIG.COM, READCAS.COM, WRITCAS.COM, SAVEKEYS.COM and SAVEKEYS.MAC) have been supplied by Gemini and will be found in the GM555 manual. The other files are all standard Digital Research CP/M files and are described in the Digital Research Manuals. These latter files will also be covered in many of the CP/M books available.

2. There really isn't much that can be said here. The Nascom I/O board can have up to 3 Z80 PIOs, a CTC and a 6402/8017 UART. Presumably you had some application in mind when buying this board and without knowing this I can only suggest looking at the relevant device manuals. In addition you will find that the CONFIG program provided with your CP/M will allow you to use one of the PIOs as a parallel printer driver instead of the one on the GM813, and will allow you to use the 6402 UART in place of the 8250 on the GM813. This latter possibility is rather pointless as the 8250 has software selectable baud rates, data bits, stop bits, parity etc as well as handshake signals whereas the 6402 has to have its baud rate set by link selection, and other parameters can only be changed by 'hacking' the PCB. One item that you may find of interest is an article in 80-BUS News Vol. 2 Issue 1 on the 'Interrupt System of the Z80'.

3. I am not aware of any specific list other than the one that your dealer will have of the software available from Gemini along with any other items that he gets from elsewhere. In addition you will find adverts in this rag from time to time. Finally, there are of course the computing monthlies, in which you will find all sorts of CP/M software packages. One thing here is to make sure that you purchase the software in the correct format - if this is not possible then a good bet is to choose IBM 3740 single density, single sided, 8" format as this is just about the only 'standard' disk format around, and a number of Gemini dealers can offer a transfer service from this.

4. There are several COMAL books around. Two that I have heard of are 'Structured Programming with COMAL' by Roy Atherton, and 'Beginning COMAL' by Borge Christensen. Both are published by Ellis Horwood, but I have read neither so can comment no further.

5. No particular comments here other than applying some common sense. Always keep the disks in their sleeves when not in use, and preferably in some sort of container away from knockable over cups of coffee. Also avoid magnets. Do not smoke around the drive as the particles are nicely abrasive on those expensive disk heads. Use decent quality diskettes that will not shed their oxide all over the heads - if you do this then you should never need to use a head cleaning kit. Some of these kits do more harm than good and the ones that look like Black and Decker sanding disks shoud definitely be avoided!

Finally, with regard again to Mr Barrington's comment about a spelling checker for Gempen, I hear that there is one of these planned for the new version of Pen, and it will be called SPeLLAID (lower case 'e' deliberate!). Let's howp thet niether Dave Hunt or Mr Barrington hav anithyng too do with the dicshunary for thiss! I hope that these comments have been of use.]

Personally I think that the 8-bit micro has still got a lot of life left in it, and in many applications a switch to a 16-bit processor will bring little or no improvement. However, one area that can show a very marked increase in performance, is in those applications involving a lot of number crunching. The direct mathematical ability of microprocessors is limited, and anything but the simplest calculations have to be emulated in software. The 16-bit processors gain here by working with a larger word size which results in fewer steps to arrive at a result of similar precision. But even in this field the balance can be redressed with the addition of a separate number crunching unit.

The idea of a 'bolt-on' floating point unit to improve performance pre-dates the microprocessor, and in the micro field a few devices have been around for while. The early approach was to add a slave processor, though the current trend seems to be towards a co-processor. ('Slave' and 'Co-' are my own definitions. Other people's usage may be different.)

The co-processor is exemplified by the Intel 8087. This is a numeric processor specifically designed for the Intel 8088/8086 family to carry out high precision integer and floating point instructions. It connects to the data bus in parallel with the CPU and effectively becomes an extension of the CPU. The 8087 has a few direct connections to the CPU, and by monitoring these and the instructions being fetched by the CPU, it can track exactly the instructions being executed. There are certain reserved instructions in the 8086/8 instruction set that are specifically acted upon by the co-processor, while the CPU responds to the same instruction in a supporting role. (e.g. It may go on to fetch an operand from memory so that 8087 can capture it from the data bus.) The co-processor is far more complex than the CPU, and the volume of sales is not so high. As a result this is reflected in the price. A recent price list puts the one-off price of an 8088 at £16.80, with the companion 8087 at £173.30. (N.B. IC prices quoted here are distributors prices, NOT retail prices.)

The slave processor approach is different, although the end result is the same. Here an autonomous processor is built, either from discrete logic (e.g. using a bit-slice approach), or integrated directly on silicon. In this case the arithmetic processor is a peripheral, and not tied to any particular microprocessor. In use it is explicitly passed one or more operands by the master CPU, followed by an instruction of what it is to with them. (e.g. divide one by the other.) The specialist processor then carries out the task, and reports when it is complete, at which point the host CPU can read back the result. For the single chip implemenation there are two types of processor available, low cost or high cost:

Examples of the low-end devices are the National MM57109 maths processor, and the TMS1018. These are basically off-shoots of calculator ICs - i.e. they are cheap, use 4-bit processors, are BCD orientated, and are slow. (The MM57109 will multiply two 8-digit mantissa, 2-digit exponent, numbers together in about 116ms. It is actually quicker to do the calculations directly on a Z80.) At the other end of the spectrum are the Advanced Micro Devices Am9511, Am9512, and the Intel 8231, & 8232. The AMD devices have been with us for about five years, and are custom designed NMOS 16-bit processors whose sole task is to crunch numbers. (e.g. dividing one 32-bit floating point number by another takes under 100uS - over 1000 times faster than the MM57109.) But with this complexity comes cost. A recent distributors one-off price I have for the Am9511 is £90.00, with the Am9512 at £111.38. (Both 2MHz devices.)

The Am9511 and Am9512 are basically the same processor, and the main difference lies in the way that they are microcoded. The 9511 offers 16-bit and 32-bit integer arithmetic, 32-bit floating point arithmetic, and a variety of derived floating point functions (sine, cos, log etc). The 9512 handles floating point numbers only, and lacks the transcendental functions of the 9511, only offering the four basic functions of + - * /. However the floating point numbers can be either single precision (32-bit), or double precision (64-bit), and all calculations are performed to the proposed IEEE floating point standard [1][2]. (I believe that after fitting that lot in to the Am9512, there was no room left for the microcode to handle the other functions.)

Anyway, all this preamble leads into the review of the Belectra HSA-88B Arithmetic Processor board. This 80-BUS board is based on the Am9511A arithmetic processor, and before we launch into the actual review, a summary of the 9511s instruction set can be found in table 1 for those of you who have never met the device.

NOTE: The 9511 is a stack orientated processor. Operands are first pushed onto the internal stack, and then a command is issued to the processor. TOS stands for Top-Of-Stack, and NOS for Next-On-Stack. The 9511 runs from a 2MHz clock, (but there are more expensive versions at 3MHz - £123.75 and 4MHz - £146.25), and the variation of execution times demonstrates the data dependency of the algorithms used.

The Belectra HSA-88B

What you get:

Submitted for this review was:

1 8x3 pcb with 6 ICs on it.

1 7-page manual + AMD Am9511A data sheet.

1 Hisoft Pascal Manual + 1 disc.

The PCB

The 8x3 pcb contains just 6 ICs, including the maths processor. I give Belectra full marks for showing restraint over the board, because it is so easy to look at all the spare space available on an 8x8 card, and to start adding CTCs, PIOs, real-time clocks etc, and before you know it the price of the board has gone up by more than £50 for features that most people could do without. Cosmetically the review card differs from the usual 80-BUS standard. Due to the relative simplicity of the circuit, Belectra have opted to produce a single-sided board. As a result there are about two dozen links on the top of the board. It is silk-screened, but there is no solder resist. The overall colour is blue, and it is fitted with a handle to ease extraction. (Mind you appearance isn't everything, people buy the Ford Sierra and I once owned a Fiat 850.)

The board occupies two I/O ports, which can be selected (via straps) to be 80/81H, 90/91H,....F0/F1H. The board came already strapped for 80/81H. The Hisoft Pascal worked with it at this address, and no indication is given of how to patch the HP5 compiler to support any of the alternative addresses. No circuit diagram is supplied, but the manual does state that DBDR is not implemented, making the card incompatible with Nascom 1s.

The Belctra Manual

The Belectra Manual is a scanty 6.5 pages long, and covers the essential information about the HSA-88B and its use of the Am9511. Two and a half pages of it are taken up by an example assembly language program which computes the squares of 16-bit integers between 0-255. This has been included as a demonstration of the speed of the Am9511A, and they also suggest that you can use it as test program to verify that the board works. For the major programming information on the Am9511A, the manual points you at the Am9511A data (also supplied).

There is just over a page on using the HSA-88B with a high level language. This covers, in a general way, all points I could think of, but I would just like to quote one sentence from it. "Assuming a **reasonably experienced** assembly language programmer, the above procedure is not difficult." (My highlighting.) I would guess the truth of the 'not difficult' in that statement will depend to a large degree on the compiler/Interpreter the programmer decides to tackle.

However the presentation of the manual is poor for a product costing £250+. It has been printed out on a dot-matrix printer, (not a crime in itself), but it is one of those typefaces with no descenders. Also it has the appearance of the 'bog standard' output. Most reasonable matrix printers these days offer an emphasised mode of printing, which, though slower, does offer a higher quality of output. As the manual is relatively short, it could even have been typed on the office typewriter.

The Am9511A Data Sheet

This is the one essential document for anyone who wants to use the HSA-88B, but is not interested in Hisoft Pascal. It gives you an outline of how the processor works, and a detailed breakdown of the instruction set. In fact I started this review with just the board and an AMD data book. Hisoft Pascal and the Belectra manual came later.

Hisoft Pascal

I don't intend this to be a review of Hisoft Pascal - I leave that to users like Dr Dark [3] - but I include here a few passing comments. (See also the following section on support.)

The supplied manual is for Hisoft Pascal version 4D. I assume that Hisoft Pascal 5, (supplied on the disc), is identical to 4, except with a small change in the run-time routines. I could find no "version differences" addendum sheet, although the Belectra manual refers to HP5 as having "various additional features" over HP4.

I have two comments to make on the manual: two small points that irritated me out of all proportion. The first is a subjective complaint (almost 'nit picking'), that I can't satisfactorily explain. The manual has been printed on a dot-matrix printer with a very acceptable typeface, and I have no objection to that, but I found parts of it very fatiguing to read. I suppose it is more the resultant small variation in quality from letter to letter that occurs in places in the manual that I find fatiguing, rather than anything else. It may be the copying process that has highlighted this, rather than the printer itself.

My second complaint is to do with the formatting of the manual. The

Am9511A

Table 1.

Command Mnemonic	Hex Code (sr = 1)	Hex Code (sr = 0)	Execution Cycles	Summary Description
16-BIT FIXED-POINT OPERATIONS				
SADD	EC	6C	16-18	Add TOS to NOS. Result to NOS. Pop Stack.
SSUB	ED	6D	30-32	Subtract TOS from NOS. Result to NOS. Pop Stack.
SMUL	EE	6E	84-94	Multiply NOS by TOS. Lower result to NOS. Pop Stack.
SMUU	F6	76	80-98	Multiply NOS by TOS. Upper result to NOS. Pop Stack.
SDIV	EF	6F	84-94	Divide NOS by TOS. Result to NOS. Pop Stack.
32-BIT FIXED-POINT OPERATIONS				
DADD	AC	2C	20-22	Add TOS to NOS. Result to NOS. Pop Stack.
DSUB	AD	2D	38-40	Subtract TOS from NOS. Result to NOS. Pop Stack.
DMUL	AE	2E	194-210	Multiply NOS by TOS. Lower result to NOS. Pop Stack.
DMUU	B6	36	182-218	Multiply NOS by TOS. Upper result to NOS. Pop Stack.
DDIV	AF	2F	196-210	Divide NOS by TOS. Result to NOS. Pop Stack.
32-BIT FLOATING-POINT PRIMARY OPERATIONS				
FADD	90	10	54-368	Add TOS to NOS. Result to NOS. Pop Stack.
FSUB	91	11	70-370	Subtract TOS from NOS. Result to NOS. Pop Stack.
FMUL	92	12	146-168	Multiply NOS by TOS. Result to NOS. Pop Stack.
FDIV	93	13	154-184	Divide NOS by TOS. Result to NOS. Pop Stack.
32-BIT FLOATING-POINT DERIVED OPERATIONS				
SQRT	81	01	782-870	Square Root of TOS. Result to TOS.
SIN	82	02	3796-4808	Sine of TOS. Result to TOS.
COS	83	03	3840-4878	Cosine of TOS. Result to TOS.
TAN	84	04	4894-5886	Tangent of TOS. Result to TOS.
ASIN	85	05	6230-7938	Inverse Sine of TOS. Result to TOS.
ACOS	86	06	6304-8284	Inverse Cosine of TOS. Result to TOS.
ATAN	87	07	4992-6536	Inverse Tangent of TOS. Result to TOS.
LOG	88	08	4474-7132	Common Logarithm of TOS. Result to TOS.
LN	89	09	4298-6956	Natural Logarithm of TOS. Result to TOS.
EXP	8A	0A	3794-4878	e raised to power in TOS. Result to TOS.
PWR	8B	0B	8290-12032	NOS raised to power in TOS. Result to NOS. Pop Stack.
DATA AND STACK MANIPULATION OPERATIONS				
NOP	80	00	4	No Operation. Clear or set SVREQ.
FIXS	9F	1F	90-214	Convert TOS from floating point format to fixed point format.
FIXD	9E	1E	90-336	
FLTS	9D	1D	62-156	Convert TOS from fixed point format to floating point format.
FLTD	9C	1C	56-342	
CHSS	F4	74	22-24	Change sign of fixed point operand on TOS.
CHSD	B4	34	26-28	
CHSF	95	15	16-20	Change sign of floating point operand on TOS.
PTOS	F7	77	16	
PTOD	B7	37	20	Push stack. Duplicate NOS in TOS.
PTOF	97	17	20	
POPS	F8	78	10	
POPD	B8	38	12	Pop stack. Old NOS becomes new TOS. Old TOS rotates to bottom.
POPF	98	18	12	
XCHS	F9	79	18	
XCHD	B9	39	26	Exchange TOS and NOS.
XCHF	99	19	26	
PUPI	9A	1A	16	Push floating point constant π onto TOS. Previous TOS becomes NOS.


```
                                .z80
                        title Patch to MBASIC for HSA-88B support 08-10-83
                        ;       'Hack' for HSA-88B review
                        ; This code is assembled to a .COM file that is executed before a
                        ; patched version of MBASIC is run. The 'front end' copies the
                        ; HSA-88B support routines up to 0C000h. Mbasic should then be run
                        ; with the command:
                        ;                       MBASIC /M:&HBFFF
                        ;       to ensure that Mbasic does not use any memory above C000.

                        ;       System equates

0C04                    fac     equ     0c04h           ; Floating point accumulator
28B7                    movrm   equ     028b7h          ; Routine to Load (HL) to registers
28A9                    movfr   equ     028a9h          ; Routine to move BCDE to the FAC

0080                    amd$d   equ     80h             ; AMD data port on HSA-88B
0081                    amd$c   equ     81h             ; AMD command port on HSA-88B

                        ;******************Front End**********************
                        ;       Short routine to move code into high memory

0000'   01 00AB                 ld      bc,codelen
0003'   11 C000                 ld      de,0c000h
0006'   21 000C'                ld      hl,ttt
0009'   ED B0                   ldir
000B'   C9                      ret
000C'                   ttt:
                        ;*********************************************************

                                .phase  0c000h          ; Locate them here

                                                        ; Address of where to patch MBASIC.
C000    C3 C01E                 jp      fpadd           ; 258B (ie change 258B to C3 00 C0...)
C003    C3 C022                 jp      fpmpy           ; 2707
C006    C3 C026                 jp      fpdiv           ; 276F
C009    C3 C02A                 jp      log             ; 26C4
C00C    C3 C02F                 jp      sqr             ;
C00F    C3 C034                 jp      sin
C012    C3 C039                 jp      cos
C015    C3 C03E                 jp      tan             ; 391C
C018    C3 C043                 jp      atn             ; 3931
C01B    C3 C048                 jp      exp             ; 372B

                        ;       Floating point ADD routine

C01E    3E 10           fpadd:  ld      a,10h           ; Add command
C020    18 2B                   jr      go
C022    3E 12           fpmpy:  ld      a,12h
C024    18 27                   jr      go
C026    3E 13           fpdiv:  ld      a,13h
C028    18 23                   jr      go
C02A    3E 09           log:    ld      a,09
C02C    F5                      push    af
C02D    18 22                   jr      go1             ; Only one operand
C02F    3E 01           sqr:    ld      a,1             ; Square root
```

```
C031    F5                     push    af
C032    18 1D                  jr      go1
C034    3E 02          sin:    ld      a,2
C036    F5                     push    af
C037    18 18                  jr      go1
C039    3E 03          cos:    ld      a,3
C03B    F5                     push    af
C03C    18 13                  jr      go1
C03E    3E 04          tan:    ld      a,4
C040    F5                     push    af
C041    18 0E                  jr      go1
C043    3E 07          atn:    ld      a,7
C045    F5                     push    af
C046    18 09                  jr      go1
C048    3E 0A          exp:    ld      a,0ah
C04A    F5                     push    af
C04B    18 04                  jr      go1

                       ; Save command, push operands into Am9511, issue command, read
                                                                                result

C04D    F5             go:     push    af
C04E    CD C06C                call    cnvamd          ; Push operand 1
C051    21 0C04        go1:    ld      hl,fac          ; Get 2nd
C054    CD 28B7                call    movrm
C057    CD C06C                call    cnvamd          ; Push it
C05A    F1                     pop     af              ; Add command
C05B    CD C064                call    cmnd            ; Carry it out
C05E    CD C08C                call    cnvms           ; Get result
C061    C3 28A9                jp      movfr           ; Put back in FAC & exit

                       ; Issue a command and wait for completion

C064    D3 81          cmnd:   out     (amd$c),a
C066    DB 81          cmnd0:  in      a,(amd$c)       ; Read status
C068    07                     rlca                    ; Test busy bit
C069    38 FB                  jr      c,cmnd0         ; Loop if so
C06B    C9                     ret                     ; Done (Ignore any errors)

                       ;       Conversion routine to convert Microsoft Format
                       ;       to AMD9511 format, while pushing the value onto
                       ;       the AMD stack.
                       ;       Number is held in BCDE, with the exponent in B.

C06C    78             cnvamd: ld      a,b             ; Load exponent
C06D    B7                     or      a               ; Zero?
C06E    28 16                  jr      z,zzz
C070    D6 80                  sub     80h             ; Remove bias
C072    87                     add     a,a             ; Shift up
C073    CB 11                  rl      c               ; Get sign of mantissa in carry
C075    1F                     rra                     ; Put on front of Exponent
C076    F5                     push    af              ; Save it
C077    37                     scf                     ; Set MSB of mantissa
C078    79                     ld      a,c
C079    1F                     rra
C07A    0E 80          camd:   ld      c,amd$d         ; Point to data port
C07C    ED 59                  out     (c),e           ; Send mantissa
```



```
C07E    ED 51                  out     (c),d
C080    ED 79                  out     (c),a
C082    F1                     pop     af              ; Recover exponent
C083    ED 79                  out     (c),a           ; Send it
C085    C9                     ret                     ; Done
C086    F5             zzz:    push    af              ; Set 0 on stack
C087    11 0000                ld      de,0            ; ensure DE is 0
C08A    18 EE                  jr      camd

                       ;       Routine to read a result back from the AMD9511, and
                       ;       to convert it to Microsoft format.

C08C    0E 80          cnvms:  ld      c,amd$d         ; Data port
C08E    ED 78                  in      a,(c)           ; Get Exponent + Mantissa sign
C090    ED 40                  in      b,(c)           ; Get mantissa
C092    ED 50                  in      d,(c)
C094    ED 58                  in      e,(c)
C096    48                     ld      c,b             ; Get to correct place
C097    47                     ld      b,a
C098    CB 11                  rl      c               ; Get rid of leading 1
C09A    87                     add     a,a             ; replace by Mantissa sign
C09B    CB 19                  rr      c
C09D    CB 2F                  sra     a               ; Extended Exponent sign while
                                                                            shifting
C09F    C6 80                  add     a,80h           ; Replace the bias
C0A1    47                     ld      b,a             ; Put in B
C0A2    DB 81                  in      a,(amd$c)       ; Check if was 0
C0A4    E6 20                  and     20h             ; Zero?
C0A6    C8                     ret     z               ; No, done
C0A7    01 0000                ld      bc,0            ; Yes, flag it
C0AA    C9                     ret                     ; Done.

                               .dephase
00AB                   codelen equ     $-ttt           ; No. of bytes to move

                               end

Macros:

Symbols:
0081    AMD$C       0080    AMD$D       C043    ATN
C07A    CAMD        C064    CMND        C066    CMNDO
C06C    CNVAMD      C08C    CNVMS       00AB    CODELEN
C039    COS         C048    EXP         0C04    FAC
C01E    FPADD       C026    FPDIV       C022    FPMPY
C04D    GO          C051    GO1         C02A    LOG
28A9    MOVFR       28B7    MOVRM       C034    SIN
C02F    SQR         C03E    TAN         000C'   TTT
C086    ZZZ

No Fatal error(s)
```

review manual came bound with one of those pull-on plastic spines. I have no objection to them, I've used them myself, but when I opened the manual I found I could not read the start of most lines as they were lost in the binding. Pulling the plastic spine off cured that, but then left me with a pile of loose paper. My alternative method of storing manuals - four-hole punching the pages and storing them in a ring binder - would not be perfect either, as the holes would mutilate some of the text. I measured the left and right margins at 10mm and 15mm respectively. I reckon they should be 15mm at an absolute minimum, 20mm would be better.

On glancing through the Hisoft HP5 manual one thing I noticed immediately was that the range of the floating point numbers was given as 3.4×10^{38} to 5.9×10^{-39}. However the range that the Am9511 can handle directly is only 9.2×10^{18} to 2.7×10^{-20}. Subsequently I wrote a short program that verified that the limits in HP5 are those of the Am9511.

Support

The area where a product like this stands or falls is in how easy it is to use. All your current software will obviously ignore it, and it is not a trivial task to get round this problem. Belectra have taken a step in the right direction by including in the price a special version of the Hisoft Pascal compiler whose run-time routines make specific use of the arithmetic processor. So if you want to "plug-in-and-go", you are currently restricted to Hisoft Pascal, or Pascal/MT+ which also has an option for using the Am9511.

Trying it out

I list below a few comparitive benchmarks of the performance. A nice benchmark I came across recently [4] is a good test of processor speed and accuracy on purely arithmetic routines. This very nicely matches the area in which the HSA-88B would be used. Try it on your current favourite interpreter and/or compiler for comparison with the figures below. Ideally it should produce a result of 2500 in as little time as possible.

```
5  REM Benchmark from DDJ No 83 (Sept '83) p120-122
10 N%=2500
20 A=1
30 FOR I%=1 TO N%-1:A=TAN(ATN(EXP(LOG(SQR(A*A)))))+1:NEXT I%
40 PRINT USING "A=££££.££££";A
```

Fig 1 - DDJ Benchmark in BASIC

Language	Result		Time	
Standard MBASIC interpreter:	2304.86	in	225 secs	
BASCOM compiler :	2304.86	in	183 secs	
*Modified MBASIC interpreter:	2326.94	in	41.8 secs	Am9511 support
Hisoft Pascal (V5) :	2326.94	in	29.1 secs	" "
Simple assembly code version of the benchmark :	2326.94	in	26.8 secs	" "

* see below

Fig 2 - Benchmark timings (4MHz Z80)

Unfortunately several Pascals do not appear to offer TAN as a function, (they do have SIN and COS), and only have ARCTAN for the inverse functions. So, for the Pascals, I have translated the PCW BM8 into Pascal.

Language		Time	
Pascal/MT+ (V5.5) normal	:	36.7 secs	
*Pascal/MT+ (V5.5) - Am9511	:	7.42 secs	Am9511 support
Compas Pascal (V1.06)	:	57.4 secs	
Hisoft Pascal (V5)	:	6.39 secs	Am9511 support

* Am9511 library routines are already available on the Pascal/MT disc

Fig 3 - PCW BM8 timings (4MHz Z80)

Using the HSA-88B with other languages.

I decided it would be instructive to try to use the processor from another language. My current language is C in the form of the C/80 compiler from the Software Toolworks (highly recommended). The version I run does not support floating point numbers, but then as I don't use them I find this no loss. A few moments thought showed that it would be very easy to add support for the arithmetic processor with this compiler, (in fact I first tried the board out with a simple C program), but I decided that it would be more instructive to try to add support to something like the ubiquitous Microsoft Basic80.

The benchmark above indicated that the Microsoft interpreter and its companion compiler use the same arithmetic subroutines, and as the benchmark for the compiler was only 20% faster than the interpreter I decided to use the interpreter rather than tackle the run-time routines of the compiler.

The easiest (although slightly clumsy) way to interface the card would be by the USR or CALL interface, but I decided to try to patch the interpreter directly. On turning up appendix C of the MBasic manual I encountered problem number one. The two floating formats are not identical (see below). However they are basically similar, so it is not out of the question to interface the HSA-88B to the interpreter.

Am9511 format
=============

The mantissa is expressed as a 24-bit (fractional) value; the exponent is expressed as an un-biased two's complement 7-bit value having a range of -64 to +63. The most significant bit is the sign of the mantissa (0=positive, 1=negative), for a total of 32 bits. The binary point is assumed to be to the left of the most significant mantissa bit (bit 23). All floating-point data values must be normalised. Bit 23 must be equal to 1, expect for the value zero, which is represented by all zeros.

MBASIC single precision
=======================

The mantissa is expressed as a 24-bit (fractional) value with the leading 1 suppressed (implied) and the binary point is to the left of the most significant bit; the exponent is expressed as a biased 8-bit number having a bias of 128. The sign of the mantissa is in bit 23 (0=positive, 1=negative).
The number zero is represented by a number with an exponent of 0.

The two are similar, and re-arranging a few of the bits can get them to line up, but the only problem is that the Microsoft exponent range is +127 to -127, as opposed to the -64 to +63 of the Am9511. As this was just an exploratory hack, I took the easy way out and ignored this!

Presented with 24k of code of MBasic, it's rather daunting to know where to begin, but armed with a copy of "Nascom Basic Dis-assembled" and Gemdebug, it is only a moment's work to discover where the arithmetic routines are in Mbasic. (Both interpreters originate from Microsoft, and so there is a certain amount of commonality. Use the "W" command to locate the sequence 78,B7,C8,3A.) After this the fun began! I must emphasise that this is a crude 'hack', and is not supposed to be a sophisticated modification. The points where I patched my routines in, appear to be correct, but there may be some unexpected side effects, and, as mentioned above, possible exponent overflow is ignored. Also without the information I had already to hand, it would have been a much harder task.

Listing 1 shows the support routines I put together over one evening. The explicit test for zero in the format conversion routines turned out to be necessary, otherwise odd results like 1.0 + 1.0 = 2.125, and 1.0 - 1.0 = 0.5 appeared! The net effect on the benchmark timings can be seen in Fig 2, so a significant increase in the execution speed of number-crunching programs could be expected

SUMMARY

If you do a lot of number crunching, or have a real-time application that requires high-speed crunching, then this is a product you should consider. However you should bear in mind that unless you are prepared to do some work yourself, you are currently restricted to Hisoft Pascal for your programming language.

It is not a trivial task to modify existing compilers/interpreters to support the HSA-88B. This is because the existing floating-point routines may not use the same floating-point format as the AMD9511, and also it is not an easy task to unravel the complexities of floating-point support packages.

If you have a dedicated application, then the HSA-88B makes the writing of a floating-point package a relatively simple operation. The main effort will be in producing the ASCII-to-floating-point, and the floating-point-to-ASCII conversion routines. But with the FIX and FLOAT commands available within the AMD9511, this shouldn't be too onerous.

The revised price of the HSA-88B of £253+VAT seems a little high - especially if you are not interested in the Hisoft Pascal compiler - but perhaps this is to be expected of a low volume product.

If you want 'more bang for your bucks', then consider getting a version with a 4MHz Am9511A fitted, you'll get a x2 increase in execution speed, for a lesser (x1.3?) increase in cost. The speed increase will only be in the arithmetic however, not in the supporting Z80 code.

One final observation: if you need higher precision and can manage without the derived functions (and Hisoft Pascal 5), I can see no reason why an Am9512 couldn't be fitted in the board in place of the Am9511A.

References:

1. COONEN,J. "An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic", COMPUTER, Vol 13-1, January 1980, pp68-79
2. STEVENSON,D. "A Proposed Standard for Binary Floating-Point Arithmetic", COMPUTER, Vol 14-3, March 1981, pp51-62.
3. DARK,Dr., "Doctor Dark's Diary - Episode 16", 80-BUS NEWS, 2-3, May-June 1983, pp54-56
4. DUNCAN,R., "16-Bit Software Toolbox", Dr Dobb's Journal, No 83, Sept 1983 pp120-122

MBASIC RANDOM ACCESS FILES, AND SOME OTHER NOTES

By C. Bowden (G3OCB)

The notes on MBASIC in 80-BUS NEWS Vol 2. Issue 3 by P.D. Coker gave a fair summary of many of the commands and functions found in MBASIC. As the author states, the manual is not a beginners book. Despite the examples in it the manual fails to show the power and scope of many of the commands. In this article I will explain a few of the features of MBASIC that are not obvious, particularly with regard to RANDOM FILES, which are an essential part of most serious programs.

First an explanation of what Random files are. There are two main types of DATA files in use on Computer systems - Sequential and Random. (Sometimes a mixture of both may be used.) A Data file will consist of something like a list of names and addresses, or a list of parts, prices and suppliers, etc. As the name implies, items in a Sequential file can only be read in strict sequence. Thus a file system on a cassette is almost certainly sequential. Disks are essentially Random access devices though, because the head can go to any part of the disk at any time. MBASIC supports Sequential and Random access files (the latter needs CP/M 2.x or better). I prefer to use Random files since they have a number of advantages over Sequential files. These include :-

1) A file can be 'OPEN' for 'READ' and 'WRITE' at the same time.
2) Any record (number) can be accessed directly.
3) There is no need to use 'RENAMING' routines to avoid losing Data files.
4) More than one file can be open at a time.

A particular advantage of Sequential files is that Data types can be mixed (i.e. String and Numeric). With Random files only String Data can be used. This does not mean that Numeric data cannot be used, as will be shown. Data files of both types are composed of 'RECORDS'. The final output of the program may be the result of combining records from several different data files. The rest of this article applies only to Random files. You will see from the title of this article that I am a Radio Amateur. Let us assume that I wish to use Random files to keep a 'card index' of stations contacted, so that I can quickly find out if I have contacted a station previously. Assume that I decide to save the following data. Obviously I could have chosen to save much more Data such as full address, equipment details, radio conditions, signal strengths etc.

For each separate Record on Disk the following Data items :-

Name	- allow 18 characters
Callsign	- allow 8 characters
Town	- allow 20 characters
County/State (etc;)	- allow 16 characters
Date of First contact	- allow 8 characters

The first thing to note is that I have allowed a certain number of characters per item. The reason for this is that the length of the Record to be saved will be the total length of all of the items. Normally BASIC will not allow the length of a Random record to vary so it is necessary to guess at the MAXIMUM length of each item, and to add them all together. The total length of the above record is 70 characters. This space must therefore be left for each

Record saved. Undoubtedly this leads to wasted space since many Data items may contain a lesser number of characters, but it is usually not worth worrying about for Domestic or small Business jobs. After all if the disk has 350K of space, and CP/M, BASIC and the operating Program use say 50K bytes then it is possible to store over 4000 (70 character) Records in the remaining space. It is also possible to split the Data over several disks. (If the original guess of maximum length of a field was too small, it will be necessary to abbreviate the Data, or a program may be written to Read and rewrite the original Data file, but with a longer Field for the item in question.)

The various items that make up the Record are saved on disk in sections called 'Fields' :-

One complete Record of 70 characters for the RADIOLOG program would be :-

NAME (18)	CALL (8)	TOWN (20)	COUNTY (16)	DATE (8)
Name field	Call field	Town field	County field	Date field

Since the Record length is fixed, BASIC can easily 'calculate' where any record starts relative to the beginning of the Data file, thus allowing easy access. Unused characters in a field are 'padded out' with blanks. The article by Mr Coker implied that a Record could not exceed 128 Bytes in length. This is certainly not the case with Vers. 5.2 of MBASIC. I have used Records of around 600 bytes in some Programs. In order to use records longer than 128 bytes though, MBASIC must be 'informed' about the longer record when it is executed as follows :-

MBASIC FILENAME / S: 286 (for 286 BYTE Records)

It is not necessary to use the / S switch for records of less than 128 Bytes. Note also that use of the / S switch does NOT make it unnecessary to declare the Record length when a file is 'OPEN'ed within the program. If you change the length of Records or Fields during program development, be sure that you always change the Field and Record declarations, or the program will either crash, or you will get some funny Data outputs. The sum of ALL the Fields MUST equal the length of record declared when the File is OPENed from within the Data processing program.

If MBASIC is run up with the / S option specifying a record longer than 128 bytes, one of two situations can arise when the File is OPENed. If the total Record declared on OPENing the File is longer than specified on run-up, then an error message will occur when the line OPENing the File is reached. But if the OPENing length is equal to, or less than the / S: declaration however, no errors occur.

I have used the BASCOM Compiler to compile files with records of up to 600 bytes or so quite successfully, without having to 'declare' them in some way before compiling.

To Read and then Write a Random file the steps are as follows :

1) OPEN the file
2) Declare the Fields
3) GET the required Record from Disk, if records already exist, or INPUT the necessary data and build the record.
4) Process the Fields
5) PUT the record onto Disk.
6) Repeat 3 to 6 until all done
7) CLOSE the File

e.g.

```
90 INPUT "Enter CALL to be found. ";CSGN$
100  'do some input error checking here.
110 OPEN "R",£1,"RADIOLOG.DAT",70
120 FIELD£1,18 AS NAME$,8 AS CALL$,20 AS TOWN$,16 AS COUNTY$,8 AS DATE$
130 FOR RNUM=X TO Y    'X,Y define range of record numbers to be read.
140 GET£1,RNUM
150 IF CALL$=CSGN$ THEN GOTO xxxx
160 NEXT RNUM
170 PRINT "Record not found. "
180 CLOSE£1
```

Line 110 - OPEN's the file, declares it as RANDOM, and describes it as File £1 until it is closed. The name of the Data file to be known as File £1 is RADIOLOG.DAT and it uses Records of 70 bytes.
Line 120 - Sets up the FIELDS for file £1. Each Field is given a Variable name and its length is set.
Line 140 - GET's each Record in turn from the Disk until the search either succeeds or fails altogether.
Line 150 - Looks for match between INPUT data and Disk record data. (N.B. in this simplified example, Line 150 will probably fail to work since "G30CB" does NOT equal "G30CB   ". To work correctly, the trailing blanks would have to be removed, or the INPUT callsign padded out to 8 characters with blanks.
Line 160 - Continues looping until record found, or all done.
Line 170 - Prints appropriate message.
Line 180 - CLOSEs the file.

FIELDS

Although it is possible to use the Data in the Fields directly, as in line 150 above, it is normal practice to 'move' the Data out of the Field variable into some other variable for processing. There are several reasons for this:-

1) The Field data will contain leading or trailing blanks which can lead to unreliable results when Processing or Printing Data.
2) Numeric data is stored in Fields as Strings. It must be converted back to numeric form for calculations.
3) Integers are stored as 2 Byte Fields, Single Precision as 4 Bytes, Double Precision as 8 Bytes.
4) It is often convenient to have the origional Field in uncorrupted form during processing.

In the example above, we could have written:

```
145 NM$=NAME$:CL$=CALL$:TN$=TOWN$:CTY$=COUNTY$
150 IF CL$=CSGN$ .......
```

The problem of the blanks would still remain though. One way of dealing with the blanks is simply to read the Field from left to right and discard blanks. This is O.K. until we realize that some Fields contain needed blanks (e.g. SOUTH ROAD). A better way to deal with the problem is to deliberately mark the end of the required data with a special Character:

```
50 INPUT "Enter Name of Town ";TN$
60 TN$=TN$+"*"
70 LSET TOWN$=TN$
```

Assuming use of a 20 byte field, and that the word 'LONDON' is INPUT, this would store the Field TOWN$ as 'LONDON*   '. To extract the Town from the Disk Record, less blanks and Field terminator, the following type of routine can be used :

```
200 TEMP$=CALL$:GOSUB 2010:CL$=OP$:TEMP$=TOWN$:GOSUB 2010:TN$=OP$: etc;
210 .....

2010 OP$=""
2020 FOR X=1 TO LEN(TEMP$)
2030 IF MID$(TEMP$,X,1)="*" THEN GOTO 2060
2040 OP$=OP$+MID$(TEMP$,X,1)
2050 NEXT X
2060 RETURN
```

The reason for using the "dummy" variables TEMP$ and OP$ is that all Fields that need processing in this way can then use the same Subroutine.

Line 2010 - Ensures that the dummy variable is empty at the start.
Line 2020 - Sets up a loop equal to the length of the Field.
Line 2030 - Reads the copy of the field stored in TEMP$ from left to right, one character at a time. If the character found is a "*", the program skips to 2060 to return to the main flow.
Line 2040 - The character is added on to OP$.
Line 2050 - The loop continues until a "*" is found or the whole length of the Field has been processed.

There is thus no need to expand the Fields already declared to accomodate the "*" character. If for example a "*" were added to the date: "20/06/83*", then since only 8 bytes have been allowed for the date Field, when LSET the "*" would "drop off the end". N.B. A 'RSET' would have stored "0/06/83*" in the Record Date Field.

USE of LSET and RSET to 'FIELD' DATA.

When Data is stored in a Field, it is padded out with Blanks. The command that achieves this is LSET or RSET.

```
110 OPEN "R",£1,"RADIOLOG.DAT",70
120 FIELD£1,18 AS NAME$,8 AS CALL$,20 AS TOWN$,16 AS COUNTY$,8 AS DATE$
140 INPUT "Enter CALLSIGN. ";CSGN$
```

```
150 INPUT "What TOWN. ";TN$
160 INPUT etc;
170 LSET CALL$=CSGN$:LSET TOWN$=TN$  'etc;
180 PUT£1,NEXTRECORD
```

Suppose that CSGN$="G30CB". Then since the CALL$ Field is 8 Bytes long, the effect of 180 is to produce a LEFT justified Field "G30CB   ", i.e. 3 trailing spaces are added to 'pad out' the variable. It does not normally matter whether the field is LEFT or RIGHT justified as long as the same standard is used throughout the program. It is obviously essential to know how the Field is justified. In some cases such as the Date Field, there will be no padding and the length will be constant. Thus the Field DATE$ can often be used directly.

NUMERIC FIELDS.

Much of the above discussion is correct only in respect of String data. It is usually O.K. to store some types of numbers directly into strings and such numbers can be retained in this way. e.g.

```
170 INPUT "Enter Telephone Number ";TEL$
```

Numbers to be used in calculations may also be input in this way if preferred. In fact it may be a good idea to input all data in string form as this can mean easier and better Error trapping. Note that numbers stored in this way will usually require more Disk space. Data stored in this way must however be converted to Numeric form by use of the VAL function before any calculations can be carried out.

The alternative way to store Numeric data still requires that it be LSET or RSET. Before this can be done the number must be converted into String form by the use of a special function. One of three possible choices must be made:

MKI$ = Make Integer Number into a 2 byte string.
MKS$ = Make Single Precision Number into 4 byte string.
MKD$ = Make Double Precision Number into 8 byte string.

And typical usage:

```
20 DEFSNG A-H

60 FIELD£2,4 AS BBAL$,etc
120 INPUT "Enter Bank Balance ";BAL
130 LSET BBAL$=MKS$(BAL)
```

Line 20 - Defines that all numeric variables from A-H are to be treated as Single Precision.
Line 60 - Since BBAL$ is the Field for the Bank Balance, assumed single Precision, a space of 4 bytes is set.
Line 120 - The balance is entered as a number.
Line 130 - The balance is made into a single precision string, and LSET into the Field.

There are also three special functions to extract numeric values from a Field string. They are:

CVI = Convert String to Integer number.
CVS = Convert String to Single Precision number.
CVD = Convert String to Double Precision number.

For example:

```
20 DEFDBL A-H
80 OPEN ....
90 FIELD£1,8 AS NUMBER$, etc
130 GET£1,6   'ie; READ record number 6
140 NUM=CVD(NUMBER$)
150 etc
```

Here record number 6 is read and the Field 'NUMBER$' is converted into a Double Precision value that is stored in variable 'NUM' for further processing. Note the declaration of the precision of numeric variables in 20. The main advantage of the use of CV and MK functions is that less space is required in the records, and on disk.

EXTENDED FIELDS and ARRAYS.

It is often necessary to store a lot of Data in the Fields of one Record. For example the amount spent per month on Petrol. It would be very tedious to have to type in something like ....., 4 AS JAN$, 4 AS FEB$ etc. (It is also convenient to restrict line lengths to less than 80 characters to allow as much use as possible of the extended screen editing features of SYS/GEMINI CP/M). Fortunately it is possible to continue FIELD statements on to several lines, and to use FOR-NEXT loops in their construction:-

```
10 DIM MTH$(11),DATES$(11),BALANCE$(11),etc
20 MTH$(0)="JAN":MTH$(1)="FEB":etc

70 OPEN "R",£1,"SALARY.DAT",214
80 FOR X=0 TO 11
90 FIELD£1,20 AS NAME$,24 AS STREET$,18 AS TOWN$,8 AS POSTCODE$
100 FIELD£1,70 AS DUMMY$,(12*X) AS SPACE$,4 AS BALANCE$(X),8 AS DATES$(X)
110 NEXT X
```

This example is not so easy to understand.

Line 20 - The names of the months are stored in an array so that they can be printed using a FOR-NEXT loop as desired later.

Line 70 - The file is OPEN'ed. It is assumed that it stores the Monthly salary and Date of payment for the person named in the record. The Record is 214 bytes long as will be shown.

Line 90 - Sets up Fields for the self evident bits of Data.There is not sufficient room in 90 to continue unless a non-screen editable line is entered. The Fields in line 90 occupy 70 bytes.

Line 100 - The first Field is a 'dummy' to ensure that all the Fields in line 100 start later than Fields in line 90.As long as no Data is LSET or RSET into the Field DUMMY$ then Data in the Fields of line 90 is not corrupted. SPACE$ is another dummy, used as described below.

Line 110 - Continue the loop until 12 'BALANCE' and 'DATE' fields are set.

The facility of moving along the Record to the next Field is used in a 'dynamic' way. The first value of X in the FOR-NEXT loop of line 80 is 0. SPACE$ is a 'dummy' field that 'grows' in size. The first time round its size is 0 bytes (since 12*0=0), so the Field BALANCE$(0) will be the 4 bytes immediately after the Field POSTCODE$. The Field DATES$(0) will be the next 8 bytes.

Note the use of ARRAYS to store the Balance and Date Fields. The second iteration of the loop will fix SPACE$ at 12 bytes and so Fields BALANCE$(1), DATES$(1) will be pushed just beyond BALANCES$(0), DATES$(0). The process continues until the 12 Fields (0) to (11) have been set up.

The total size of the Record is thus 70+(12*12), or 214 bytes. Data can be set into the BALANCES$(), DATES$() array Fields from loops, or directly as the program dictates. Again so long as no Data is 'SET' into SPACES$, then no corruption of other Data occurs.

DYNAMIC DATA FILES and AUTOMATIC BACKUP.

It is always good practice to make at least two (separate disk) copies of all Files, and DATA files are no exception. It is quite easy to make the BASIC program save Records to Disks on two or more drives during processing.

Another problem that can often arise is that one might wish to run a certain program on one of a number of data files. For example I might have my Callsign files organised as "A-G.DAT", "H-M.DAT", "N-R.DAT" or "S-Z.DAT".The problem here is not too great as I could offer a choice of the 4 files, and open the appropriate one. One program that I wrote needed to access 99 different small DATA files of 2 or 4K bytes each, and so a better method was needed.

```
10 DIM CLASS(20),
20 INPUT "Which Class (1 to 20). ";CN$
30 CN=VAL(CN$)
40 REM CHECK ROUTINE HERE - Check 'CN' for Errors
50 FNAM$="CLASS"+"CN$"+".DAT"
60 OPEN "R",£1, FNAM$,78
70 FIELD etc;
```

When I use Random files, I always like to 'Initialize' my Data files before running my Data handling program for the first time (assuming that I know how many Records the file will contain). There are several reasons for this. The main reason is that I can then be sure that I know exactly what Data is in any Field of any Record. If Data is read from an unwritten and un-initialized record, sometimes peculiar things can happen, since this can contain any 'junk' depending what that part of the disk was used for previously. I usually set up Fields as all blank, or all 'Periods' as these are convenient for immediate printing. It may sometimes be useful to set up a small field as a 'Marker' to indicate whether the Record is Free or not. There is scope here for the use of FOR-NEXT loops.

```
10 REM FILE INITIALIZATION -  INITAREA.BAS 06/09/83
20 PRINT CHR$(26):PRINT TAB(20) "AREA FILE INITIALIZATION"
30 FOR X=1 TO 3
40 FNAM$="AREA"+STR$(X)+".DAT"
50 FOR RN=1 TO 100
```

```
60 OPEN "R",£1,FNAM$,582
70 FIELD£1,2 AS MK$,3 AS NUM$,5 AS DIS$,22 AS NAM$
80 FOR A=1 TO 9
90 FIELD£1,32 AS DM$,30*(A-1) AS JP$,5 AS CN$(A),20 AS CNA$(A),5 AS CONS$(A)
100 FIELD£1,62 AS JM$,240 AS JU$,31*(A-1) AS M$,14 AS T$(A),11 AS O$(A),6 AS
    C$(A)
110 NEXT A
120 LSET MK$="FR":LSET DIS$=".....":LSET NAM$="...................."
130 LSET NUM$="..."
140 FOR A=1 TO 9
150 LSET CN$(A)=".....":LSET CNA$(A)="..................."
160 LSET CONS$(A)=".....":LSET T$(A)=".............."
170 LSET O$(A)="...........":LSET C$(A)="......"
180 NEXT A
190 PUT£1,RN
200 CLOSE£1
210 OPEN "R",£1,"B:"+FNAM$,582
220 FIELD£1,2 AS MK$,3 AS NUM$,5 AS DIS$,22 AS NAM$
230 FOR A=1 TO 9
240 FIELD£1,32 AS DM$,30*(A-1) AS JP$,5 AS CN$(A),20 AS CNA$(A),5 AS CONS$(A)
250 FIELD£1,62 AS JM$,240 AS JU$,31*(A-1) AS M$,14 AS T$(A),11 AS O$(A),6 AS
    C$(A)
260 NEXT A
270 PUT£1,RN
280 CLOSE£1
290 PRINT "FILE ";FNAM$;" - RECORD No ";RN;" INITIALIZED. "
300 NEXT RN
310 NEXT X
320 SYSTEM
```

This example includes a number of the features previously described. The code used may not necessarily be the most elegant or concise, and the example has been 'made up' to show the various points.

Line 10 - Advisable for record purposes.
Line 20 - Clear Screen and print a Title. (Assumes Gemini IVC Screen.)
Line 30 - We will have 3 Data Files. AREA1.DAT to AREA3.DAT.
Line 40 - Construct File Name.
Line 50 - Set record counter for one File.
Line 60 - OPEN the File on drive "A", RANDOM, 582 byte Records.
Line 70 - 110. Set up the Fields for drive 'A', (File £1). Note line 100 the use of JM$ and JU$. MBASIC does not like numbers over 255 in Field statements, so the dummy has been split. (62+240=302.)
Line 120 - 180. Set ALL fields (Except MK$) to periods. This allows a nice display when Data entry program is run and current state of any 'empty' Fields is displayed.
Line 120 - Set MK$ to "FR", to mark the record as FREE.
Line 190 - Save the initialized record to Drive "A".
Line 200 - CLOSE File on Drive "A".
Line 210 - 270. OPEN a File on Drive "B", set up the FIELDS, and Save the 'empty' Record. N.B. No need to LSET again since the FIELDS are identical.
Line 280 - CLOSE file again.
Line 290 - Keep the operator happy.
Line 300 - Continue until 100 Records written to the file.

Line 310 - Continue until 3 Files done.
Line 320 - Back to CP/M.

The above Program would initialize 3 files of 100 records, each of 582 bytes, on drives A and B, marking each record as free, and filling all fields (except MK$) with 'Period' characters. There are 9 repetitions of 6 different fields in each record in this example.

e.g. CN$(1),CNA$(1)......O$(1),C$(1) up to CN$(9),CNA$(9)......O$(9),C$(9)

Each File will consist of 100 records x 582 bytes = 58,200 bytes or app 58K. Thus the Three Data files will occupy about one half of each Disk space. It will sometimes be convenient to try to structure a Data File so that it can be kept to less than about 40K (assuming 64K RAM). If this can be done then if a lot of Data processing is going to occur the Data can be READ directly into ARRAYS in RAM for processing, and WRITTEN out to Disk afterwards. Processing will then be a lot faster, and of course Disk and Drive wear will be minimized. Another alternative is to use Virtual Disk.

Note that in this example the SAME field variables were used for drives A and B. Because of this, the two files could not be OPEN and PUT at the same time, but only one set of 'LSET' statements was needed. An alternative method is to have both files OPEN at the same time. If this is done then the file number '£1' cannot be used for both files and one would have to be referred to by another number like '£2'. The two sets of FIELD statements would then have had to use differing Variables, and an additional set of 'LSET' statements would also have been needed. A further slight complication is that extra DIM statements might be necessary, for any extra FIELD variables that use arrays. (Especially where the program is to be compiled and all Arrays must be DIM'ed.)

TO SCROLL or NOT TO SCROLL?

When processing Data it is common practice for programmers to use a lot of PRINT CHR$(26) statements to clear the screen, and then to loop back to reprint updated Data, or to PRINT a succession of INPUT prompts at the bottom of the screen. This causes the display to scroll upwards, particularly where input errors have occured and Error messages and repeat prompts are printed. A much more effective method is available. The Gemini IVC card supports a number of commands that enable the cursor to be repositioned, and the screen to be cleared from the cursor. If this approach is used, then a much steadier display is possible. e.g.

```
150 FLAG=0:GOSUB 3000
160 INPUT "Enter the name of the circuit. ";CIRNAM$
170 - etc;  would be error trapping and data storing routines with error
      messages

200 FLAG=0:GOSUB 3000
210 INPUT "Enter TIME .... etc;
                                    20H offset
3000 RR=18:CC=0
3010 PRINT CHR$(27);"=";CHR$(32+RR);CHR$(32+CC)
3020 IF FLAG=1 THEN RETURN
3030 PRINT CHR$(27);"%"   —   clear screen,
3040 RETURN
```

The subroutine at 3000 would position the cursor on the eighteenth line, first column, assumed to be just below the nicely displayed and formatted data relating to the Record being edited, and the screen from lines 18 to 25 would be cleared. By calling this subroutine before each message then scrolling would be avoided. If RR, CC are set and the routine is called at line 3010, then other cursor positions can be used. The FLAG in line 3020 can be set or not depending whether it is desired to clear the screen from the cursor. Other commands may be used to 'lock' the screen. The Gemini IVC manual will give full details.

By placing the cursor on the top of the screen, the existing display can be updated without scrolling or clearing. The effect is much more pleasing, particularly with compiled BASIC, where the speed is such that there is no suggestion of flicker or snow.

OTHER ODD BITS of INFORMATION.

WIDE PRINTING

As DRH has remarked several times, if all else fails READ THE MANUAL. I didn't read it carefully enough, but David was quite polite when I bothered him with this problem recently. I wanted to print 233 Chars. wide (Condensed) on a MX100, but I was getting LF's at the default of 132. I had tried altering the print width in SYS, as well as playing with the switches in the printer, but no BASIC commands did what I wanted. Prompted by Dave, I read the manual more closely, and this revealed an additional command - LPRINT WIDTH, which solved the problem. (This command was described under WIDTH, and not LPRINT in the MBASIC manual. Maybe this is why it was not very obvious.)

GRAPHICS CARD ADDRESSES

I recently built the ANIMATION GRAPHICS card to keep my son amused. This card offers reasonably high resolution in 16 colours with relatively easy programming, as well as a number of other features. A major problem for me was that the addresses on the card run from 0 to 29 (including NASIO), clashing with my NASCOM I/O card. It is quite easy to move the address range though. Cut the track from IC2 pin 16, to IC7 pin 9. (This is address line 5). Then mount a suitable Inverter chip (e.g. 74LS04) on a small piece of VERO board (about 1" by 1/2"), and stick this with some double sided white sticky tape to the component side of the card, near to IC's 5, 7, having first soldered on 4 short wires for +5v, E, input and output to one inverter circuit. Connect the inverter across the cut track so that it inverts the signal on address line 5. The addresses on the card will now range from 32 to 40. Note that the NASIO O/P will NOT now be correct, and this signal if needed will have to come from elsewhere.

This article was written with the new GEMPEN/DISKPEN and this new version is a tremendous improvement over the earlier versions. [Ed. - although this article may have been written using PEN, it has since been converted by me into a WordStar file, as I now like to deal with the complete mag. with one word processor, if possible, and WordStar is much more appropriate for major literary works like this!] The 'HELP' overlays are very useful, and the commands have been improved so as to render disasters much less likely. Printer support has been vastly improved. The net result is a very useful program. Although not as sophisticated as WORDSTAR, I prefer it for the sort of jobs that I carry out on my machine.

AUNT AGATHA'S AGONY COLUMN

by David Parkinson

LEADER

Here we are again! It only seems like yesterday that I finished the previous one of these. [Ed. - I'm sure there was something similar to this said in Vol.2 Iss.3. Zero points for originality!] Anyway as promised...

NASCOM 2 and 2716/2732 EPROMS

Aletter from Bill Stewart of Kings Lynn has prompted me to blow the dust off my Nascom 2. He has modified his N2 to take 2716 EPROMs in place of the 2708s in the eight sockets on the main pcb and has run into a speed problem. He has found that he has to use wait states if he wants to execute code in the 2716s. (His system runs at 4MHz). His letter doesn't explain why, but he's trying to squeeze the last drop of speed out of his N2, and his current target is to only enable wait states while accessing the on-board EPROMs during M1 cycles.

Rather than tackling his problem directly, I've taken one step back from it to the question of how you should modify the N2 to support the 2716 & 2732 type EPROMs. Bill doesn't say how he has modified his N2, but various suggestions have been published in the NEWS [1][2][3]. I would hope that the suggested approach below would allow the EPROMs to be used without wait states. The 2716/32 type of EPROMs, as well as having a greater capacity than the 2708 and only requiring a single +5v supply, also have a more more subtle difference. When the 2716 was first introduced by Intel, if my memory serves me right, pin 18 was designated PWR DWN, and pin 20 was /CS. (If you check the data sheets you will find that the 2716s power consumption drops by about a factor of 5 when pin 18 is at a '1'). Shortly afterwards they renamed the pins, and pin 18 became /CE (Chip Enable) and pin 20 /OE (Output Enable). The distinction is fairly important, and if you check the data sheet again you'll see why. The access time for a standard 2716 is 450ns from the address lines and the /CE pin, but only 120ns from the /OE pin. Thus in a memory system, in order to make the most of the EPROM, the /CE decoding should be done as soon as the address is available. Any later qualification, (like using /MREQ to distinguish between a memory or an IO access), should preferably be applied to the /OE signal. Also the 2-pin approach makes it easy to design systems without memory contention problems.

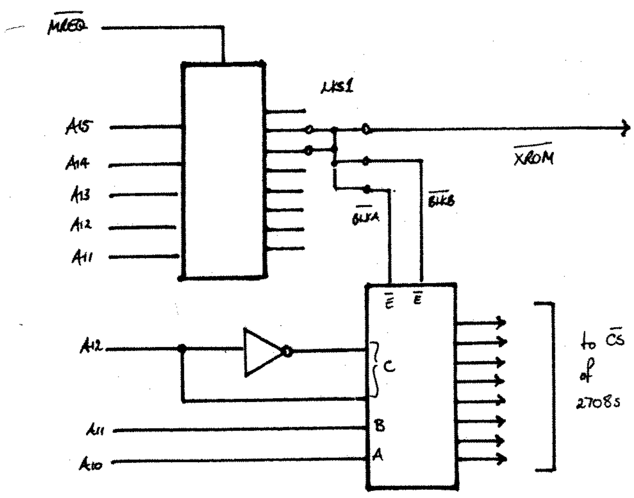


Fig 1 - Normal decode

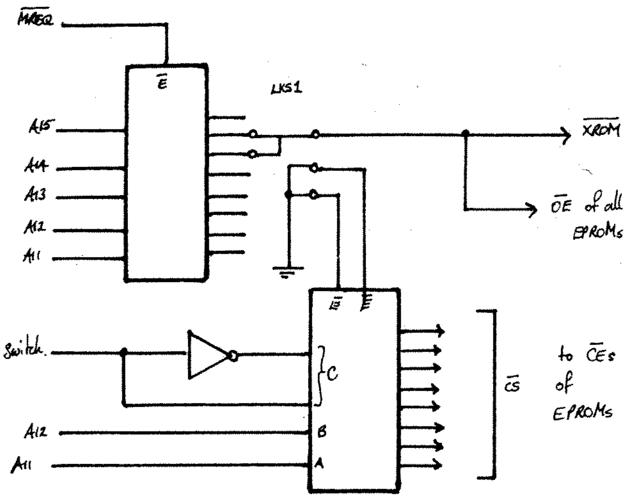
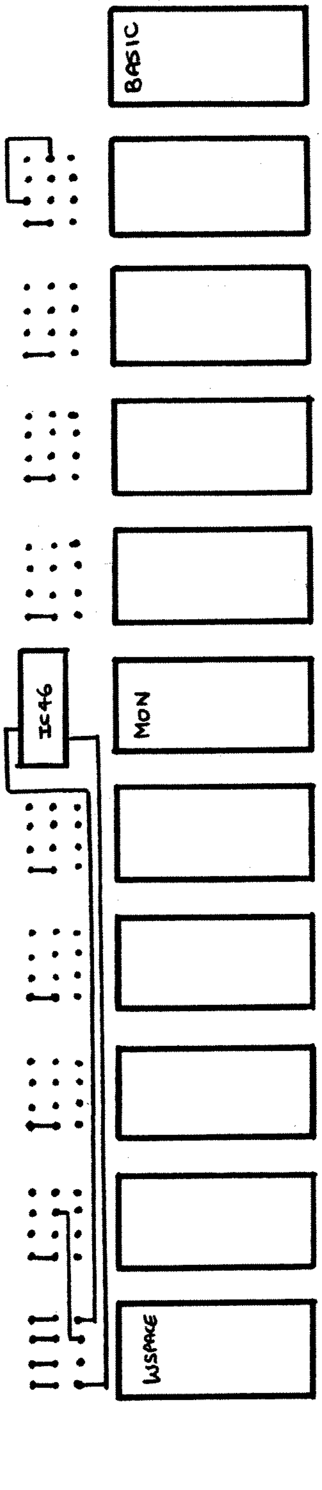


Fig 2 - Modified decode

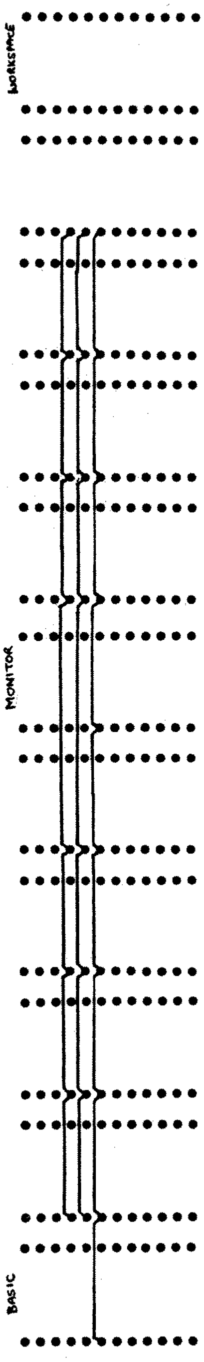


If we now turn to the N2 memory decoding you will see a simplified version in Fig 1. As you can see, the address decoder IC46 is only enabled by the decode PROM when /MREQ goes low. This is fine for the 2708s, as their /CS access time is 120ns, but is not suitable for connection to the /CE pin of the 2716 or 2732. The answer is to change the circuit configuration slightly as shown in Fig 2. Here the /CS decoding of IC46 is no longer qualified by /MREQ, and so as soon as an address appears on the address bus it is decoded to a /CE line in order to start the data access on that particuler EPROM. (Remember - this has to be done at least 450ns before we want the data). Subsequently, if and when /MREQ goes low, the high order address bits are decoded in the PROM (IC47). If it is a valid access to the EPROMs, then the Output Enable pin of all the EPROMs will be taken low, and the one that has been selected by its /CE line will output its data onto the Bus. (In this case the /OE signal has to appear at least 120ns before the data is required). If it is not an EPROM access, then the outputs are not enabled, and the only penalty paid is that one EPROM has been powered up but not used.

How to do it.

The following description will assume the modification is being performed on a standard N2 to which no exotic modifications have been made. Where the word TEST appears, it indicates that you should put the N2 back in its frame, power up, and check that Nas-Sys and Basic still run. If they don't, the reason for it should be easily identifiable. For the wiring on the bottom of the board I used a Vero wiring pen with the polyeurathane covered solderable enamel copper wire. Its easy to use, and makes a neat job of the modification.

Start by removing the N2 from its card frame and placing it in front of you. Remove any EPROMs already fitted in IC sockets 35-42. Remove any associated links from LKS1. (This should just have the standard 3, 1-16 2-15 8-9). Remove all links from the strapping fields LKB1-LKB8. TEST. Put the card component side down with the EPROM sockets along the bottom. Ignoring the workspace RAM at the right hand end, wire together all pin 19s of the byte-wide sockets including the Monitor and the Basic ROM. (This connects in A10.) See Fig. TEST. Next wire together all pin 20s, this time only do the 8 empty sockets, DO NOT INCLUDE THE MONITOR OR BASIC. TEST. Next wire together all pin 21s, once again only do the 8 empty sockets, DO NOT INCLUDE THE MONITOR OR BASIC. TEST. Turn the card over. On the eight link blocks link pin 4 to pin 8. (Connects /CS from decoder to /CE) - see Fig. TEST. Remove IC46 (LS155 decoder) from its socket and gently bend up pins 3 and 13. Solder two lengths of thin insulated wire to these pins.



Reinsert IC46 while making sure that pins 3 and 13 remain out of their respective holes. Connect the wire from pin 3 (decoder address A0) to pin 12 of workspace link block (LKB9). (Connects to A11). Connect the wire from pin 13 (decoder address A1) to pin 9 of workspace link block (LKB9). (Connects to A12). TEST. On LKB1 connect pin 5 to pin 3. (Connects all pin 21s to +5V). TEST. Connect LKB8 pin 5 to LKB9 pin 10. (Connects all /OE pins to the /XROM signal). TEST.

Now we come to setting up the decoding. What you do now is a matter of taste. What I did was: Ensure LSW1 switch 8 up, switch 7 down. On LKS1 connect 4 to 6 and on to TP13 (0V in bottom right corner) - this permanently enables the decoder IC46. Connect any two desired 4k blocks from their respective pins on LKS1 to pin 7 (/XROM). This gives a memory map of:

0000-07FF	Nas-Sys
0800-0FFF	Video+Workspace
1000-BFFF	Ram B(?)
C000-DFFF	4 x 2716 EPROMs
E000-FFFF	Basic.

Adding an external switch between IC6 pin 13 (or lower end of R56) and Ground allows either bank of four 2716s to be switched in to the C000-DFFF address range. If the full 16K is wanted on line together, then IC6 could be connected to A13 rather than a switch. In either case it might be advisable to lift the leg of IC6 from its socket and make the connection direct. If you don't, your system will crash if you absent mindedly close switch LSW1/7.

The modification for 2732 EPROMs is very similar. Instead of strapping pin 5 to pin 3 on LKB1 (to put 5V on the pins 21), extend the under-board wiring of the pins 21 to pin 8 of the Basic ROM to pick up A11. The other differences are concerned with the address decoding. Connect pin 3 of IC46 to LKB9 pin 9 (to pick up A12). I suggest that pin 13 of IC46 is connected to another switch. The two switches now let you select one of four 8k banks to occupy the selected 8k block of EPROM memory. Obviously the switches could be replaced by address lines as above. (In extremis this would give 32k of extra ROM - makes the N2 a little bit like the BBC, all ROM and no RAM!).

The more adventurous among you might like to replace the switches by a 74LS74 and some additional logic, and have software controlled selection of the EPROM banks via an IO port .... even more like the BBC!

Wait States

Returning briefly to the original question. Bill had already carried out the standard modification of only enabling Wait states during M1 cycles, not every memory cycle. (Looking at my N2 I see I did this by lifting the leg of

IC17 pin 2 - top right-hand corner of the board - and hard wiring it to pin 6 of IC9 - its immediate neighbour). He then tried ORing /M1 and the /XROM signal in an LS32, and using that to drive IC17/2, which was where failure set in. His problem lies in the accumualtion of delays within the circuit. Let's try a quick total up:

Clock falling edge to /MREQ low	- 85ns max out of Z80
/MREQ delay through buffer	- 12ns (?)
/XROM decoding delay thru IC47	- 25ns (typ)
Delay thru' LS32	- 14ns (typ)
D-type data setup time (IC17)	- 20ns
TOTAL	=156ns

Time available from clock falling edge to rising edge that clocks the wait state generator = 125ns (4MHz system in an ideal world). 156 is greater than 125, so failure! In practice the /MREQ delay is unlikely to be the full 85ns, and in fact his letter indicates that the /WAIT input does just make it low before the rising edge of the clock, but obviously not soon enough. The overall delay must be reduced, and one way is to do the address decoding separately (i.e. duplicate the PROM) and remove the dependence on /MREQ. The new PROM would be gated by /M1, and the LS32 would be discarded, the decoded output of the second prom being connected directly to IC17/2. (Alternatively an LS155, LS138 or similar decoder plus a few gates could be used in place of the PROM.) The timing figure would now be:

Clock rising edge to /M1 low	- 100ns max
Clock rising edge to Address valid	- 110ns max (NB 10ns Worse than above)
Delay through buffers	- 12ns
PROM (or logic) decoding delay	- 25ns
D-type data setup time	- 20ns
TOTAL	=167ns

Time available to rising edge of clock, 250ns. - plenty to spare!
Notice we have gained in two ways, first the decoding delays have been reduced as the LS32 is no longer used, and secondly everything is now refered to the rising edge of the system clock and we have an entire clock cycle available, rather than just half a cycle that was available previously.

Hopefully the above is of some use if all else fails, but I would first check the circuit modifications done for the 2716s in the hope that matters can be cured there.

TRAILER

I only seem to have covered one topic this time, but at least there isn't a mention of discs anywhere in it. Remember, the above is fueled by your letters, so write!

References:

1. ANDERSON E.P.T., "2K's on an N2", INMC80-5, Oct-Dec 1981, pp32-33
2. ANDERSON Paul, "16K CMOS RAM extension for the Nascom 2 main pcb", 80-BUS NEWS 1-3 July-Oct 1982, pp21-23
3. ROLLASON J., "2K 2716 EPROM & 6116 RAM for the Nascom 2 Main Board", ibid, pp28-29

Review of 12K BASIC for Nascoms

by G.S. Murray

Hisoft Pascals have, rightly, been the subject of much favourable review, but I have yet to see any comment on their 12K BASIC. Having had this BASIC for over two years and used it extensively during that time, the following comments may be of interest, particularly for those readers who still use cassette tapes rather than disks.

Hisoft supply their 12K BASIC on tape at 300 baud in the format required by their (4K) Nasmon monitor. That it needs the latter monitor has probably been a limiting factor in BAS12K's popularity. I was drawn to it by my need for greater arithmetical capability for payroll and similar programs that I was wanting to write in the days before there was a double precision package for Nascom ROM BASIC.

The Nasmon monitor occupies addresses 0000H to 07FFH and 1000H to 17FFH. I overcame the problem of 'abandoning' Nas-Sys by using a dual monitor board on my Nascom 1. The dual monitor board allows switching between the first half of Nasmon and Nas-Sys and, by using a double-pole switch, I am able to select between the monitors on the Nascom and between RAM and EPROM at 1000H to 17FFH on one of my RAM 'A' boards. My basic system, incidentally, consists of Nascom 1 and Buffer board with two RAM 'A' boards running at 4MHz without wait states; 64K RAM except where disabled by EPROM.

And what delights does BAS12K bring to justify all this trouble? Well, of course, it does calculations to 12 decimal places with up to 11 places displayed. The numeric display may be controlled by a PRECISION command which sets the number of places following the decimal point or by the sophisticated PRINT USING command. Its other commands, apart from those which it shares in common with most 8K BASICs, are somewhat similar to those recently described as being in the Microsoft disk BASIC, MBASIC, apart, that is, from the file handling routines.

But before commenting on those, I want to mention one feature which has proved extremely useful; the ability to use MID$ on the left-hand side of an assignment statement. This means that a slice of a string can easily be altered. Because, if the string is one read from a DATA statement incorporated in a program, the computer in READing the data merely creates a pointer to the program line in question, any amendment of the string concerned alters the DATA statement in the program. This will be SAVEd when the program itself is SAVEd and thus one has a ready means of amending stored data in the absence of formal file handling routines. If blank DATA lines of appropriate length are incorporated in the program they can be READ as blank strings which can later have necessary data put in them by the left-hand MID$(....) statement. Data are incorporated in program lines in a continuous string and chopped up by string manipulation rather than by using comma separators in DATA lines.

Programs are SAVEd and LOADed by name, which may be up to 10 characters long. An adverse feature attaches to SAVE command and that is that it is necessary to start the tape before pressing Enter to commence saving; the tape autostart doesn't operate before the name is sent to tape.

As well as the normal SAVE and LOAD commands, which use tokens for reserved words, there are the ASAVE and ALOAD commands which keep reserved words in their full ASCII form. Along with these two commands comes an AMERGE command which will load an ASAVEd program and merge it with an existing program already in the computer. Care in the use of this is necessary as the programs are just merged as they stand, so if line numbers are duplicated in the programs being merged, the resultant program will have more than one line with the same number.

Program lines may be generated and manipulated by AUTO, COPY, DELETE, EDIT & RENUMBER commands. AUTO will start at 10 with an interval of 10 unless otherwise instructed. COPY enables blocks of lines (or single lines) to be copied to elsewhere in a program. This command produces an error if copying would overwite existing lines. DELETE will delete a block of lines. Start and finish lines must exist or an error message will be generated. EDIT allows line editing of the up to 256 character lines which BAS12K supports. While not as convenient in some ways as screen editing, the editor is powerful and convenient once you become accustomed to it. Its commands are very similar to those outlined for MBASIC in 80BUS News for May-June 1983. RENUMBER renumbers the whole program, default start and interval 10, but other starts and intervals may be specified.

As well as REM, BAS12K supports the single quote and that quote suffices in place of the ":REM" which otherwise would be needed to seperate program from comment.

An EXCHANGE (swap) command allows string or numeric variables to be exchanged but only within their own types. This makes for convenient writing of alpha or numeric sort routines.

A TRACE command causes print out of line numbers as they are executed. The command takes a numeric non-zero argument to switch trace on and a zero argument to stop it. There is also an LTRACE commeand which directs the output to the serial port for a suitable printer.

Like TRACE, PRINT and PRINT USING have their version with preceeding 'L' to direct their output to printer.

Another useful additional command is LVAR (and its print version LLVAR) which lists all variables (other than array variables) and their values. One can STOP a program, examine the variables, alter one or more, and then CONTinue, provided you haven't altered the program.

BAS12K also features multi-line user-defined functions. Such functions are like Pascal functions in that local variables may be used in calculating the value to be returned to the main program.

While BAS12K does not have its own screen edit facility, it is possible to use the screen edit which forms part of the Nasmon monitor. From BAS12K this is entered via the SCREEN command. Text edited with the Nasmon editor may afterwards be converted back to BAS12K. The limitation on this option is that it may only be used if there is sufficient space in memory for the BASIC program to be stored in both formats at the same time. The Nasmon editor is essentially the editor now incorporated in Hisoft Pascal about which there has been considerable favourable comment from Dr Dark and others.

All in all, I have found BAS12K and the Nasmon monitor, which I had perforce to adopt to use the former, to be a very useful package. Having the Nasmon monitor, I also got the Nasgen assembler to go with it (Oh, how much faster than ZEAP!) and the same supplier's dis-assembler, Nasnem. This is a pure diassembler as Nasmon itself incorporates a 'front panel' system etc. to allow debugging of machine code programs. I must confess however that I have not done much machine code work with this monitor as I have not found it as easy to understand as Nas-Sys, maybe because it does not come with the fully commented source code such as accompanies Nas-Sys.

NASCOM ROM BASIC DIS—ASSEMBLED

PART 3

BY CARL LLOYD—PARKER


```
E668 FE20      PUTBUF: CP    " "           ; Is it a control code?
E66A DA10E6            JP    C,MORINP      ; Yes - Ignore
E66D 78        PUTCTL: LD    A,B           ; Get number of bytes in buffer
E66E FE49              CP    72+1          ; Test for line overflow
E670 3E07              LD    A,CTRLG       ; Set a bell
E672 D282E6            JP    NC,OUTNBS     ; Ring bell if buffer full
E675 79                LD    A,C           ; Get character
E676 71                LD    (HL),C        ; Save in buffer
E677 32CC10            LD    (LSTBIN),A    ; Save last input byte
E67A 23                INC   HL            ; Move up buffer
E67B 04                INC   B             ; Increment length
E67C CD9BE6    OUTIT:  CALL  OUTC          ; Output the character entered
E67F C310E6            JP    MORINP        ; Get another character

E682 CD9BE6    OUTNBS: CALL  OUTC          ; Output bell and back over it
E685 3E08              LD    A,BKSP        ; Set back space
E687 C37CE6            JP    OUTIT         ; Output it and get more

E68A 7C        CPDEHL: LD    A,H           ; Get H
E68B 92                SUB   D             ; Compare with D
E68C C0                RET   NZ            ; Different - Exit
E68D 7D                LD    A,L           ; Get L
E68E 93                SUB   E             ; Compare with E
E68F C9                RET                 ; Return status

E690 7E        CHKSYN: LD    A,(HL)        ; Check syntax of character
E691 E3                EX    (SP),HL       ; Address of test byte
E692 BE                CP    (HL)          ; Same as in code string?
E693 23                INC   HL            ; Return address
E694 E3                EX    (SP),HL       ; Put it back
E695 CA36E8            JP    Z,GETCHR      ; Yes - Get next character
E698 C3ADE3            JP    SNERR         ; Different - ?SN Error
```

```
E69B F5         OUTC:    PUSH  AF              ; Save character
E69C 3A4510              LD    A,(CTLOFG)      ; Get control "O" flag
E69F B7                  OR    A               ; Is it set?
E6A0 C245F2              JP    NZ,POPAF        ; Yes - don't output
E6A3 F1                  POP   AF              ; Restore character
E6A4 C5                  PUSH  BC              ; Save buffer length
E6A5 F5                  PUSH  AF              ; Save character
E6A6 FE20                CP    " "             ; Is it a control code?
E6A8 DABFE6              JP    C,DINPOS        ; Yes - Don't INC POS(X)
E6AB 3A4210              LD    A,(LWIDTH)      ; Get line width
E6AE 47                  LD    B,A             ; To B
E6AF 3AAB10              LD    A,(CURPOS)      ; Get cursor position
E6B2 04                  INC   B               ; Width 255?
E6B3 CABBE6              JP    Z,INCLEN        ; Yes - No width limit
E6B6 05                  DEC   B               ; Restore width
E6B7 B8                  CP    B               ; At end of line?
E6B8 CC81EB              CALL  Z,PRNTCR        ; Yes - output CRLF
E6BB 3C         INCLEN:  INC   A               ; Move on one character
E6BC 32AB10              LD    (CURPOS),A      ; Save new position
E6BF F1         DINPOS:  POP   AF              ; Restore character
E6C0 C1                  POP   BC              ; Restore buffer length
E6C1 F5                  PUSH  AF              ; << This sequence >>
E6C2 F1                  POP   AF              ; << is not needed >>
E6C3 F5                  PUSH  AF              ; Save character
E6C4 C5                  PUSH  BC              ; Save buffer length
E6C5 4F                  LD    C,A             ; Character to C
E6C6 CDD9FC              CALL  CONMON          ; Send it
E6C9 C1                  POP   BC              ; Restore buffer length
E6CA F1                  POP   AF              ; Restore character
E6CB C9                  RET

E6CC CD05FD     CLOTST:  CALL  GETINP          ; Get input character
E6CF E67F                AND   01111111B       ; Strip bit 7
E6D1 FEOF                CP    CTRLO           ; Is it control "O"?
E6D3 CO                  RET   NZ              ; No don't flip flag
E6D4 3A4510              LD    A,(CTLOFG)      ; Get flag
E6D7 2F                  CPL                   ; Flip it
E6D8 324510              LD    (CTLOFG),A      ; Put it back
E6DB AF                  XOR   A               ; Null character
E6DC C9                  RET
```



```
E6DD CDA5E9     LIST:    CALL  ATOH            ; ASCII number to DE
E6E0 CO                  RET   NZ              ; Return if anything extra
E6E1 C1                  POP   BC              ; Rubbish - Not needed
E6E2 CD99E4              CALL  SRCHLN          ; Search for line number in DE
E6E5 C5                  PUSH  BC              ; Save address of line
E6E6 CD33E7              CALL  SETLIN          ; Set up lines counter
E6E9 E1         LISTLP:  POP   HL              ; Restore address of line
E6EA 4E                  LD    C,(HL)          ; Get LSB of next line
E6EB 23                  INC   HL
E6EC 46                  LD    B,(HL)          ; Get MSB of next line
E6ED 23                  INC   HL
E6EE 78                  LD    A,B             ; BC = O (End of program)?
E6EF B1                  OR    C
E6F0 CAF8E3              JP    Z,PRNTOK        ; Yes - Go to command mode
E6F3 CD46E7              CALL  COUNT           ; Count lines
E6F6 CD61E8              CALL  TSTBRK          ; Test for break key
E6F9 C5                  PUSH  BC              ; Save address of next line
E6FA CD81EB              CALL  PRNTCR          ; Output CRLF
E6FD 5E                  LD    E,(HL)          ; Get LSB of line number
E6FE 23                  INC   HL
E6FF 56                  LD    D,(HL)          ; Get MSB of line number
E700 23                  INC   HL
E701 E5                  PUSH  HL              ; Save address of line start
E702 EB                  EX    DE,HL           ; Line number to HL
E703 CDADF9              CALL  PRNTHL          ; Output line number in decimal
E706 3E20                LD    A," "           ; Space after line number
E708 E1                  POP   HL              ; Restore start of line address
E709 CD9BE6     LSTLP2:  CALL  OUTC            ; Output character in A
E70C 7E         LSTLP3:  LD    A,(HL)          ; Get next byte in line
E70D B7                  OR    A               ; End of line?
E70E 23                  INC   HL              ; To next byte in line
E70F CAE9E6              JP    Z,LISTLP        ; Yes - get next line
E712 F209E7              JP    P,LSTLP2        ; No token - output it
E715 D67F                SUB   ZEND-1          ; Find and output word
E717 4F                  LD    C,A             ; Token offset+1 to C
E718 1143E1              LD    DE,WORDS        ; Reserved word list
E71B 1A         FNDTOK:  LD    A,(DE)          ; Get character in list
E71C 13                  INC   DE              ; Move on to next
E71D B7                  OR    A               ; Is it start of word?
E71E F21BE7              JP    P,FNDTOK        ; No - Keep looking for word
E721 OD                  DEC   C               ; Count words
E722 C21BE7              JP    NZ,FNDTOK       ; Not there - keep looking
E725 E67F       OUTWRD:  AND   01111111B       ; Strip bit 7
E727 CD9BE6              CALL  OUTC            ; Output first character
E72A 1A                  LD    A,(DE)          ; Get next character
E72B 13                  INC   DE              ; Move on to next
E72C B7                  OR    A               ; Is it end of word?
E72D F225E7              JP    P,OUTWRD        ; No - output the rest
E730 C30CE7              JP    LSTLP3          ; Next byte in line
```

```
E733 E5         SETLIN: PUSH  HL              ; Set up LINES counter
E734 2A4810             LD    HL,(LINESN)     ; Get LINES number
E737 224610             LD    (LINESC),HL     ; Save in LINES counter
E73A E1                 POP   HL
E73B C9                 RET

E73C 21DEFE     LDNMI1: LD    HL,BREAK        ; Break routine
E73F 227E0C             LD    (NMI),HL        ; NMI forces break
E742 C3F8E3             JP    PRNTOK          ; Go to command mode

E745 FE                 DEFB  (CP n)          ; <<< NO REFERENCE TO HERE >>>

E746 E5         COUNT:  PUSH  HL              ; Save code string address
E747 D5                 PUSH  DE
E748 2A4610             LD    HL,(LINESC)     ; Get LINES counter
E74B 11FFFF             LD    DE,-1
E74E ED5A               ADC   HL,DE           ; Decrement
E750 224610             LD    (LINESC),HL     ; Put it back
E753 D1                 POP   DE
E754 E1                 POP   HL              ; Restore code string address
E755 F0                 RET   P               ; Return if more lines to go
E756 E5                 PUSH  HL              ; Save code string address
E757 2A4810             LD    HL,(LINESN)     ; Get LINES number
E75A 224610             LD    (LINESC),HL     ; Reset LINES counter
E75D 3A4C10             LD    A,(NMIFLG)      ; Break by NMI?
E760 B7                 OR    A
E761 C2E5FE             JP    NZ,ARETN        ; Yes - "RETN"
E764 CD05FD             CALL  GETINP          ; Get input character
E767 FE03               CP    CTRLC           ; Is it control "C"?
E769 CA70E7             JP    Z,RSLNBK        ; Yes - Reset LINES an break
E76C E1                 POP   HL              ; Restore code string address
E76D C346E7             JP    COUNT           ; Keep on counting

E770 2A4810     RSLNBK: LD    HL,(LINESN)     ; Get LINES number
E773 224610             LD    (LINESC),HL     ; Reset LINES countr
E776 C3B1E0             JP    BRKRET          ; Go and output "Break"
```



```
E779 3E64       FOR:    LD    A,64H           ; Flag "FOR" assignment
E77B 32CB10             LD    (FORFLG),A      ; Save "FOR" flag
E77E CD87EA             CALL  LET             ; Set up initial index
E781 C1                 POP   BC              ; Drop RETurn address
E782 E5                 PUSH  HL              ; Save code string address
E783 CD70EA             CALL  DATA            ; Get next statement address
E786 22C710             LD    (LOOPST),HL     ; Save it for start of loop
E789 210200             LD    HL,2            ; Offset for "FOR" block
E78C 39                 ADD   HL,SP           ; Point to it
E78D CD5AE3     FORSLP: CALL  LOKFOR          ; Look for existing "FOR" block
E790 D1                 POP   DE              ; Get code string address
E791 C2A9E7             JP    NZ,FORFND       ; No nesting found
E794 09                 ADD   HL,BC           ; Move into "FOR" block
E795 D5                 PUSH  DE              ; Save code string address
E796 2B                 DEC   HL
E797 56                 LD    D,(HL)          ; Get MSB of loop statement
E798 2B                 DEC   HL
E799 5E                 LD    E,(HL)          ; Get LSB of loop statement
E79A 23                 INC   HL
E79B 23                 INC   HL
E79C E5                 PUSH  HL              ; Save block address
E79D 2AC710             LD    HL,(LOOPST)     ; Get address of loop statement
E7A0 CD8AE6             CALL  CPDEHL          ; Compare the FOR loops
E7A3 E1                 POP   HL              ; Restore block address
E7A4 C28DE7             JP    NZ,FORSLP       ; Different FORs - Find another
E7A7 D1                 POP   DE              ; Restore code string address
E7A8 F9                 LD    SP,HL           ; Remove all nested loops
```

```
E7A9 EB         FORFND: EX      DE,HL           ; Code string address to HL
E7AA 0E08               LD      C,8
E7AC CD8AE3             CALL    CHKSTK          ; Check for 8 levels of stack
E7AF E5                 PUSH    HL              ; Save code string address
E7B0 2AC710             LD      HL,(LOOPST)     ; Get first statement of loop
E7B3 E3                 EX      (SP),HL         ; Save and restore code string
E7B4 E5                 PUSH    HL              ; Re-save code string address
E7B5 2A5C10             LD      HL,(LINEAT)     ; Get current line number
E7B8 E3                 EX      (SP),HL         ; Save and restore code string
E7B9 CD44ED             CALL    TSTNUM          ; Make sure it's a number
E7BC CD90E6             CALL    CHKSYN          ; Make sure "TO" is next
E7BF A6                 DEFB    ZTO             ; "TO" token
E7C0 CD41ED             CALL    GETNUM          ; Get "TO" expression value
E7C3 E5                 PUSH    HL              ; Save code string address
E7C4 CD5FF8             CALL    BCDEFP          ; Move "TO" value to BCDE
E7C7 E1                 POP     HL              ; Restore code string address
E7C8 C5                 PUSH    BC              ; Save "TO" value in block
E7C9 D5                 PUSH    DE
E7CA 010081             LD      BC,8100H        ; BCDE = 1 (default STEP)
E7CD 51                 LD      D,C             ; C=0
E7CE 5A                 LD      E,D             ; D=0
E7CF 7E                 LD      A,(HL)          ; Get next byte in code string
E7D0 FEAB               CP      ZSTEP           ; See if "STEP" is stated
E7D2 3E01               LD      A,1             ; Sign of step = 1
E7D4 C2E5E7             JP      NZ,SAVSTP       ; No STEP given - Default to 1
E7D7 CD36E8             CALL    GETCHR          ; Jump over "STEP" token
E7DA CD41ED             CALL    GETNUM          ; Get step value
E7DD E5                 PUSH    HL              ; Save code string address
E7DE CD5FF8             CALL    BCDEFP          ; Move STEP to BCDE
E7E1 CD13F8             CALL    TSTSGN          ; Test sign of FPREG
E7E4 E1                 POP     HL              ; Restore code string address
E7E5 C5         SAVSTP: PUSH    BC              ; Save the STEP value in block
E7E6 D5                 PUSH    DE
E7E7 F5                 PUSH    AF              ; Save sign of STEP
E7E8 33                 INC     SP              ; Don't save flags
E7E9 E5                 PUSH    HL              ; Save code string address
E7EA 2ACE10             LD      HL,(BRKLIN)     ; Get address of index variable
E7ED E3                 EX      (SP),HL         ; Save and restore code string
E7EE 0681       PUTFID: LD      B,ZFOR          ; "FOR" block marker
E7F0 C5                 PUSH    BC              ; Save it
E7F1 33                 INC     SP              ; Don't save C
```



```
E7F2 CD40FD     RUNCNT: CALL    CHKBRK          ; Execution driver - Test break
E7F5 B7                 OR      A               ; Break key hit?
E7F6 C466E8             CALL    NZ,STALL        ; Yes - Pause for a key
E7F9 22CE10             LD      (BRKLIN),HL     ; Save code address for break
E7FC 7E                 LD      A,(HL)          ; Get next byte in code string
E7FD FE3A               CP      ":"             ; Multi statement line?
E7FF CA16E8             JP      Z,EXCUTE        ; Yes - Execute it
E802 B7                 OR      A               ; End of line?
E803 C2ADE3             JP      NZ,SNERR        ; No - Syntax error
E806 23                 INC     HL              ; Point to address of next line
E807 7E                 LD      A,(HL)          ; Get LSB of line pointer
E808 23                 INC     HL
E809 B6                 OR      (HL)            ; Is it zero (End of prog)?
E80A CA7AE8             JP      Z,ENDPRG        ; Yes - Terminate execution
E80D 23                 INC     HL              ; Point to line number
E80E 5E                 LD      E,(HL)          ; Get LSB of line number
E80F 23                 INC     HL
E810 56                 LD      D,(HL)          ; Get MSB of line number
E811 EB                 EX      DE,HL           ; Line number to HL
E812 225C10             LD      (LINEAT),HL     ; Save as current line number
E815 EB                 EX      DE,HL           ; Line number back to DE
E816 CD36E8     EXCUTE: CALL    GETCHR          ; Get key word
E819 11F2E7             LD      DE,RUNCNT       ; Where to RETurn to
E81C D5                 PUSH    DE              ; Save for RETurn
E81D C8         IFJMP:  RET     Z               ; Go to RUNCNT if end of STMT
E81E D680       ONJMP:  SUB     ZEND            ; Is it a token?
E820 DA87EA             JP      C,LET           ; No - try to assign it
E823 FE25               CP      ZNEW+1-ZEND     ; END to NEW ?
E825 D2ADE3             JP      NC,SNERR        ; Not a key word - ?SN Error
E828 07                 RLCA                    ; Double it
E829 4F                 LD      C,A             ; BC = Offset into table
E82A 0600               LD      B,0
E82C EB                 EX      DE,HL           ; Save code string address
E82D 215AE2             LD      HL,WORDTB       ; Keyword address table
E830 09                 ADD     HL,BC           ; Point to routine address
E831 4E                 LD      C,(HL)          ; Get LSB of routine address
E832 23                 INC     HL
E833 46                 LD      B,(HL)          ; Get MSB of routine address
E834 C5                 PUSH    BC              ; Save routine address
E835 EB                 EX      DE,HL           ; Restore code string address

E836 23         GETCHR: INC     HL              ; Point to next character
E837 7E                 LD      A,(HL)          ; Get next code string byte
E838 FE3A               CP      ":"             ; Z  if ":"
E83A D0                 RET     NC              ; NC if > "9"
E83B FE20               CP      " "
E83D CA36E8             JP      Z,GETCHR        ; Skip over spaces
E840 FE30               CP      "0"
E842 3F                 CCF                     ; NC if < "0"
E843 3C                 INC     A               ; Test for zero - Leave carry
E844 3D                 DEC     A               ; Z  if Null
E845 C9                 RET
```

```
E846 EB          RESTOR: EX    DE,HL           ; Save code string address
E847 2A5E10              LD    HL,(BASTXT)     ; Point to start of program
E84A CA5BE8              JP    Z,RESTNL        ; Just RESTORE - reset pointer
E84D EB                  EX    DE,HL           ; Restore code string address
E84E CDA5E9              CALL  ATOH            ; Get line number to DE
E851 E5                  PUSH  HL              ; Save code string address
E852 CD99E4              CALL  SRCHLN          ; Search for line number in DE
E855 60                  LD    H,B             ; HL = Address of line
E856 69                  LD    L,C
E857 D1                  POP   DE              ; Restore code string address
E858 D246EA              JP    NC,ULERR        ; ?UL Error if not found
E85B 2B          RESTNL: DEC   HL              ; Byte before DATA statement
E85C 22DC10      UPDATA: LD    (NXTDAT),HL     ; Update DATA pointer
E85F EB                  EX    DE,HL           ; Restore code string address
E860 C9                  RET

E861 CD40FD      TSTBRK: CALL  CHKBRK          ; Test for interrupts
E864 B7                  OR    A
E865 C8                  RET   Z               ; Return if no key pressed
E866 CDCCE6      STALL:  CALL  CLOTST          ; Get input and test for ^O
E869 FE13                CP    CTRLS           ; Is it control "S"
E86B CCCCE6              CALL  Z,CLOTST        ; Yes - Get another character
E86E FE03                CP    CTRLC           ; Return if not control "C"
E870 C0          STOP:   RET   NZ              ; Exit if anything else
E871 F6                  DEFB  (OR n)          ; Flag "STOP"
E872 C0          PEND:   RET   NZ              ; Exit if anything else
E873 22CE10              LD    (BRKLIN),HL     ; Save point of break
E876 21                  DEFB  (LD HL,nn)      ; Skip "OR 11111111B"
E877 F6FF        INPBRK: OR    11111111B       ; Flag "Break" wanted
E879 C1                  POP   BC              ; Return not needed and more
E87A 2A5C10      ENDPRG: LD    HL,(LINEAT)     ; Get current line number
E87D F5                  PUSH  AF              ; Save STOP / END status
E87E 7D                  LD    A,L             ; Is it direct break?
E87F A4                  AND   H
E880 3C                  INC   A               ; Line is -1 if direct break
E881 CA8DE8              JP    Z,NOLIN         ; Yes - No line number
E884 22D210              LD    (ERRLIN),HL     ; Save line of break
E887 2ACE10              LD    HL,(BRKLIN)     ; Get point of break
E88A 22D410              LD    (CONTAD),HL     ; Save point to CONTinue
E88D AF          NOLIN:  XOR   A
E88E 324510              LD    (CTLOFG),A      ; Enable output
E891 CD74EB              CALL  STTLIN          ; Start a new line
E894 F1                  POP   AF              ; Restore STOP / END status
E895 2150E3              LD    HL,BRKMSG       ; "Break" message
E898 C2E1E3              JP    NZ,ERRIN        ; "in line" wanted?
E89B C3F8E3              JP    PRNTOK          ; Go to command mode

E89E 2AD410      CONT:   LD    HL,(CONTAD)     ; Get CONTinue address
E8A1 7C                  LD    A,H             ; Is it zero?
E8A2 B5                  OR    L
E8A3 1E20                LD    E,CN            ; ?CN Error
E8A5 CAC1E3              JP    Z,ERROR         ; Yes - output "?CN Error"
E8A8 EB                  EX    DE,HL           ; Save code string address
E8A9 2AD210              LD    HL,(ERRLIN)     ; Get line of last break
E8AC 225C10              LD    (LINEAT),HL     ; Set up current line number
E8AF EB                  EX    DE,HL           ; Restore code string address
E8B0 C9                  RET                   ; CONTinue where left off
```



```
E8B1 CD84F4      NULL:   CALL  GETINT          ; Get integer 0-255
E8B4 C0                  RET   NZ              ; Return if bad value
E8B5 324110              LD    (NULLS),A       ; Set nulls number
E8B8 C9                  RET

E8B9 06FF        ARRLD1: LD    B,-1            ; Flag array load
E8BB CD36E8      ARRSV1: CALL  GETCHR          ; Skip "*"
E8BE 78                  LD    A,B             ; CLOAD* or CSAVE*
E8BF 32CE10              LD    (BRKLIN),A      ; Save it
E8C2 3E01                LD    A,1             ; It's an array
E8C4 32CB10              LD    (FORFLG),A      ; Flag array name
E8C7 CD2DEF              CALL  GETVAR          ; Get address of array name
E8CA E5                  PUSH  HL              ; Save code string address
E8CB 32CB10              LD    (FORFLG),A      ; Clear flag
E8CE 60                  LD    H,B             ; Address of array to HL
E8CF 69                  LD    L,C
E8D0 0B                  DEC   BC              ; Back space
E8D1 0B                  DEC   BC              ;  to point
E8D2 0B                  DEC   BC              ;  to the
E8D3 0B                  DEC   BC              ; array name
E8D4 3ACE10              LD    A,(BRKLIN)      ; CLOAD* or CSAVE* ?
E8D7 B7                  OR    A
E8D8 F5                  PUSH  AF              ; Save CLOAD* / CSAVE* status
E8D9 EB                  EX    DE,HL           ; Array data length
E8DA 19                  ADD   HL,DE           ; End of data
E8DB EB                  EX    DE,HL           ; To DE
E8DC 4E                  LD    C,(HL)          ; Get dimension bytes
E8DD 0600                LD    B,0
E8DF 09                  ADD   HL,BC           ; 2 Bytes each dimension
E8E0 09                  ADD   HL,BC
E8E1 23                  INC   HL              ; Over number of dimensions
E8E2 E5                  PUSH  HL              ; Address of array data
E8E3 D5                  PUSH  DE              ; End of array data
E8E4 C5                  PUSH  BC              ; Number of dimensions
E8E5 3ACE10              LD    A,(BRKLIN)      ; CLOAD* or CSAVE* ?
E8E8 FEFF                CP    -1
E8EA CCD5FC              CALL  Z,CASFF         ; CLOAD* - Cassette on
E8ED 3ACE10              LD    A,(BRKLIN)      ; CLOAD* or CSAVE* ?
E8F0 FEFF                CP    -1
E8F2 C4C8FC              CALL  NZ,CASFFW       ; CSAVE* - Cassette on and wait
E8F5 00                  NOP
E8F6 00                  NOP
E8F7 00                  NOP
E8F8 210000              LD    HL,0
E8FB 224A10              LD    (CHKSUM),HL     ; Zero check sum
E8FE C1                  POP   BC              ; Number of dimensions
E8FF D1                  POP   DE              ; End of array data
E900 E1                  POP   HL              ; Address of array data
E901 06D2                LD    B,11010010B     ; Header byte
E903 C3D6FF              JP    JPLDSV          ; CSAVE-SNDHDR , CLOAD-GETHDR

E906 78          SNDHDR: LD    A,B             ; Get header byte
E907 CDB7F4              CALL  WUART2          ; Send 2 bytes to UART
E90A CDB7F4              CALL  WUART2          ; Send 2 bytes to UART
E90D C31DE9              JP    SNDARY          ; Send array data
```

```
E910 0E04     GETHDR: LD    C,4            ; 4 Bytes to check
E912 CDB4F4   HDRLP:  CALL  RUART          ; Read byte from UART
E915 B8               CP    B              ; Same as header?
E916 C210E9           JP    NZ,GETHDR      ; No - Wait for another
E919 0D               DEC   C              ; Count bytes
E91A C212E9           JP    NZ,HDRLP       ; More needed
E91D CD44ED   SNDARY: CALL  TSTNUM         ; Check it's a numerical array
E920 CD8AE6   ARYLP:  CALL  CPDEHL         ; All array data done
E923 CA37E9           JP    Z,SUMOFF       ; Yes - Do check sum
E926 F1               POP   AF             ; CLOAD* or CSAVE* ?
E927 F5               PUSH  AF             ; Re-save flags
E928 7E               LD    A,(HL)         ; Get byte
E929 F4BAF4           CALL  P,WUART        ; CSAVE* - Write byte
E92C FCB4F4           CALL  M,RUART        ; CLOAD* - Read byte
E92F 77               LD    (HL),A         ; Save byte in case of CLOAD*
E930 CD40E9           CALL  ACCSUM         ; Accumulate check sum
E933 23               INC   HL             ; Next byte
E934 C320E9           JP    ARYLP          ; Repeat

E937 CD4DE9   SUMOFF: CALL  DOSUM          ; Do check sum
E93A CDD5FC           CALL  CASFF          ; Cassette off
E93D F1               POP   AF             ; Not needed any more
E93E E1               POP   HL             ; Restore code string address
E93F C9               RET

E940 E5       ACCSUM: PUSH  HL             ; Save address in array
E941 2A4A10           LD    HL,(CHKSUM)    ; Get check sum
E944 0600             LD    B,0            ; BC = Value of byte
E946 4F               LD    C,A
E947 09               ADD   HL,BC          ; Add byte to check sum
E948 224A10           LD    (CHKSUM),HL    ; Re-save check sum
E94B E1               POP   HL             ; Restore address in array
E94C C9               RET

E94D 3ACE10   DOSUM:  LD    A,(BRKLIN)     ; CLOAD* or CSAVE* ?
E950 B7               OR    A
E951 FA60E9           JP    M,CHSUMS       ; CLOAD* - Check if sums match
E954 3A4A10           LD    A,(CHKSUM)     ; Get LSB of check sum
E957 CDBAF4           CALL  WUART          ; Write to UART
E95A 3A4B10           LD    A,(CHKSUM+1)   ; Get MSB of check sum
E95D C3BAF4           JP    WUART          ; Write to UART and return

E960 CDB4F4   CHSUMS: CALL  RUART          ; Read LSB of check sum
E963 F5               PUSH  AF             ; Save it
E964 CDB4F4           CALL  RUART          ; Read MSB of check sum
E967 C1               POP   BC             ; LSB to B
E968 58               LD    E,B            ; LSB to E
E969 57               LD    D,A            ; MSB to D
E96A 2A4A10           LD    HL,(CHKSUM)    ; Get accumulated check sum
E96D CD8AE6           CALL  CPDEHL         ; Are they the same?
E970 C8               RET   Z              ; Yes - End CLOAD*
E971 CDD5FC           CALL  CASFF          ; Cassette off
E974 C36BF5           JP    OUTBAD         ; Different - Output "Bad"
```



```
E977 7E       CHKLTR: LD    A,(HL)         ; Get byte
E978 FE41             CP    "A"            ; < "A" ?
E97A D8               RET   C              ; Carry set if not letter
E97B FE5B             CP    "Z"+1          ; > "Z" ?
E97D 3F               CCF
E97E C9               RET                  ; Carry set if not letter

E97F CD36E8   FPSINT: CALL  GETCHR         ; Get next character
E982 CD41ED   POSINT: CALL  GETNUM         ; Get integer 0 to 32767
E985 CD13F8   DEPINT: CALL  TSTSGN         ; Test sign of FPREG
E988 FAA0E9           JP    M,FCERR        ; Negative - ?FC Error
E98B 3AE710   DEINT:  LD    A,(FPEXP)      ; Get integer value to DE
E98E FE90             CP    80H+16         ; Exponent in range (16 bits)?
E990 DABBF8           JP    C,FPINT        ; Yes - convert it
E993 018090           LD    BC,9080H       ; BCDE = -32768
E996 110000           LD    DE,0000
E999 E5               PUSH  HL             ; Save code string address
E99A CD8EF8           CALL  CMPNUM         ; Compare FPREG with BCDE
E99D E1               POP   HL             ; Restore code string address
E99E 51               LD    D,C            ; MSB to D
E99F C8               RET   Z              ; Return if in range
E9A0 1E08     FCERR:  LD    E,FC           ; ?FC Error
E9A2 C3C1E3           JP    ERROR          ; Output error.

E9A5 2B       ATOH:   DEC   HL             ; ASCII number to DE binary
E9A6 110000   GETLN:  LD    DE,0           ; Get number to DE
E9A9 CD36E8   GTLNLP: CALL  GETCHR         ; Get next character
E9AC D0               RET   NC             ; Exit if not a digit
E9AD E5               PUSH  HL             ; Save code string address
E9AE F5               PUSH  AF             ; Save digit
E9AF 219819           LD    HL,65529/10    ; Largest number 65529
E9B2 CD8AE6           CALL  CPDEHL         ; Number in range?
E9B5 DAADE3           JP    C,SNERR        ; No - ?SN Error
E9B8 62               LD    H,D            ; HL = Number
E9B9 6B               LD    L,E
E9BA 19               ADD   HL,DE          ; Times 2
E9BB 29               ADD   HL,HL          ; Times 4
E9BC 19               ADD   HL,DE          ; Times 5
E9BD 29               ADD   HL,HL          ; Times 10
E9BE F1               POP   AF             ; Restore digit
E9BF D630             SUB   "0"            ; Make it 0 to 9
E9C1 5F               LD    E,A            ; DE = Value of digit
E9C2 1600             LD    D,0
E9C4 19               ADD   HL,DE          ; Add to number
E9C5 EB               EX    DE,HL          ; Number to DE
E9C6 E1               POP   HL             ; Restore code string address
E9C7 C3A9E9           JP    GTLNLP         ; Go to next character
```

```
E9CA CAC9E4     CLEAR:  JP      Z,INTVAR        ; Just "CLEAR" Keep parameters
E9CD CD82E9             CALL    POSINT          ; Get integer 0 to 32767 to DE
E9D0 2B                 DEC     HL              ; Cancel increment
E9D1 CD36E8             CALL    GETCHR          ; Get next character
E9D4 E5                 PUSH    HL              ; Save code string address
E9D5 2AAF10             LD      HL,(LSTRAM)     ; Get end of RAM
E9D8 CAEDE9             JP      Z,STORED        ; No value given - Use stored
E9DB E1                 POP     HL              ; Restore code string address
E9DC CD90E6             CALL    CHKSYN          ; Check for comma
E9DF 2C                 DEFB    ","
E9E0 D5                 PUSH    DE              ; Save number
E9E1 CD82E9             CALL    POSINT          ; Get integer 0 to 32767
E9E4 2B                 DEC     HL              ; Cancel increment
E9E5 CD36E8             CALL    GETCHR          ; Get next character
E9E8 C2ADE3             JP      NZ,SNERR        ; ?SN Error if more on line
E9EB E3                 EX      (SP),HL         ; Save code string address
E9EC EB                 EX      DE,HL           ; Number to DE
E9ED 7D         STORED: LD      A,L             ; Get LSB of new RAM top
E9EE 93                 SUB     E               ; Subtract LSB of string space
E9EF 5F                 LD      E,A             ; Save LSB
E9F0 7C                 LD      A,H             ; Get MSB of new RAM top
E9F1 9A                 SBC     A,D             ; Subtract MSB of string space
E9F2 57                 LD      D,A             ; Save MSB
E9F3 DAA2E3             JP      C,OMERR         ; ?OM Error if not enough mem
E9F6 E5                 PUSH    HL              ; Save RAM top
E9F7 2AD610             LD      HL,(PROGND)     ; Get program end
E9FA 012800             LD      BC,40           ; 40 Bytes minimum working RAM
E9FD 09                 ADD     HL,BC           ; Get lowest address
E9FE CD8AE6             CALL    CPDEHL          ; Enough memory?
EA01 D2A2E3             JP      NC,OMERR        ; No - ?OM Error
EA04 EB                 EX      DE,HL           ; RAM top to HL
EA05 225A10             LD      (STRSPC),HL     ; Set new string space
EA08 E1                 POP     HL              ; End of memory to use
EA09 22AF10             LD      (LSTRAM),HL     ; Set new top of RAM
EA0C E1                 POP     HL              ; Restore code string address
EA0D C3C9E4             JP      INTVAR          ; Initialise variables
```



```
EA10 CAC5E4     RUN:    JP      Z,RUNFST        ; RUN from start if just RUN
EA13 CDC9E4             CALL    INTVAR          ; Initialise variables
EA16 01F2E7             LD      BC,RUNCNT       ; Execution driver loop
EA19 C32CEA             JP      RUNLIN          ; RUN from line number

EA1C 0E03       GOSUB:  LD      C,3             ; 3 Levels of stack needed
EA1E CD8AE3             CALL    CHKSTK          ; Check for 3 levels of stack
EA21 C1                 POP     BC              ; Get return address
EA22 E5                 PUSH    HL              ; Save code string for RETURN
EA23 E5                 PUSH    HL              ; And for GOSUB routine
EA24 2A5C10             LD      HL,(LINEAT)     ; Get current line
EA27 E3                 EX      (SP),HL         ; Into stack - Code string out
EA28 3E8C               LD      A,ZGOSUB        ; "GOSUB" token
EA2A F5                 PUSH    AF              ; Save token
EA2B 33                 INC     SP              ; Don't save flags

EA2C C5         RUNLIN: PUSH    BC              ; Save return address
EA2D CDA5E9     GOTO:   CALL    ATOH            ; ASCII number to DE binary
EA30 CD72EA             CALL    REM             ; Get end of line
EA33 E5                 PUSH    HL              ; Save end of line
EA34 2A5C10             LD      HL,(LINEAT)     ; Get current line
EA37 CD8AE6             CALL    CPDEHL          ; Line after current?
EA3A E1                 POP     HL              ; Restore end of line
EA3B 23                 INC     HL              ; Start of next line
EA3C DC9CE4             CALL    C,SRCHLP        ; Line is after current line
EA3F D499E4             CALL    NC,SRCHLN       ; Line is before current line
EA42 60                 LD      H,B             ; Set up code string address
EA43 69                 LD      L,C
EA44 2B                 DEC     HL              ; Incremented after
EA45 D8                 RET     C               ; Line found
EA46 1E0E       ULERR:  LD      E,UL            ; ?UL Error
EA48 C3C1E3             JP      ERROR           ; Output error message

EA4B C0         RETURN: RET     NZ              ; Return if not just RETURN
EA4C 16FF               LD      D,-1            ; Flag "GOSUB" search
EA4E CD56E3             CALL    BAKSTK          ; Look "GOSUB" block
EA51 F9                 LD      SP,HL           ; Kill all FORs in subroutine
EA52 FE8C               CP      ZGOSUB          ; Test for "GOSUB" token
EA54 1E04               LD      E,RG            ; ?RG Error
EA56 C2C1E3             JP      NZ,ERROR        ; Error if no "GOSUB" found
EA59 E1                 POP     HL              ; Get RETURN line number
EA5A 225C10             LD      (LINEAT),HL     ; Save as current
EA5D 23                 INC     HL              ; Was it from direct statement?
EA5E 7C                 LD      A,H
EA5F B5                 OR      L
EA60 C26AEA             JP      NZ,RETLIN       ; No - Return to line
EA63 3ACC10             LD      A,(LSTBIN)      ; Any INPUT in subroutine?
EA66 B7                 OR      A               ; If so buffer is corrupted
EA67 C2F7E3             JP      NZ,POPNOK       ; Yes - Go to command mode
EA6A 21F2E7     RETLIN: LD      HL,RUNCNT       ; Execution driver loop
EA6D E3                 EX      (SP),HL         ; Into stack - Code string out
EA6E 3E                 DEFB    (LD A,n)        ; Skip "POP HL"
EA6F E1         NXTDTA: POP     HL              ; Restore code string address
```

```
EA70 013A       DATA:    DEFB   (LD BC,":")     ; ":" End of statement
EA72 0E00       REM:     LD     C,0             ; 00  End of statement
EA74 0600                LD     B,0
EA76 79         NXTSTL:  LD     A,C             ; Statement end byte
EA77 48                  LD     C,B
EA78 47                  LD     B,A             ; Statement end byte
EA79 7E         NXTSTT:  LD     A,(HL)          ; Get byte
EA7A B7                  OR     A               ; End of line?
EA7B C8                  RET    Z               ; Yes - Exit
EA7C B8                  CP     B               ; End of statement?
EA7D C8                  RET    Z               ; Yes - Exit
EA7E 23                  INC    HL              ; Next byte
EA7F FE22                CP     '"'             ; Literal string?
EA81 CA76EA              JP     Z,NXTSTL        ; Yes - Look for another '"'
EA84 C379EA              JP     NXTSTT          ; Keep looking

EA87 CD2DEF     LET:     CALL   GETVAR          ; Get variable name
EA8A CD90E6              CALL   CHKSYN          ; Make sure "=" follows
EA8D B4                  DEFB   ZEQUAL          ; "=" token
EA8E D5                  PUSH   DE              ; Save address of variable
EA8F 3AAD10              LD     A,(TYPE)        ; Get data type
EA92 F5                  PUSH   AF              ; Save type
EA93 CD5AED              CALL   EVAL            ; Evaluate expression
EA96 F1                  POP    AF              ; Restore type
EA97 E3                  EX     (SP),HL         ; Save code - Get var addr
EA98 22CE10              LD     (BRKLIN),HL     ; Save address of variable
EA9B 1F                  RRA                    ; Adjust type
EA9C CD46ED              CALL   CHKTYP          ; Check types are the same
EA9F CADAEA              JP     Z,LETNUM        ; Numeric - Move value
EAA2 E5         LETSTR:  PUSH   HL              ; Save address of string var
EAA3 2AE410              LD     HL,(FPREG)      ; Pointer to string entry
EAA6 E5                  PUSH   HL              ; Save it on stack
EAA7 23                  INC    HL              ; Skip over length
EAA8 23                  INC    HL
EAA9 5E                  LD     E,(HL)          ; LSB of string address
EAAA 23                  INC    HL
EAAB 56                  LD     D,(HL)          ; MSB of string address
EAAC 2A5E10              LD     HL,(BASTXT)     ; Point to start of program
EAAF CD8AE6              CALL   CPDEHL          ; Is string before program?
EAB2 D2C9EA              JP     NC,CRESTR       ; Yes - Create string entry
EAB5 2A5A10              LD     HL,(STRSPC)     ; Point to string space
EAB8 CD8AE6              CALL   CPDEHL          ; Is string literal in program?
EABB D1                  POP    DE              ; Restore address of string
EABC D2D1EA              JP     NC,MVSTPT       ; Yes - Set up pointer
EABF 21BF10              LD     HL,TMPSTR       ; Temporary string pool
EAC2 CD8AE6              CALL   CPDEHL          ; Is string in temporary pool?
EAC5 D2D1EA              JP     NC,MVSTPT       ; No - Set up pointer
EAC8 3E                  DEFB   (LD A,n)        ; Skip "POP DE"
EAC9 D1         CRESTR:  POP    DE              ; Restore address of string
EACA CD71F3              CALL   BAKTMP          ; Back to last tmp-str entry
EACD EB                  EX     DE,HL           ; Address of string entry
EACE CDAAF1              CALL   SAVSTR          ; Save string in string area
EAD1 CD71F3     MVSTPT:  CALL   BAKTMP          ; Back to last tmp-str entry
EAD4 E1                  POP    HL              ; Get string pointer
EAD5 CD6EF8              CALL   DETHL4          ; Move string pointer to var
EAD8 E1                  POP    HL              ; Restore code string address
EAD9 C9                  RET
```



```
EADA E5         LETNUM:  PUSH   HL              ; Save address of variable
EADB CD6BF8              CALL   FPTHL           ; Move value to variable
EADE D1                  POP    DE              ; Restore address of variable
EADF E1                  POP    HL              ; Restore code string address
EAE0 C9                  RET

EAE1 CD84F4     ON:      CALL   GETINT          ; Get integer 0-255
EAE4 7E                  LD     A,(HL)          ; Get "GOTO" or "GOSUB" token
EAE5 47                  LD     B,A             ; Save in B
EAE6 FE8C                CP     ZGOSUB          ; "GOSUB" token?
EAE8 CAF0EA              JP     Z,ONGO          ; Yes - Find line number
EAEB CD90E6              CALL   CHKSYN          ; Make sure it's "GOTO"
EAEE 88                  DEFB   ZGOTO           ; "GOTO" token
EAEF 2B                  DEC    HL              ; Cancel increment
EAF0 4B         ONGO:    LD     C,E             ; Integer of branch value
EAF1 0D         ONGOLP:  DEC    C               ; Count branches
EAF2 78                  LD     A,B             ; Get "GOTO" or "GOSUB" token
EAF3 CA1EE8              JP     Z,ONJMP         ; Go to that line if right one
EAF6 CDA6E9              CALL   GETLN           ; Get line number to DE
EAF9 FE2C                CP     ","             ; Another line number?
EAFB C0                  RET    NZ              ; No - Drop through
EAFC C3F1EA              JP     ONGOLP          ; Yes - loop

EAFF CD5AED     IF:      CALL   EVAL            ; Evaluate expression
EB02 7E                  LD     A,(HL)          ; Get token
EB03 FE88                CP     ZGOTO           ; "GOTO" token?
EB05 CA0DEB              JP     Z,IFGO          ; Yes - Get line
EB08 CD90E6              CALL   CHKSYN          ; Make sure it's "THEN"
EB0B A9                  DEFB   ZTHEN           ; "THEN" token
EB0C 2B                  DEC    HL              ; Cancel increment
EB0D CD44ED     IFGO:    CALL   TSTNUM          ; Make sure it's numeric
EB10 CD13F8              CALL   TSTSGN          ; Test state of expression
EB13 CA72EA              JP     Z,REM           ; False - Drop through
EB16 CD36E8              CALL   GETCHR          ; Get next character
EB19 DA2DEA              JP     C,GOTO          ; Number - GOTO that line
EB1C C31DE8              JP     IFJMP           ; Otherwise do statement
```

Review of the Gemini Galaxy 2

Introduction

The Galaxy 2 computer by Gemini Microcomputers is the result of progressive development from the earlier Galaxy 1 computer, incorporating a number of improvements which further enhance the performance of the machine. Gemini have retained the modular card approach with the Galaxy 2, an approach which has always been the province of 'professional' computer design. The 'single board' approach adopted by many computer designers has advantages in terms of production cost, but disadvantages in terms of flexibility, expansion and repair. The single board approach is probably suitable for the business user where, once the specification of the machine has been decided, there would be little likelihood of needing expansion. However, Gemini have produced a modular machine equal to or better in performance than most popular single board computers, and at a very competitive price. Because of its modular design many permutations of the basic machine may be manufactured without the extra cost involved in manufacturing 'specials', and with all the advantages of expansion and flexibility as we shall see later. This makes the Galaxy 2 not only competitive in the business market, but ideal in the system development or laboratory situation, an area where many single board computers could not compete.

Specification

The basic specification of the machine is similar to many other machines using the CP/M operating system: a high capacity twin disk system, Z80A processor, 64K of RAM, 80 by 25 VDU type display, etc. Unlike many other CP/M machines, however, the Galaxy 2 has an internal bus with three slots spare conforming to the popular 80-BUS/Nasbus standard, allowing expansion with cards manufactured by a number of other companies.

Appearance

The Galaxy comes complete with all leads and necessary connectors to plug in directly. A quality 12" green or amber screen monitor is also supplied. The Galaxy case is steel measuring 18" by 12" by 7", attractively finished in cream and matt black paint. The paint finish is a high temperature baked resin finish and is particularly hard wearing, very easy to keep clean and not prone to scratches. In these days of flimsy vacuum formed plastic cases it is a delight to see the solid and robust case of the Galaxy, which whilst boxy in shape is both pleasantly and conveniently proportioned. The solid case makes the Galaxy feature as one of the very few computers you could actually stand on without damage. Not a major sales point perhaps, but one that should appeal to the educationalists or to users where hard physical use (or abuse) of a computer takes place. The separate keyboard case is also of steel construction, finished in a similar fashion to the main computer case, and is sufficiently low profile for experienced typists to find it comfortable in use

The only features on the front face of the computer are the vertically mounted twin disk drives and an LED power on indicator. The back panel has all the necessary sockets and connectors. A DB25 connector is provided for direct connection to RS232 printers or a modem, a 36 way Amphenol type connector for direct connection to printers using the parallel 'Centronics' protocol. A DB15 connector is provided for keyboard connection and DIN sockets for external tape I/O and light pen connection and a PL259 socket for the video output. The power and reset switches are also mounted on the back panel, along with the quick release fan filter and a blanked off hole approximately 1" by 2.5" to allow connection to be made to any expansion cards fitted by the user which require leads to external equipment, etc. All connectors on the back panel are clearly labelled and an adequate description of each is provided in the manual.

The Keyboard

The keyboard is separately detachable and mounted in a steel case. Connection is made by a multiway 36" coiled cable which expands to about 80". This makes for neat and tidy connection, but unfortunately the connector on the keyboard is on the right hand side of back panel of the keyboard case which means that if the cable is laid 'naturally', it runs from the left hand side of the computer across to the keyboard connector on the other side of the keyboard case. This means that more of the cable is in view than would be strictly necessary had the keyboard connector been fitted to the left hand side. The keyboard is of high quality with a light yet positive touch and is a full function keyboard with a standard QWERTY layout. Ten function keys, four cursor control keys and a numeric pad are also provided. An unusual feature of the function keys is that the cursor and numeric pad keys may also be programmed as function keys. Additionally, the functions can also be shifted so that each programmable key may serve a dual function. This gives access to no less than 60 fully programmable function keys if required. This feature would commonly be used with programs such as WORDSTAR, where the numeric pad would be reprogrammed as the WORDSTAR cursor keys. The only criticism of the keyboard as such is the cursor keys themselves which are, in my opinion, badly positioned and psychologically in the wrong order. The order could be changed by reprogramming but this should not be necessary on a machine like the Galaxy. Programming the function keys from the keyboard is simple from within the CP/M command mode (and often from within programs) and a utility program is provided to store the reprogrammed key patterns for instant recall at a later date.

The Disk Drives

The drives fitted as standard are the Micropolis 1115F-V 96 track per inch, single sided 80 track drives working in a double density MFM format. These drives are extremely solidly built and use such advanced techniques as phase locked motor speed control and an internal dedicated processor. They are widely recognised as being one of the best and most reliable drives available. The Micropolis drives give a formatted space of 400K, which, when system tracks and directory areas are taken into account, leave the user with 388K of useable disk space per drive. A factory option is to fit the Micropolis 1115F-VI drives which are of similar specification but double sided, providing a useable space of 788K per drive. The drives are loaded in the conventional manner with flap doors; no spring disk eject mechanism is fitted. A small point is that with the drives mounted vertically, there is no obvious 'up' when inserting disks in the disk drives, and although obvious to the experienced user, it is only too easy for the inexperienced to put a disk into the drive 'upside down' and to suffer a failure of the system to boot. A point not covered in the manual.

An interesting point if the double sided 1115F-VI drives are fitted is that the Galaxy could be used to read disks from many other machines (including Gemini's earlier double density machines). As the drive head step pitch is the now standard 96 tracks per inch, and as the standard pitch of many double density machines is 48 tracks per inch, it is apparent that if the Micropolis drives were arranged to double step, then the data could be read. This of course would require software patches in the low level disk primitives in the Gemini BIOS which would not be the easiest thing to undertake but not outside the scope of many proficient machine code programmers. Note that Gemini are not likely to provide support for this sort thing unless demand warrants it, but it might be worth asking if only to indicate that the demand existed.

Powering up

The Galaxy 2 auto boots on power up when a system disk is inserted in drive A, an almost foolproof idea accompanied by a flashing instruction, displayed in inverse video, on the monitor which says 'Insert disk in drive A'. Should the disk system fail to boot because a disk with no system or incorrect system was inserted, then a message to that effect is displayed. If a correct disk is inserted and the system fails to boot then it is possible to cause the Galaxy to revert to the very simple internal ROM monitor which is capable of carrying out diagnostic procedures. Only the bare essentials of the diagnostics are covered in the manual and for more sophisticated checks it is necessary to revert to the individual card manuals which are available from Gemini as an optional extra. In any event, unless the user has a very thorough working knowledge of the internals of the machine, he would be advised to consult Gemini through their country wide dealer network if problems are encountered.

Video Output

Computers of modular design are often associated with separate terminals as display devices, not so with the Galaxy. A separate card takes care of the video and feeds its output to the back panel socket for connection direct to a standard monitor display. The 25 line by 80 column display is to professional standards and the video card contains within its command set a sub-set which emulates the Lear Siegler ADM-3A terminal. However, whilst the Lear Siegler terminal is a serial device and hence somewhat slow, the video card plugs straight on to the system bus and is commendably fast. One reason for the choice of the Lear Siegler instruction sub-set is that much popular CP/M software is designed for use with this particular terminal and very little, if any, patching is usually required. However, the video card improves on the Lear Siegler instruction set in many ways, and much proprietary software can be further improved as regards speed by the incorporation of the extended commands of the video card. The video card also features a user programmable character set and also limited 160 by 75 pixel graphics.

The Galaxy is supplied with a standard monitor, the Phoenix P12, a 12" green or amber screen monitor of Italian origin. Unlike the Galaxy, this has a plastic case, in a colour which matches the colour scheme of the Galaxy. The Phoenix monitor has logic rather than analogue inputs and is therefore free from that faint patterning which often accompanies analogue monitors although this was just discernable when displaying inverse video. The monitor was extremely sharp to the edges and almost free of picture 'breathing' when switching rapidly from normal to inverse video. Picture geometry was excellent. When supplied the monitor picture was a little undersize and slightly to one side. Not many users would notice this, although it is easily adjusted if required.

Internal Layout

Internally the layout of the Galaxy is simple and compact. At the rear of the case is the fan and power supply unit. The fan ran reasonably quietly and was certainly an improvement over the fans fitted to earlier Galaxy 1's. The power supply is a switch mode one, fully RF screened and with impressive mains supply filtering. No effects of mains glitches were observed even under the 'dirtiest' of mains conditions. The drives are fitted to the front panel immediately in front of the power supply and solidly secured to a sub-chassis on the base of the computer case. To the left of the drives is a card cage which contains six card slots. The three lower slots contain the three modular cards needed for the standard Galaxy, the upper three are empty and are available for expansion. As mentioned earlier a blanked off hole is provided on the back panel adjacent to the card cage to allow entry for leads which might need to be connected to any envisaged expansion cards. Air flow through the case is dictated by the position of the ventilation slots on the sides and to the front of the case top wrapper. There are no slots on the top of the case, so the computer is reasonably safe against spilt coffee, etc. Having removed the top cover, it is only too easy to replace it the wrong way round and so restrict the ventilation although no harm seems come if this accidentally happens, indicating more than adequate cooling.

The standard Galaxy is fitted with three cards, the CPU/RAM card, the disk controller card and the video card. As is to be expected, all three cards are engineered to the highest standards, the pc layout whilst compact, is orderly and neat, with gold flashed connectors, pc solder resist and component legend screening on the fibreglass boards. The boards are flow soldered with the ICs socketed where necessary. Good quality components are used throughout

The CPU Card

The CPU card contains the main processor, a 4MHz Z80A. The clock speed is 4MHz without wait states. The board also incorporates 64K of 4164 dynamic RAM and the 2K simple monitor/boot EPROM. A Z80A PIO device takes care of the parallel I/O, normally used for the 'Centronics' protocol parallel interface, whilst a WD8250 serial controller provides RS232 I/O and full modem control. A number of links are provided for reconfiguring the board for specialist purposes which are fully detailed in an optional manual for the card. All I/O to the bus is fully buffered with considerable expansion in mind. The RAM and ROM are paged and can be selected and deselected from software. The software is so arranged that the EPROM is normally paged out of the system on boot up, leaving the whole of the 64K RAM available for CP/M. The RAM is similarly paged, with extended addressing to 2M byte. This means that implementation of CP/M Plus (CP/M 3.x) will present no particular problems as the additional RAM required for this operating system could be provided by the inclusion of the Gemini GM802 64K expansion RAM card, which already exists within the Gemini range of expansion cards.

The Disk Controller Card

The disk controller card features the Western Digital WD1797 three chip disk controller set, and incorporates software controlled clock rate select for 5.25" or 8" drives. Connectors are provided for the standard 5.25" and 8" drive connectors, whilst a third connector is used for the SASI interface to Winchester hard disks. This card is extremely flexible in its own right, although normally only used to drive the internal 5.25" drives. 8" drives and a Winchester drive can be fitted externally and because of the software selectable clock rates and the SASI interface, given the correct software, they can all be used together from the one card!! A further extension to the disk system is the option of fitting Gemini GM833 512K RAM-DISK cards which are organised as a port driven memory plane (not page mode). This is intended to be used exactly like an additional 512K disk drive (or more with multiple cards), and the Gemini CP/M BIOS is designed to cope with this. The disadvantage of course is that data is lost on power down, so good discipline is required to ensure that the virtual disk is always adequately backed up. However, in use the virtual disk card is incredibly fast and is a boon when used in large data base programs where updating large disk based indices inevitably slows down work. Naturally the virtual disk may be used with the existing 5.25" drives, and of course the 8" and Winchester drives as well. A truly flexible configuration.

The Video Card

The video card, as already mentioned, handles all the video processing under the control of a 6845 video processor. The card also features the second Z80A mentioned in the advertising for internal control of the video processor and to provide the very flexible range of video card commands. As the video card is not memory mapped, I/O from the bus is via three ports, one for input and output, one for status and one for reset. The keyboard is also connected to the video card. The card carries a 4K EPROM which is the operating system for the Z80A handling all output to the video and all input from the keyboard and a further 2K EPROM is fitted for the 128 character fixed character set. A total of 6K of RAM is provided on this card, 2K for the video display, 2K for the programmable character set, 1/2K for the function keyboard tables, 64 bytes of keyboard buffer, 128 bytes of video input buffer and the remainder as workspace, leaving about 1K free. It is possible to further extend the range of commands to the video card by writing programs for use in this 1K of spare memory, however, because of the extreme difficulty of debugging programs in this area, this is best left to experts. On power up the fixed character set is complimented and then loaded to the programmable character generator providing 128 normal characters and 128 inverse video characters. Individual characters within the programmable set may be reprogrammed at will or complete character sets changed as required. The preset function key definitions are down loaded from the EPROM into the RAM table on power up, and may be changed by keyboard entry or by program from disk. The function key table space is allocated dynamically. Normally, each function key would be assigned the few bytes it needs in the table space, however, it is possible to program one key with almost all the 1/2K available at the expense of programming space for the other keys. The 64 byte keyboard buffer allows a useful type ahead facility.

Twin Z80s?

The Gemini advertising boasts that the Galaxy features two Z80A processors, which whilst strictly true does not present the full picture. The second Z80A certainly speeds video operations and greatly enhances the flexibility of the video card, however, it would be difficult in the extreme and probably be a self defeating exercise to try and make the second processor share in some multi-processor role. In view of this, as each drive contains a dedicated processor, the 6845 and the WD1797 are also both dedicated processors and the keyboard contains a further derivative of the 8045 family, why not advertise that the machine contains no fewer than seven processors?

CP/M Operating System

The Galaxy uses the familiar CP/M version 2.2 as its operating system. Little need be said here about CP/M and its utility programs as these have all been adequately covered elsewhere. However, the Galaxy features a number of small but significant improvements related to the BIOS. Many, criticisms have been levelled at CP/M as an operating system, not the least of which refer to its 'unfriendliness'. Some criticism is justified, as is probably the case for any operating system, but CP/M comes in for a lot of unjustified criticism, which is not the fault of Digital Research, but the fault of the programmers who wrote the BIOSes for various machines. Gemini amply demonstrate what can be done to confound the critics, and deserve the highest praise for both the orderly approach and the manner in which they have removed much of the supposed unfriendliness of CP/M without touching a byte of CP/M itself.

Perhaps the worst areas of CP/M are its very bald and unhelpful disk error trapping. Now as the disk access routines are in the BIOS and as stated, the BIOS is the province of the system programmer and not Digital Research, scope exists to improve things. With the Galaxy the days of 'BDOS ERROR ON DRIVE x', a very unhelpful statement, are gone. In the event of a disk read/write error, the Galaxy BIOS traps it and attempts to reread it four times, if that fails, then the heads are drawn back to track 0, repositioned, and the read attempted another four times. Only after this is an error reported with a message indicating the type of error, the track and sector where the error occurred and a polite request to the user to either try again, accept the error anyway or boot the system. In the odd instances where disk errors have occurred and been reported, the fault can often be cleared by removing and replacing the disk in the drive and then requesting a retry.

Another notable improvement is the 'on screen' editing using the cursor controls. An invaluable feature, particularly for careless two finger typists such as the author. The CP/M input works normally until the EDIT key is depressed, where upon the cursor changes from its normal underline symbol to a solid inverse video block, indicating edit mode. The cursor can then be moved around the screen at will and incorrect input lines edited. This feature was found to be particularly useful with MBASIC, as most editing could be achieved without recourse to the somewhat cumbersome internal Microsoft editor. The screen editing suffers from two limitations, the edit line must not exceed one displayed line, 80 characters, and it can only be used with programs that use the CP/M buffered line input mode. These features are all under the control of the programmer who wrote the original BIOS, and not a function of the Galaxy hardware, so it is pertinent to ask why these features do not seem to have appeared on other machines?

In common with most implementations of CP/M, there are a number of default modes, whether to run a program on start up, what type of output device to select, etc, which, whilst selectable by the user, are often poorly documented (if at all) and usually hard to find. Likewise the hardware also has a number of defaults which need to be declared on start up, such as the speed of the UART, etc. Gemini overcome this problem by supplying a small utility program, CONFIG.COM, which allows all the defaults to be set in the most simple manner. This program is a delight to use and works by using a menu of the options available and then allowing the particular option to be set using the screen editing. The program is self documenting and explains the consequences of any particular default set up.

Utility Programs

Gemini also supply a number of other utility programs (over and above those normally supplied with CP/M). FORMAT.COM is a disk formatting program, allowing the user to format his own disks for use in the Galaxy. Format programs are a vital necessity, and it surprises me that some manufacturers can actually supply machines without a format program and then cash in selling 'own brand' formatted disks at exhorbitant prices. The format program takes about two minutes to run in all and carries out a thorough check of the disk validity. Any disk errors are reported, it being up to the user to reject the disk or try reformatting it. As far as disk tolerance is concerned, the Galaxy seemed to be happy with the most decrepit of disks. Even well worn disks originally supplied as single density were found to format, verify and run correctly without error.

Among the other utilities is a disk backup program. With disks of 400K capacity, backing up a disk using the CP/M PIP program can be a tedious business. The backup program accomplishes the feat in about 110 seconds (less for a disk containing less data), having fully verified the copy.

As mentioned previously, a utility is supplied to save the user defined function keys, further utilities are provided for cassette tape read and write.

Documentation

The documentation for the Galaxy itself comes bound in an A4 folder along with a CP/M manual and a number of other documents such as the CP/M license, circuit diagrams, the software manual for the video card, guarantees, etc. The Guide to the Galaxy (with apologies to Richard Adams) is an explanation of the machine and how to use it, written in a refreshing and friendly style, keeping explanations simple and straight forward. The Galaxy manual does not set out to teach the user how to 'drive' CP/M, but lists the specific features of the machine and departures from the normal CP/M machine, such as the screen editing and the utility programs which are all covered in detail. The blow by blow account of how to use CP/M, is left to the CP/M guide. This approach is more than adequate for the user interested in running typical applications software but falls somewhat short of the documentation required to gain an in depth understanding of the machine. Gemini overcome this shortcoming by supplying all the additional documentation for the individual cards and software at very modest prices through their dealer network. The user intending to use the Galaxy as a development tool need not be worried by the lack of heavy technical documentation as the optional card and software manuals are all easily available and most comprehensive.

Expansion

By using an internal bus structure conforming to the popular 80BUS/Nasbus standard, and by providing three additional card slots within the machine, Gemini have allowed for considerable expansion. Gemini's own range of expansion is quite large and includes 5.4 Mbyte and 10.8 Mbyte hard disks. But the overall range is quite impressive, coming from E V Computing, Microcode, Climax Computers, Nascom and IO Research amongst others. The exceptionally high resolution Pluto colour card sub-system by IO Research is one such, whilst colour graphics at a more modest price (and resolution) is provided by the Climax colour card. Real time clocks are provided by both Gemini (two versions) and from E V, whose clock makes ingenious use of the Z80 on the video card, displaying real time in the top corner of the screen. A collection of various I/O cards are available from Gemini, E V, Nascom and IO Research, whilst battery backed RAM IS available from Microcode. A full implementation IEEE488 card is available from E V and a High Speed Arithmetic Processor is now available from Belectra. The already broad range covers most needs and the main problem appears to be finding out about the various expansion facilities on offer. Also a number of new cards will be added before the end of this year.

Network System

One of the more impressive features of the Gemini is the Network system, which can be installed retrospectively into a Galaxy. Essentially the Gemini MultiNet network system consists of a file server and up to 31 stations. The file server is based on a standard Galaxy 2 but substitutes a 5 or 10 MB Winchester drive in place of floppy in drive A. A small Network controller card is connected to the PIO on the CPU card and thence to an additional three pin connector on the back panel. Each station appears to the user as a separate disk based CP/M computer of very similar specification to the Galaxy 2 using the Winchester disk of the file server for disk access. Although the file server is normally supplied as a complete entity it is possible to up-grade an existing Galaxy into a Network file server by feeding an external Winchester sub-system (GM835) and an internal Network card (GM836UPG) and, of course, the necessary software. Gemini supply workstations for use on the Network which are similar in style to the Galaxy, but physically smaller, as they do not require an on-board disk. As with the file server the Galaxy 2 can be converted for use as a workstation with the advantage of retaining a local floppy disk capability. Gemini Dealers are able to carry out either conversions.

In use the Network is extremely reliable and the software is delightfully "bug" free. The speed is surprising; when only one station is requiring access the workstation actually responds faster than the normal Galaxy 2 due to the 250K baud network data rate and use of RAM spooling buffers on the file server. Naturally, the reponse times get longer with heavy usage. However, a 10 station Network is not markedly slower than a normal disk system. In terms of cost a Network becomes viable at about four systems, being cheaper than four separate Galaxy machines.

Conclusions

Over all the Gemini Galaxy is a complete machine from a number of angles. Complete in the respect that it can be purchased and plugged in and it will go. Complete in the respect that the machine is robust enough and with enough expansion capability to be of serious use as a development tool from both the hardware and software aspects. Complete in that would appeal to the serious business user and the computing enthusiast alike. All in all, a very capable all rounder with only a small number of niggles and at a very realistic price.

PRIVATE ADVERTISEMENTS

There seems to be some confusion over private advertising in 80-BUS News. If you are selling equipment or software that you purchased yourself but no longer want, then you may place a short ad. in 80-BUS News free-of-charge. If, however, you wish to sell copies of a program that you have written or a board that you have designed, then please send in for details of our rates.

NOTE: In the case of the **free** ads. these will be placed in the first issue in which there is available space for the advert. It is therefore possible that the ad. will not appear for a little while. PLEASE DO NOT WRITE IN AND COMPLAIN!!

Doctor Dark's Diary — Episode 18.

Horror story to make your flesh crawl!

My cat recently discovered that he could just fit into the spare space in my system's rack frame (shudder!) and he proceded to do just that. Some time later, when I came to use the machine, I found huge masses of loose fur on the rear of the end board, and thought that his usual static charge would almost certainly have destroyed pretty nearly all the memory on the board, which just happens to be my first MAP80 RAM board, with 256K of highly expensive chips on it. So I cleaned the fur off, as carefully as I knew how, panicking all the time, and cursing steadily in Finnish (the only language suitable for swearing on such a scale - an example would surely destroy the disc it was saved on, so I leave it to your imagination) and ran a test, which consisted of copying a full disc to drive P, using the verify option of PIP.COM to test the memory. This eventually came up with a verify error, and I thought my worst fears were confirmed. But the amount of space left on P was 58K. So it looked as if the MAP board was OK, and that the problem must be on the modified Gemini GM802 64K board also on the system. This board had been modified in accordance with MAP80's instructions, but because the virtual disc had never been full before, the GM802 modification had never been fully tested.

To summarise, I have a Nascom 2, MAP80 RAM with 256K, modified Gemini GM802 with 64K, Gemini GM809 disc controller, and IO Research Pluto graphics board. It seems very likely that there must be some sort of addressing conflict going on between the Nascom 2 and the GM802, so that when a program attempts to read the top 64K of memory, provided by the GM802, it manages instead to read the Simon ROM, and the verification fails. If this is so, and I am only guessing, how do I change things to cure it? Any suggestions from all the hardware whiz kids among you would be most gratefully accepted. A letter about the problem, which I sent to MAP80 a long time ago, has still not been answered, which earns them a black mark.

A.S.A. Advertisement of the Year Award.

I should like to nominate Climax for this award, on account of their misleading statement that there is no need to pay over two hundred pounds for a colour graphics board. Their board costs £199, to which you have to add VAT at the usual 15%. That comes to more than two hundred pounds when I work it out... [Ed. - only if you are NOT a company AND live in the U.K.]

Colour Graphics - Why Pay More Than £100?

If you can only afford that much, and you want your 80-BUS system to have colour graphics, here is one of my more intelligent hardware ideas. You can have 256 x 192 colour graphics, in eight colours, although the display does not fill your television screen, having a border all round it. The graphics unit will be able to run programs independantly of the main computer, which is obviously a good thing. All you have to do is buy a 16K Spectrum, and interface the 80-BUS to its edge connector. All the data and address lines are there, along with the Bus Request and Bus Acknowledge lines, so the Spectrum could be treated as a page of memory by the 80-BUS system. All you have to do is design a simple circuit to go between the two systems, and you will make your fortune!

Free Program! (Couldn't get anyone to buy it!)
This program prints out a brief (very!) explanation of each of the error messages given out by the Hisoft Pascal 4 compiler. It is run from CP/M by typing the word "explain", followed by a space, then the error number, and then "enter". This saves one the bother of getting the manual out and turning to page 35, which is where I have copied all the text from, so the program probably belongs to Hisoft at least as much as it does to me! I did actually try to sell it to Hisoft, then I tried to give it to them, in the hope that they would send it out with all their compilers, but they were not putting up with such amateurish efforts, or something like that...

Dreadful Chess Tournament Ends in Exhaustion.
A friend of mine has been improving Sargon for some time now, and recently sent me a tape of his latest version. As it uses good old Nas-Sys in the approved manner, it works fine with MONITOR.COM, as you would expect. I beat it on level three, and decided to play it against my Spectrum, which I loaded with Quicksilva's program "The Chess Player". (Silly Scenario number 42, an alien android has arrived and will destroy Earth unless you can beat it at Chess! Presumably this is a sop to the space invader fans.) The resulting game was very enlightening. It was also very exhausting, as Marvin is upstairs, while the Spectrum is downstairs. This exercise factor was one reason why the game did not get finished. The main reason, however, was that the Spectrum was so dreadfully slow that in a tournament it would have run out of time. Marvin, running the improved Sargon, produced a move within two minutes, at all times. When in check, the response was much faster, under ten seconds.

The Spectrum played with the white pieces, while Marvin was black.

1	e2 e4	c7 c5	10	e1 d1	g8 f6	19	b2 b3	f6 f5
2	b1 c3	b8 c6	11	f7 h8	e7 e5	20	h1 g1	b7 b6
3	g1 f3	d7 d6	12	d5 e6 chk	d7 e6	21	h2 h3	g4 d4
4	d2 d4	c8 g4	13	c1 g5	f8 e7	22	d1 d4	c5 d4
5	d4 d5	c6 e5	14	e4 e5	d6 e5	23	c3 b5	a7 a5
6	f3 e5 ?	g4 d1	15	d1 e2	a8 h8	24	g1 d1	e7 b4
7	f1 b5 chk	d8 d7	16	g5 f6	g7 f6	25	f2 f4	b4 c5
8	b5 d7 chk	e8 d8	17	a1 d1	h8 g8	26	Dr Dark to bed...	
9	e5 f7 chk	d8 d7	18	g2 g3	g8 g4			

If you play it through, you will probably reach the same conclusions that I did. The Spectrum program is very heavily biased towards moves that put its opponent in check, and almost always takes when offered an exchange. The final position would almost certainly lead to a draw, with just the Kings left. At least Sargon's habit of throwing its Queen away in four moves has been fixed, but neither program seems to have much idea of how to put the other in check mate - there are several points in the game where good moves were ignored, in favour of material gain.

Why have I bought a Spectrum?
Not just to play chess against Marvin, I can assure you. And not to use as a colour display board either, for that matter! No, it is far more devious than that, I can assure you! Some months ago, when I first heard about Micronet 800, I decided that Marvin just had to be connected. I phoned the local British Telecom sales department, who had never heard of Prestel, and eventually managed to get the number of the Micronet people. The man I spoke to was very interested to hear about Marvin, but was also very firm - there

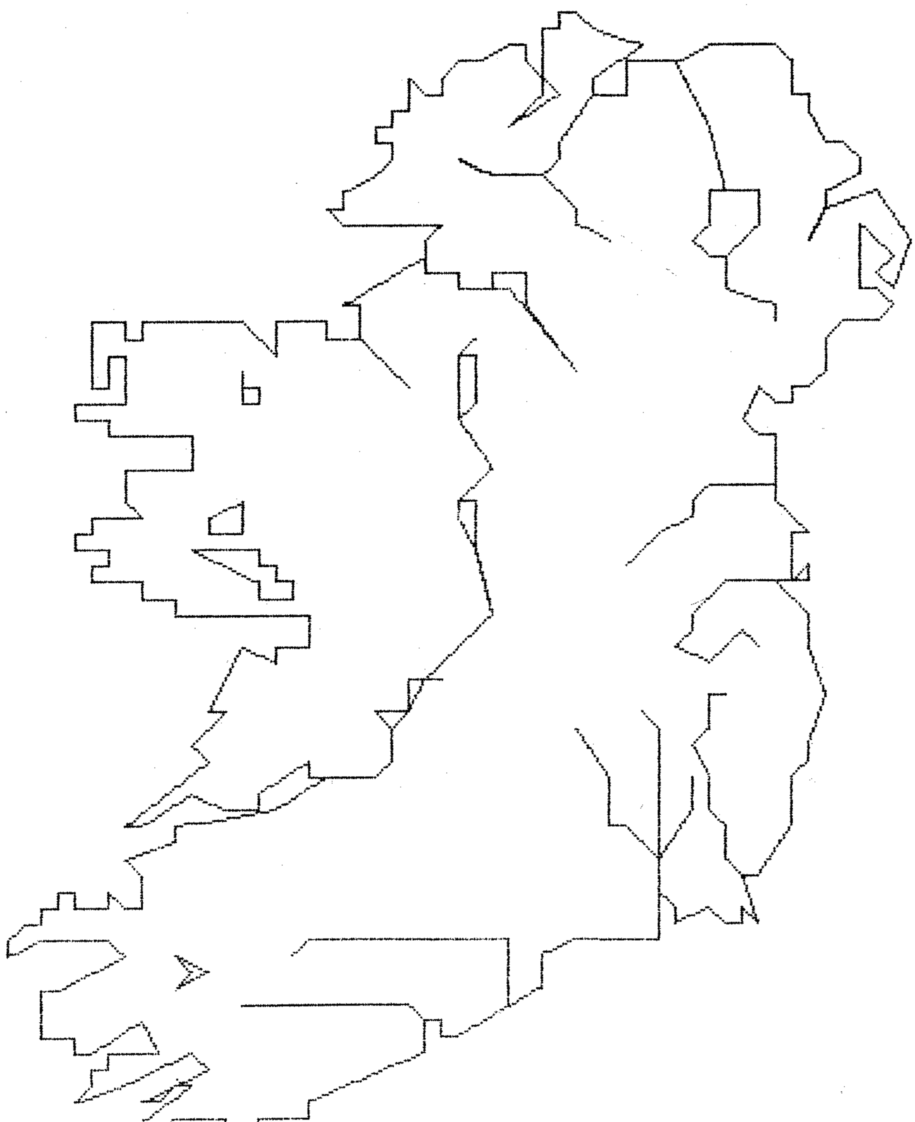
```
PROGRAM explain;
VAR
  errnum : INTEGER;
PROCEDURE getnum;
{This routine extracts the number that followed the "explain" command from the
CP/M default buffer. It is easily confused by silly input, although this will
not do any harm, you just get told that there is no such error number. That'll
teach you!}
VAR
  bytes, tens, units : CHAR;
BEGIN
  tens := CHR(48); units := CHR(48); bytes := PEEK(128,CHAR);
  IF bytes = CHR(2) THEN units := PEEK(130,CHAR);
  IF bytes = CHR(3) THEN
    BEGIN tens := PEEK(130,CHAR); units := PEEK(131,CHAR) END;
  errnum := ((ORD(tens)-48)*10)+ORD(units)-48
END;
{Main program follows. It consists entirely of a call to the routine "getnum"
followed by a monstrous CASE statement that prints out the appropriate error
message.}
BEGIN
  getnum;
  CASE errnum OF
     1 : WRITELN('Number too large.');
     2 : WRITELN('Semi-colon expected.');
     3 : WRITELN('Undeclared identifier.');
     4 : WRITELN('Identifier expected.');
     5 : WRITELN('Use "=", not ":=", in a constant declaration.');
     6 : WRITELN('"=" expected.');
     7 : WRITELN('This identifier cannot begin a statement.');
     8 : WRITELN('":=" expected.');
     9 : WRITELN('")" expected.');
    10 : WRITELN('Wrong type.');
    11 : WRITELN('"." expected.');
    12 : WRITELN('Factor expected.');
    13 : WRITELN('Constant expected.');
    14 : WRITELN('This identifier is not a constant.');
    15 : WRITELN('"THEN" expected.');
    16 : WRITELN('"DO" expected.');
    17 : WRITELN('"TO" or "DOWNTO" expected.');
    18 : WRITELN('"(" expected.');
    19 : WRITELN('Cannot write this type of expression.');
    20 : WRITELN('"OF" expected.');
    21 : WRITELN('"," expected.');
    22 : WRITELN('":" expected.');
    23 : WRITELN('"PROGRAM" expected.');
    24 : BEGIN WRITELN('Variable expected since parameter is a variable');
               WRITELN('parameter.') END;
    25 : WRITELN('"BEGIN" expected.');
    26 : WRITELN('Variable expected in call to READ.');
    27 : WRITELN('Cannot compare expressions of this type.');
    28 : WRITELN('Should be either type INTEGER or type REAL.');
    29 : WRITELN('Cannot read this type of variable.');
    30 : WRITELN('This identifier is not a type. ');
    31 : WRITELN('Exponent expected in real number.');
    32 : WRITELN('Scalar expression (not numeric) expected.');
    33 : WRITELN('Null strings not allowed (use CHR(0)).');
    34 : WRITELN('"[" expected.');
    35 : WRITELN('"]" expected.');
    36 : WRITELN('Array index type must be scalar.');
    37 : WRITELN('".." expected.');
    38 : WRITELN('"]" or "," expected in array declaration.');
    39 : WRITELN('Lowerbound greater than upperbound.');
    40 : WRITELN('Set too large (more than 256 possible elements).');
    41 : WRITELN('Function result must be type identifier.');
    42 : WRITELN('"," or "]" expected in set.');
    43 : WRITELN('".." or "," or "]" expected in set.');
    44 : WRITELN('Type of parameter must be a type identifier.');
    45 : BEGIN WRITELN('Null set cannot be the first factor in a non-');
               WRITELN('assignment statement.') END;
    46 : WRITELN('Scalar (including real) expected.');
    47 : WRITELN('Scalar (not including real) expected.');
    48 : WRITELN('Sets incompatible.');
    49 : WRITELN('"<" and ">" cannot be used to compare sets.');
    50 : BEGIN WRITELN('"FORWARD", "LABEL", "CONST", "VAR", "TYPE", or');
               WRITELN('"BEGIN" expected.') END;
    51 : WRITELN('Hexadecimal digit expected.');
    52 : WRITELN('Cannot POKE sets.');
    53 : WRITELN('Array too large (>64K).');
    54 : WRITELN('"END" or ";" expected in RECORD definition.');
    55 : WRITELN('Field identifier expected.');
    56 : WRITELN('Variable expected after "WITH".');
    57 : WRITELN('Variable in "WITH" must be of RECORD type.');
    58 : BEGIN WRITELN('Field identifier has not had associated "WITH"');
               WRITELN('statement.') END;
  {Purists should not look at the next four messages, or they will be ill.}
    59 : WRITELN('Unsigned integer expected after "LABEL".');
    60 : WRITELN('Unsigned integer expected after "GOTO".');
    61 : WRITELN('This label is at the wrong level.');
    62 : WRITELN('Undeclared label.');
    63 : WRITELN('Cannot assign or POKE files.');
    64 : WRITELN('Can only use equality tests for pointers.');
    65 : BEGIN WRITELN('The parameter of this procedure/function should');
               WRITELN('be of a FILE type.') END;
    66 : WRITELN('File buffer too large (>=256 records i.e. 32K).');
    67 : BEGIN WRITELN('The only write parameter for integers with two');
               WRITELN('":"''s is "e:m:H".') END;
    68 : WRITELN('Strings may not contain end of line characters.');
    69 : BEGIN WRITELN('The parameter of NEW, MARK or RELEASE should be');
               WRITELN('a variable of pointer type.') END;
    70 : WRITELN('The parameter of ADDR should be a variable.');
    71 : BEGIN WRITELN('All files must be FILEs of CHAR or subrange');
               WRITELN('thereof.') END;
    72 : BEGIN WRITELN('Files may only be used as global variables or');
               WRITELN('variable parameters.') END;
    73 : BEGIN WRITELN('"RESET" and "REWRITE" may not be used on INPUT');
               WRITELN('or OUTPUT.') END
  ELSE WRITELN('There is no such error number.')
END.
```

was no way I would ever get permission to hook a home brewed heap like that up to British Telecom's lines. You have to have one of the acceptable machines, like the grossly overpriced BBC machine with its pathetic 32K, or a Spectrum. So I am going to connect my Spectrum up to Micronet. I may also be able to link the Spectrum to Marvin, if the circuit I am trying to design works. No doubt, if it does not, I shall have a dead Spectrum to bury! Further progress will be reported as soon as there is any. I am still waiting 28 days for the modem to arrive...

And finally...

Remember the Marx Brothers film, where Groucho dictates a letter to his dentist, "Please find enclosed a cheque for $500"? When Chico asks if he should put the cheque in, Groucho says "You do and I'll fire you!" In his letter advising me of the deadline for this issue, our noble editor asked me to put this article on "the enclosed disk". Well, I looked ever so carefully, but there was no sign of any disc! Still, he didn't forget to send me a cheque, so all is forgiven...

END.



Climax Routine **by R. O'Farrell**

As David Hunt has written a comparative review of the three colour boards for the 80-BUS, I have willingly left the detailed review of the Climax MV256 to him. Listing 1 is a program to run on it. This program draws a map of Ireland including the major rivers. Included are two extensions to this program. Listing 2, also written in MBASIC, is to be inserted into Listing 1 according to its line numbers. This draws the picture on an EPSON FX80 installed as the LST: device in Graphics Mode 4. This is slow, taking 27 minutes to draw the picture!

What happens in the printer driver is this: The Y coordinate is set to 255, and the X coord to 0. GETCOL(X,Y) is called to return the colour of the pixel at X,Y. If this is not black, then we calculate the value needed to display this bit if we print rows of 8 bits vertically. We work down 8 rows, in the first column, building up the value of the byte. Then we move to the next column, top row, and carry on. Having finished the screen-width, we transmit the graphics line selectors, and the entire row. Then on to the next group of 8 raster lines. Because of the looping involved, this is very slow! Listing 3 is an improved version of this. It patches into the MVLINK primitives and is written in assembly language. It should be inserted into MV2.MAC after INIT, and MVLINK.MAC reassembled and linked with M80/L80. Using machine code, this performs the same task in 35 seconds! Note that there are two other entries needed to use this listing - an insertion of a call to DUMP in the jump table (p53, Appendix B, Microvector Manual), at address 02FA and the insertion of a new label in GETP(X,Y), page 59, label name GETCOL at address 049A (PUSH BC) in the MVLINK listing. This latter saves a call to GETPAR and READY, to speed things up slightly. It is important that the LST: device driver should not interfere with the data at all, otherwise spurious characters may be printed. This means that users of the SYS BIOS and the MAP80 BIOS will have to ensure that the page length is set to 0 by changing the byte in the BIOS at plpag:, see the system notes for details. If using the new primitive, I suggest that it live at MVBASE+99, and be called DUMP. It should be called with one parameter, the colour of the background of the screen, which will not be printed. All other colours will be. It is possible, using this primitive, to print an enlarged picture by drawing on screen four successive quadrants, printing each, and winding the printer back up after them using the reverse line feed. This is left as an exercise for you!

It is important when using the MV256 that the program be saved to disc before executing. If through some error or oversight one of the CALLs is not properly defined, it will be treated as CALL 0, and the program crashes, causing much weeping and gnashing of teeth. While there are methods of regaining the source code, it is easier to MVLINK, and LOAD "FILENAME"!

When purchasing the MV256, it is worth getting the disc with the driving primitives and demonstration program. The demo programs are quite impressive, and give one something to aim at.

LISTING 1

```
09 DEFINT X,Y,C,I,N
* INSERT HERE THE CONTENTS OF FILE PRIMDEF ON CLIMAX DEMO DISK*
400 PRINT CHR$(26);
410 INPUT "Enter Scale Required";SCALE
420 IF (SCALE<=0)OR (SCALE>10) GOTO 400
440 C=0:CALL PENCOL(C):REM select black background
450 CALL SCAN:REM set screen
460 C=6:CALL PENCOL(C):REM select Yellow for outline
470 GOSUB 540:REM call lineprinter
480 C=9:REM SELECT BLUE
490 CALL PENCOL(C)
500 FOR J=0 TO 14:REM 15 RIVERS
510 GOSUB 540:REM call lineprinter
520 NEXT J
521 REM *** LISTING 2 will be inserted here ***
539 END
540 REM Procedure to print line.
550 REM First data entry is name, which we read and discard
560 REM Second is number of points making up line
570 READ A$,N:N=N-1
580 READ X(1),Y(1)
590 X(1)=INT(X(1)*SCALE):Y(1)=INT(Y(1)*SCALE)
600 FOR I=1 TO N
610 X(0)=X(1):Y(0)=Y(1)
620 READ X(1),Y(1)
630 X(1)=INT(X(1)*SCALE):Y(1)=INT(Y(1)*SCALE)
640 CALL LINEI(X(0),Y(0),X(1),Y(1))
650 NEXT I
660 RETURN
670 DATA "IRELAND",218
680 DATA 8,0,12,3,11,3,10,2,9,2,13,4,12,5,10,4,8,3,5,2,6,3
690 DATA 6,4,7,4,7,5,10,5,9,7,6,5,5,5,5,6,3,6,3,9,4,9,8,11,7,12,3,12,1,11
700 DATA 1,12,2,13,3,13,3,14,4,14,4,15,5,15,5,14,7,14,7,15
710 DATA 8,14,9,14,9,16,8,17,11,18,11,19,17,20,20,22,19,22,19
720 DATA 23,16,21,16,20,14,20,12,21,9,19,8,19
730 DATA 13,23,12,24,14,26,13,26,15,30,17,29,17,30,19,30,19
740 DATA 32,11,32,11,33,9,33,9,34,6,34,6,35,7,35,7,36,5,36,5
750 DATA 37,6,37,6,38,9,38,8,39,8,41,12,41,12,43,7,43,7,44,5
760 DATA 44,5,45,8,45,8,48,7,48,7,46,6,46,6,50
770 DATA 8,50,8,49,9,49,9,50,15,50,17,48,17,50,20,50,20,49,22
780 DATA 49,22,51,21,51,26,54,26,55,27,56,21,56,20,57,21,57,21
790 DATA 58,23,59,24,60,24,61,23,61,23,62,24,62,24,63,25,63,25
800 DATA 65,26,64,27,64,27,65,28,66,29,66,31,67,32,67,32,66
810 DATA 33,65,33,64,32,63,31,62,34,64,33,65,33,68,34,68,34,69
820 DATA 35,69,36,68,39,67,36,65,36,64,38,64,38,66,41,66,43,67
830 DATA 47,67,48,66,48,64,49,64,49,63,50,61,51,61,52,60,52,59
840 DATA 50,58,50,57,49,55,50,57,53,58,55,55,54,52,53,53,54,54
850 DATA 52,56,52,52,53,52,54,51,53,50,51,50,50,49,50,47,49,46
860 DATA 48,46,48,45,47,45,46,46,45,44,46,43,47,43,47,39,49,37
870 DATA 48,37,48,34,49,35,49,34,47,34,49,32,49,30,50,27,49,24
880 DATA 49,23,48,22,48,19,46,16,45,16,46,13,45,14,45,13,44,13,43,14,41,13
890 DATA 41,14,40,15,40,12,35,12,33,11,33,9,28,6,27,6,27,7,26
900 DATA 7,26,5,19,2,19,1,16,1,16,0,14,0,14,1,11,1,10,0,8,0:REM 12,3:REM ,11,3
910 REM RIVER DATA
920 DATA "SHANNON",30
930 DATA 20,22,23,22,24,23,24,25,25,26,26,28,27,28,26,28,30,32,29
940 DATA 36,29,39,28,39,30,41,28,44,28,48,29,49
950 DATA 28,48,29,48,29,45,28,44,30,41,28,39,28,38,29,36,30,32,26
960 DATA 28,25,28,25,26,23,26,24,25:REM SHANNON WITH LAKES
970 DATA "CORRIB",10
980 DATA 16,33,16,34,12,36,16,36,16,35,17,35
990 DATA 17,34,18,34,18,33,16,33:REM CORRIB
1000 DATA "UPR CORRIB",5
1010 DATA 13,38,15,39,15,37,13,37,13,38:REM UPR CORRIB
1020 DATA "MASK",6
1030 DATA 15,45,15,47,15,46,16,46,16,45,15,45:REM MASK
1040 DATA "MOY",2
1050 DATA 22,49,25,46:REM MOY
1060 DATA "ERNE",11
1070 DATA 26,54,26,53,28,53,28,52,30,52,30,53,32,53
1080 DATA 32,51,35,47,31,52,30,52:REM ERNE
1090 DATA "FOYLE",12
1100 DATA 36,64,34,61,34,60,33,59,30,59,28,60,28,60
1110 DATA 30,59,33,59,35,57,35,56,37,55:REM FOYLE
1120 DATA "BANN",17
1130 DATA 41,66,43,62,44,58,43,58,43,56,42,55,43,54,44
1140 DATA 54,46,56,46,58,44,58,46,58,46,56,44,54,44,52,47,51,47,50:REM BANN
1150 DATA "BOYNE",6
1160 DATA 47,40,43,40,42,39,42,38,40,37,38,35:REM BOYNE
1170 DATA "LIFFEY",8
1180 DATA 47,34,44,34,42,32,42,31,41,30,43,29,45,31,46,30:REM LIFFEY
1190 DATA "SLANEY",9
1200 DATA 45,16,44,17,44,19,43,20,43,22,42,24,43,25,43,27,44,27:REM SLANEY
1210 DATA "SUIR",14
1220 DATA 40,14,40,17,42,20,42,22,42,20,40,17,40,25,39,26,40,25
1230 DATA 40,17,38,19,37,19,37,22,35,25:REM NORE/SUIR
1240 DATA"BLACKWATER",4
1250 DATA 31,8,31,12,19,12,18,11:REM BLACKWATER
1260 DATA "LEE",3
1270 DATA 26,7,25,8,15,8:REM LEE
1280 DATA "KILLARNEY",5
1290 DATA 11,9,12,10,11,11,13,10,11,9:REM KILLARNEY
1300 END
```

LISTING 2

```
521 REM PRINTER DRIVER
522 LPRINT CHR$(27);"1"
523 BLACK=0
524 FOR YY=255 TO 0 STEP -8
525 LPRINT CHR$(27);"*";CHR$(4);CHR$(0);CHR$(1);
526 Z=YY MOD 8
527 FOR XX=0 TO 255
528 BYTE=0:REM INITIALISE
529 FOR J=Z TO 0 STEP -1:REM BUILDUP CHAR
530 N=YY+J-7
531 CALL GETP(XX,N)
532 COL=INP(&HDO)-240
533 IF COL<>BLACK THEN BYTE=2^J+BYTE:REM BUILD CHAR
534 NEXT J
535 LPRINT CHR$(BYTE);
536 NEXT XX
537 LPRINT:REM CRLF at end of line
538 NEXT YY
```

LISTING 3

MV256 Screen Dump to Epson FX80

```
        .Z80
dump:   ld a,cr         ;Make sure we start new line
        call output
        R
        ;output linefeed selector
        ld a,esc
        call output
        R
        ld a,three      ;select linefeed spacing 24/216"
        call output
        R
        ld a,twnty4     ;spacing parameter
        call output
        R
        call getpar     ;get background colour
        R
        ld ix,dumpmod
        ld (ix+1),a     ;store colour at appropriate location
        ld de,00ffh     ;Set E=255, D=0
nxtrow: ld a,esc
        call output     ;output graphline selectors
        R
        ld a,"*"
        call output
        R
        ld a,gr4        ;select graphics mode 4
        call output
        R
        ld a,0          ;no. chars mod 255
        call output
        R
        ld a,1          ;no. chars div 255
        call output
        R
        push de         ;save row
row:    ld b,8
        ld c,0          ;initialise byte for output
        ld hl,0000h     ;set L=0, column pointer, H=0
makbyt: call getcol
        R
dumpmod:cp 0            ;test colour at pixel
        jr nz,skip      ;does it differ from background?
        defb 0cbh,31h   ;*** UNDOCUMENTED Z80 INSTRUCTION ***
                        ;shifts register C left one bit, with
                        ;a one shifting in from right
        jr skip1
skip:   sla c           ;set bit equal to 0
skip1:  dec e           ;adjust row pointer
        djnz makbyt
        pop de          ;restore top row
        inc l
        jr nz,row
        ld a,cr
        call output
        R
        ld a,lf
        call output     ;output CRLF
        R
        ;E = E-8
        push hl
        ex de,hl        ;rows now in hl
        ld de,8
        or a
        sbc hl,de
        ex de,hl
        pop hl
        jr nc,nxtrow
        ;reset line feeds
        ld a,esc
        call output
        R
        ld a,two
        call output     ;reset line spacing
        R
        ret

lf      equ 0ah         ;linefeed character
cr      equ 0dh         ;carriage return
ffeed   equ 0ch         ;form feed character
esc     equ 1bh
gr4     equ 04h         ;selector for graphics mode 4
one     equ "1"
twnty4  equ 24          ;linefeed spacing

output: push af         ;save registers
        push bc
        push de
        push hl
        push ix
        push iy
        ld e,a          ;transfer byte to be transmitted to correct reg
        ld c,5          ;select LST: device
        call bdos
        pop iy
        pop ix
        pop hl
        pop de
        pop bc
        pop af
        return
END
```

LISTING 4

```
521 REM REPLACE PREVIOUS CODE AT 521 et seq with
522 C=0:REM SELECT BLACK BACKGROUND COLOUR
523 CALL DUMP(C)
524 END

*** ADD TO PRIMDEF FOLLOWING LINE ***
399 DUMP=MVBASE+99:REM THIS DEFINES DUMP'S VALUE FOR CALL
```

The Dave Hunt Sundry Bits

Due to Paul's holiday coming immediately after mine, all progress on the 80-BUS rags has ceased during most of August and the first bit of September, and as a consequence only two letters have come my way. The first from S. Sedwell in Romford Essex is a plea for suggestions as to where to get his Nascom 1 repaired. He asks if there are any clubs in his area, or, is there anyone with the necessary expertise locally who could help. If anyone out there is prepared to sort out a recalcitrant Nascom 1, please drop us a line and we'll pass it on.

The second letter, from G. Orford of Henleaze, Bristol has a couple of greivances to air, so I have transcribed his letter in full.

"The Editor, 80BUS News.

Having waited a few weeks to cool off, I still find myself annoyed at Dr. Dark's Episode 15. Many of us who don't jump at his every suggestion are not "idle bodies". Indeed somewhile back I submitted an article for consideration for publication. Not even a rejection was received in reply although you claim to have gone "professional". Disheartened, I am concentrating on new projects and unless a grovelling apology is received, I shall not divulge my CW/RTTY programs (no chopping of boards needed) or advise on PIO ports used by an expansion board advertised in 80-BUS News.

In short, I shall be as helpful as D. R. Hunt on his cure for computer generated RFI.

Yours faithfully, G. Orford."

Well, first, as to Dr. Dark, I have no intention of making an apology on his behalf, I suggest you drop him a line or go and see him, after all he's a lot nearer you than me, just down the road on the M5 at Taunton. Personally, I thought his comments in episode 15 were a bit strong, but experience has proved that the readership in general is pretty lethargic and it requires someone to say something pretty rude occasionally, otherwise nothing seems to happen. Mind you, despite my (printed) request to be included his 'Ring of Iron', nothing seems to have come my way.

Now perhaps I'll attempt the grovelling apology, or at least an explanation. The lack of rejection slips, and the non-return of the orginal copy (something you didn't mention). I don't recollect seeing your article, although I may have, however, nothing is ultimately rejected unless it is so badly written that I or others refuse to 'ghost re-write' it on the grounds that we can't understand it. All other unprinted articles are carefully kept, perhaps for rewiting or, maybe, until another relevant article comes along to retain the balance. Or for a variety of other reasons.

As far as RTTY and CW programs go, these could be very useful if I ever get my way for a Radio Amateur issue. My CP/M RS232 RTTY software all works well, but, dare I confess it, the RTTY to RS232 decoder isn't entirely successful, due to my wrong choice of phase lock device. As to my being unhelpful with my cure for RFI, I thought I covered it by saying I bunged the whole lot in an aluminium box and made sure the computer was all thoroughly shielded.

All About RFI

Still, perhaps a more thorough investigation into RFI is called for. For the uninitiate, RFI stands for Radio Frequency Interference, and if you are still wondering what that is I suggest you stand your average portable radio next to the computer whilst it's running. Now lets have a quick look at where this muck is coming from. Its origin is the system clock, or clocks, but it isn't the clock which is necessarily causing the trouble. The problem lies in the fact that virtually all the computer uses square waves throughout its logic. In fact most of its logic would refuse to work if fed with anything other than square waves.

It is a law of nature that square waves contain odd harmonics of the fundamental frequency, the better the shape of the square wave the more harmonics there are. In practice the harmonics from the high frequency clock is not likely to contain much in the way of harmonic content due to the relatively poor wave shape at those frequencies, but it is successively divided and shoved through gates to perform the various functions of the computer. All this causes the fundamental frequencies to be reduced and consequent improvement in the wave shapes. The net effect is that there are square waves busily clocking things around the computer at all sorts of fundamental frequencies and at the same time generating many harmonics of the various frequencies involved. All this causes broad band noise with a spectrum from a few thousand Hz to several hundred MHz. All that is required now is for something to radiate this switchng noise and we have considerable RFI.

Of course the pc tracks and wires connected to the computer all act as aerials, and worse, because of the broad band nature of the noise, it is inevitable that some of the tracks will be tuned lengths at the higher frequencies, improving the radiating efficiency of the tracks and leads at those frequencies. Any radio placed in the vicinity of the computer will pick up the noise and if the computer is doing something repetitive, then the sound heard will be the familiar chirps and whistles associated with a busy computer. Even if the computer is doing nothing, it is still performing its own housekeeping, keeping the video and dynamic RAM refreshed, scanning the keyboard for something to happen, etc. The keyboard scanning rate of a Nascom 2 is about 1.5KHz, and guess the frequency of the predominant tone heard when listening to a Nascom waiting for a keypress. This is probably the strongest signal heard, as the radiating element is the keyboard cable which is of substantial length.

And so on to the cure. Well I don't claim it to be a complete cure, as it has not totally eliminated RFI, what it has done is to reduce RFI to sensible proportions so that it is no longer a factor contributing to the difficulty of working dx on 2 metres. It has also removed the faint background hiss which I had attributed to poor aerials on stereo VHF reception and an AM portable radio can now be used in the same room as the computer with no interference on the stronger signals. As I said, I bunged it all in a box. Well, that's not entirely true, more precisely, I had a box made. I thought about 19" racks, but by the time you have acquired a pretty Vero case to go round it, it works out prohibitively expensive. No my box measures about 15" by 9" by 8" and is made out of 16g aluminium sheet. It was made with a base plate with a half inch flange all round, the longer two side plates were made flanged on the ends, and two blank end plates. The metal chopping and folding was done to my rough sketches by H. L. Smith of Edgware Road, W2 who specialise in metal bashing.

Having got the plates, I got out the steel rule, pop punch and electric drill and started work. I had decided where the plugs for keyboard, serial I/O, mains, video and other connectors were to go on the end plates, so these were painstaking chopped out, the smaller ones using a selection of Q-Max punches, the larger ones drilled and then opened up with a metal fret saw and filed to shape. One end plate would carry a 5" Muffin fan to keep the whole lot cool whilst the other end plate had all the holes for the sockets and the mountings for the power supply. The fan presented a problem. I could not find any metal mesh to cover the fan opening, and leaving it open would leak RF like a sieve, so I marked out about 500 3 mm holes over the intake area of the fan and spent an entertaining evening drilling them all out. As the box in its finished form would be almost airtight, I had to drill a similar number of ventilation holes in the other end plate to exhaust the air sucked in by the fan. The intake and exhaust holes for the fan had to be fairly small, as I didn't want to take any chance of RF leaks. Large holes let RF out!! Making all these holes neat and tidy was no easy task.

With the two finished end plates, I started assembling the box. The two side plates were fitted to the base plate in turn, held in place with engineers clamps and a row of holes spaced at 1" was first piloted and then drilled out to 3.3 mm through the side plates into the base plate flanges. The two side plates were then pop rivetted to the base plate. Similarly, the two end plates were clamped to the now fitted side plates and drilled and rivetted in a like manner. This made a nice substantial box without a lid, it was even reasonably square. The box went back to Smiths and a tight fitting flanged lid was then made to fit. The whole lot cost about a tenner, but I dare say that was a special price to me as they owed me a couple of favours. Anyway the box was a lot cheaper than buying a ready made frame and box.

The seven slot mother board was made from 8" wide Veroboard and the positions of the mounting holes marked and drilled on the base plate. The lid was drilled for a number of PK self tapping screws. The time came for a dummy assembly and to see if I had forgotten any holes. The whole lot seemed to fit together, the fan, the PSU, the mother board, the sockets and all, so the whole thing was taken to pieces again. The outside of the box was then painted in dark blue Hammerite paint making sure to mask things like earth points, etc.

The fateful day arrived, the whole lot was reassembled in the box, wired up and switched on for the first time. One thing was immediately apparrent, the fan make an awful racket and the amount of air rushing through the box was overkill in the extreme, more like a wind tunnel than a computer box. On the basis that large diameter slow moving fans are quiet, some way of slowing the fan down had to be found. Simple, a 0.33uF 600V paper capacitor in series with the fan dropped the supply to about 90V, the fan ran a lot slower and was very much quieter. Was the cooling now adequate? Having left the thing on for several hours with all the cards in revealed a temerature rise of all of 10 degrees C so I guessed it was all Ok.

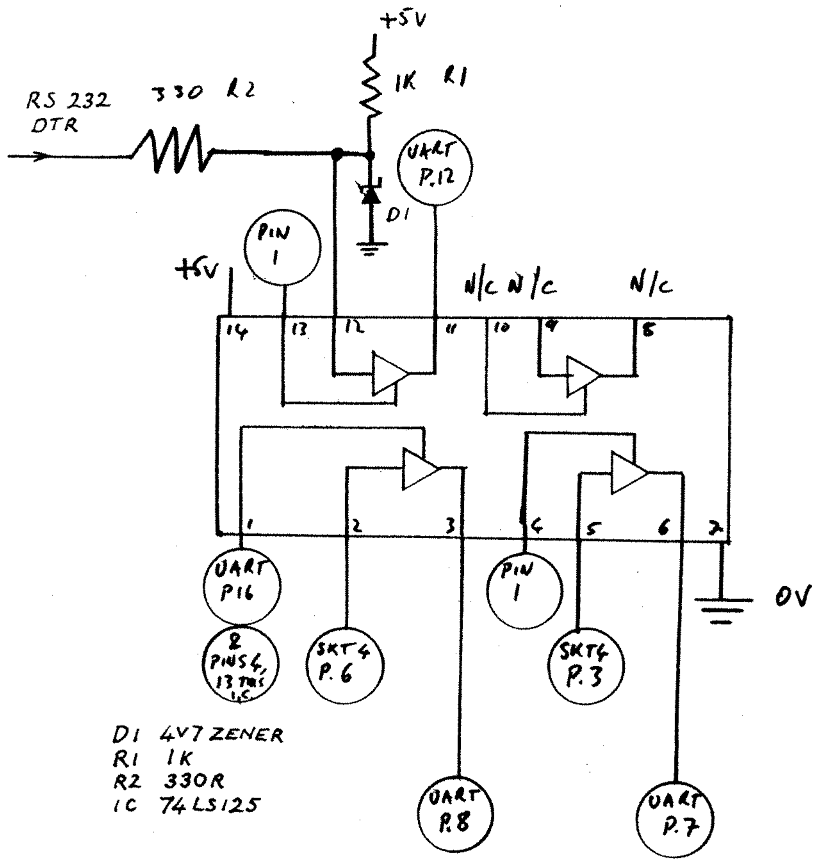
Did it cure the RFI? Well no, not entirely. There was still a lot of keyboard polling noise which inexplicably did not go away when I detached the keyboard. It went away when I detached the printer!! So the printer cable was replaced with a screened cable, leaving the keyboard ribbon cable as before. Still not quite right, the interference shot up when the disks started up so the disk drive cable was rolled into a cylinder and stuffed down the inner of the lapped screen extracted from a piece of RG8 aerial cable. The screened drive cable so manufactured was bonded to the computer box at one end and the drive box at the other. This cured most of the drive noise.

That's where I left it. RFI is now of acceptable proportions, although I dare say I could improve matters if I so wished. Screening all cables to every peripheral might be effective, but to date it will do. I have noticed that when I swing my beam aerial in the direction of the computer more RFI is received, but it will pass until, that is, the nice new 10 element beam I've just bought goes up on the aerial pole, then I guess the hunt for RF leaks will start again.

That just about wraps it up for this session, except for one radio and computer related topic. Recently I have taken up cudgels with the 'C Programming Language'. At the moment I have a fair idea who's winning, and it's not me!! So what I need is a project, a totally selfish project to ensure that I actually do something about it, so I thought I might write a generalised CP/M AMTOR program in 'C' (AMTOR is a way of making computers talk over radio). The snag is, I don't know the protocols, extracts yes, but the complete bit, no. So if anyone has a copy they would part with, or knows where the original references can be found, I'd be grateful if they'd let me know.

So finally, just to get my own back on Dr. Dark for all that Pascal rot he writes ......

```
main()
{
        printf("Bye all\n");
}
```



D1 4V7 ZENER
R1 1K
R2 330R
IC 74LS125

Nascom I/O Board Mod.

Nascom users will know the need to provide the handshake for the RS232 printer on bit 7 of Port 0. One of the shortcomings of the Nascom I/O board is the lack of the RS232 status lines, in particular one input for use as a printer handshake. For three years now I have been using the UART of the I/O board as my printer driver, with handshake on a spare bit of Port 0, the Nascom keyboard port. This has allowed me to drive my printer as fast as possible - currently 19200 baud, using a customised driver. Changing to a GM813 CPU card either means changing the printer to use the UART on the CPU card, or wasting one line of a PIO port on the handshake. Worrying about this, as I don't like untidynesses of that sort, I came up with another answer.

With a little deft work, it is possible to provide the handshake on the Nascom I/O board using the UART status port (port 11H on my system). To do this, it is necessary to tristate the handshake signal onto the UART. This is done using a 74LS125 at a cost of some 25p or so, mounted in a wirewrap socket. As an added extra, we can also obtain two more general purpose input lines.

The UART uses bits 1,2,3,6 & 7 to report status of various conditions on Port 12H. This leaves bits 0,4 & 5 free for us to use. If you remove the board from the rack, and unplug the big chips, plugging them into antistatic foam or a foil covered ceiling tile for safety, you will observe that in the corner beside Skt 4, there is a space of about one socket size. It is possible, using a Veroboard template and a 1mm (or thereabouts) drill, to drill the holes for a 14 pin DIL socket. Into these holes you can insert a wirewrap socket. If you examine the situation carefully, you will find that by careful manipulation, you can bend pin 14 ond pin 7 of this socket to enter tracks of the appropriate voltages. It may help to move some nearby resistor's positive ends. Then connect pins 1 and 13 to UART p16. This picks up the enable signal for the tristate buffers, so that they switch through any data to the bus.

The general connections are shown in the drawing. It is advisable to wire in the level shifter as well for the RS232 levels, as EPSONs and most other printers provide their handshakes at that level. The other two connections, shown as P3 and P6 of SKT 4 are available for your own nefarious purposes, at TTL levels, unless shifted. It is necessary now to replace your handshake routine which tests bit 7 of port 0, with one which tests bit 0 of port 12H (or whatever your I/O board's status address is). If using CP/M, it will be necessàry to change two areas - the LST: device driver, and the Status check for Despool.

Happy Talk **by R. O'Farrell**

One of the awkward things about CP/M is that it is exceedingly difficult to get programs and data from one machine to another, unless they can read each others discs, like the Gemini M-F-B system. The patches to the BIOS to allow multiple format disc read/write are not easy. One method is a modem transfer program (BSTAM or MODEM 7). If one doesn't have one of these, then usually one resorts to PIP.

One can connect up the serial port on two dissimilar machines, ensuring that the same baud rates, parity and so forth are selected, and using STAT to change PUN: to be PTP: (paper tape punch) on one and RDR: to be PTR: (paper tape reader) on the other. One connects the two serial ports together with an appropriate cable, checking that the transmit and receive lines on the different machines are connected to their opposites on the other machine. So on the reader machine one enters:

```
PIP filename.ext=RDR:
```

and on the transmit machine:

```
PIP PUN:=filename.ext
```

For ASCII files, such as assembler listings, this works O.K. up to a length of about 26k, depending on CP/M size. Intoxicated by power, we then try a COM file. It may appear to work. But STAT or XDIR or SD the received file, and compare the filesize with the untransmitted version. Diasaster has struck! Most probably, the received file is dramatically shorter than the original. What has happened is that PIP, on receiving a CTRLZ (1AH) from the serial port has interpreted that as an end of file character. In the case of an ASCII file it would be, as an ASCII file contains only characters 20H to 7FH, with CRs and LFs. In the case of a COM file, the 1AH is an instruction, meaning LD A,(DE) - most useful! The specification of PIP suggests that selecting option [O] for PIP should cause it to ignore ignore the normal CP/M end of file mark, and look to the Console (keyboard) on the receiving machine for the EOF. If this is properly implemented in BDOS, and it usually isn't that I know of, then when all disc accesses on the transmitter have ceased and the cursor is restored, typing CTRLZ on the keyboard of the receiver will cause it to break into a flurry of activity and save the file to disc. Very rarely have I found this to work - why, I don't know.

Another method of file transfer is as follows: Using a disassembler, take the file you wish to transfer (FILE.COM). Disassemble the first 4k of it (say). When you reach the appropriate part of the Disassembler input, tell it that the entire 4k block is DEFB instructions. i.e., treat it as a data area. This avoids any need to fiddle around trying to sort out labels. Take the FILE.ASM produced by the disassembler, and assemble it, linking it if necessary, to produce FILE.HEX. Then PIP this from machine to machine. There is no need to select any options, as being a HEX file, it is ASCII. On the receiver, just LOAD it or ZSID it (can use DDT instead) to convert back into a COM file. If the COM file you start off with is longer than 4k, then do it as a number of 4k blocks (possibly 6k, depends on your machine and CP/M). Remember that a HEX file produced from a COM file multiplies its length by approx 2.65. So the 4k block becomes about 11k. It is better to be shorter than longer, as with disimmilar machines the buffer lengths can differ dramatically. It is disheartening to hear one machine start up its disc to save the file on buffer full while the other machine is still transmitting.

If doing a large program, then it is worthwhile using SUBMIT files with parameters to lessen the amount of work. These can remarkably shorten the repetitive process of assembly and linking. Unfortunately DISZILOG will not work off a SUBMIT file (it doesn't use much buffered console input). If it did, it would speed up the process even more. Has anyone tried a mod on it to do just that? If so I'd be very greatful for details. Minor problems arise. One needs to keep track of the sections of the file - so add an index number, as FILE1, FILE2 etc. Some problem may arise with ORG statements needing to be deleted from the ASM files or with the linker (if used) inserting unwanted jumps to where it thinks entry points ought to be.

When all is transmitted, and the receiving machine has a disk full of FILE1.HEX, ... FILEn.HEX, one can do two things. Either PIP FILE.HEX=FILE1.HEX,FILE2.HEX,...,FILEn.HEX to concatenate the files into one which you then LOAD. Maximum filesize again about 26k - machine and CP/M dependant! or else

```
ZSID
IFILE1.HEX             {Insert file control block of that name}
R                      {Read file}
IFILE2.HEX             {I've omitted ZSID's response of filesize}
R
IFILEn.HEX
R
START END FREE         {Note END address to calculate number of blocks to save}
GO
SAVE xx FILE.COM
```

This entire process is laborious, but it does work. To get the hang of this method, start off with a short ASCII program and get that to transmit. This proves that your initialisation, baud rates, connections etc. are all O.K. Then try a trivial COM file - I wrote one under ZSID. 100H bytes of A's, 100H bytes of B's, 100H of 1A's, 100H of D's, SAVEd 4 FILE.COM, and tried disassembling that in 100H blocks. You will become very familiar with your Disassembler, your assembler, your linker, and above all, with PIP. Do remember that a ^C may reset PUN: and RDR: to other values. Do check them after ^C's until you are sure what happens. Remember PIP option [E] to echo all transfer operations to the screen. A little bit of experiment, copious notes of parameters and options, and you will soon be in meaningful communication with the rest of the world.

More about databases **by D.R. Hunt**

THE SIMPLE BIT

Having read the title and having been put off by 'databases', a little diversion for starters. Time and again newcomers to BASIC are stumped by the need to look up things in databases. The first problem is that they don't know what a database is, and secondly, when they find out, they get frightened of it. If this sounds like you, read on, otherwise, skip to the harder bit.

I don't know why, but when beginning to learn programming in BASIC, most people prefer to leave arrays very strictly alone. I know I was like that when first confronted with an array when I actually laid hands on a Nascom BASIC (was it really all those years ago?). I even went to considerable lengths to program around them. Yet in truth, an array variable is just like any other variable except a bit more versatile. The main problem lies in visualising arrays. I explain to people thus:

'A single dimensional array is easy to picture. Imagine a single row of pidgeon holes numbered 0 to n, where n is as big a number as the number of pidgeon holes required. Anyway, there are these pidgeon holes, and this row is called P (for pidgeon), so P0 is the first pidgeon hole, P1 is the second pidgeon hole, up to Pn where n is the highest numbered pidgeon hole. Got that? Ok. Now lets indulge in a bit of lateral thinking ('cos it's row, remember), well, lets call the whole row P, and refer to the numbers of the pidgeon holes by the same numbers as before, but put the numbers in brackets to indicate that the numbers refer to the individual pidgeon holes while the 'P' refers to the row. So each pidgeon hole is now called P(0) through to P(n). Ok. Now that's the easy bit. Now let's add another row of pidgeon holes beneath the existing row P. We can't call this P, 'cos the other one is called P, so by stretching the imagination we'll call this one Q. Now notice, Q(0) is directly beneath P(0), and Q(n) is directly beneath P(n). We now have two 'one dimensional' arrays. We can add as many one dimensional arrays as we need for any job. The important thing is the pidgeon hole (n) in one array is always directly related to pidgeon hole (n) in any other one dimensional array. Now there are such things as two dimensional arrays, best imagined as areas of squared wall paper, or three dimensional arrays, looking like lots of cubes arranged into box shapes. After three dimensions, my imagination gives up, but having grasped three dimensional arrays, n dimensional arrays aren't too difficult to use, even if you can't imagine them.'

Well, back to the simple database lookup bit, lets go for a telephone directory. All very simple, we use two one-dimensional arrays, one containing the names to be looked up and one containing the appropriate telephone number. The only pre-requisite for this sort of program is the necessity of being able to save and reload string arrays. Now watch out for this one, Nascom 8K BASIC has severe problems in this area as it can't save and reload string arrays. A number of ingenious solutions have been published in early editions of INMC80 so it's back into the archives if you want to use this program with Nascom BASIC.

The program goes like this:

```
 10 DIM N$(300),P$(300): ' Set up the maximum size of the arrays.
 20 ' Reload the names array counting the number of entries in N.
 30 ' Reload the phone numbers array.
 40 '
 50 ' Get the input name
 60 INPUT "What is the name please (enter E to end) ";I$
 70 IF I$="E" THEN 270
 80 '
 90 ' Start the search of N$(A) for a match with I$
100 FOR A=0 TO N
110 IF N$(A)=I$ THEN 230
120 NEXT
130 '
140 ' No find so see if it's to be added, if so add it
150 INPUT "The name wasn't found, do you want to add it (Y/N) ";J$
160 IF J$="N" THEN PRINT: GOTO 60
170 N=N+1
180 INPUT "What is the new name please ";N$(N)
190 INPUT "What is the new number please ";P$(N)
200 PRINT: GOTO 60
210 '
220 ' Name found so print the result
230 PRINT I$;"'s phone number is ";P$(A)
240 PRINT: GOTO 60
250 '
260 ' End, so save the arrays
270 ' Down load from N$(0) to N$(N) to save the names
280 ' Down load from N$(0) to N$(N) to save the phone numbers
290 END
```

All very easy stuff, using two parallel one dimensional arrays, one, N$(n) as the key, the second P$(n) as the data. Notice that I chose to use a string array for P$(n) even though it only contains numbers. This is because it needn't only contain numbers, for instance I always write phone numbers, 01-402-6822, which certainly isn't numeric (that isn't my number by the way, so don't try it.) Or P$(n) could contain an address as well as the phone number, or recipes (short ones as string length is restricted to 255 characters), or, well, you name it.

THE HARDER BIT

Having looked at both types of commonly used database, the random access and the sequential (and the sequential free field type) and having looked at the way in which the data would be split up into fields within the records in the file (neat summary that, databases consist of fields in a record in a file). It is plainly obvious that both types of database have snags when it comes to access. The sequential database can usually only be looked at in a sequential manner, which means that if the file is of any length, then access time to any record will depend on the position of that record within the file. If it happens to be at the end of the file it just takes a long time to get at, tuff!! The problem with the random access file is not so much the access time, it only takes a few milliseconds to move the disk head to the correct track/sector position, but how the devil do you know which is the correct track/sector position.

So how are these problems overcome? Sequential files always are a problem, although if the data was entered in a fixed field/record format, it is often possible to read the data by random access methods. MBASIC allows this, but it's up to you, the programmer, to know what it was you were after, and it still leaves the problem of which record is to be randomly accessed. Normally speaking the only way to find something in a sequential program is to run through it from start to end looking for the match, thus, we are looking for a match with I$ from the sequential file FRED:

```
10 OPEN "I",£1,"FRED.DAT"
20 IF EOF(1) THEN PRINT "Not found": END
30 INPUT£1,J$
40 IF J$ <> I$ THEN 20
50 PRINT "Found it !!!": END
```

The loop is tight and it surprising just how fast this can be read. However, the time to access is directly related to the length of the file and the position of the record in the file. Of course the IF would usually be a little more complicated as it is not often looking for a complete match, more usually the compare is for the key which would be a part of J$. Despite this the process is still fast and can whip through an 8K file in a few seconds.

The random access file is faster in most circumstances, and shows an advantage over the sequential file after only a few records. Secondly the access time is pretty constant, consisting of the time to move the disk head (variable, depending on how close it was to its destination) and the time taken to locate and read the sector(s). Head step times vary with drive manufacturer, from 3mS per track for Teac and Mitsubishi drives to 40mS per track for the slowest of the Shugart drives. Nascom now use half height Teac drives but don't take full advantage of the head step time and end up at 6mS per track. This gives Nascom the edge in seek time over Gemini who clock in with 25mS per track with Pertec or 10mS with Micropolis drives. Not that head seek time matters much unless a lot of 'disk churning' is to be done.

As an aside, it's interesting to note that the Gemini based version of the Kenilworth portable computer uses the half height Teac drives and achieves the minimum possible head seek time of 3mS in a rather unusual way. The reason Nascom can only acheive 6mS is because the 17XX controller chip used in the Nascom disk controller card will only produce a minimum head step pulse of 6mS when used with the 2MHz clock which has to be used for 5.25" drives. The Gemini GM829 controller card also uses the 17XX series of controller chip, so on the face of it the Gemini card is stuck with the same limitation. Now it just so happens that 8" drives with their double data rate must use a 4MHz clock so a head step pulse repetition rate of 3mS could be achieved with a 4MHz clock. The Gemini card has one advantage over the Nascom, and that is software selectable clock rates, so what is done is that the Gemini written Teac routine which steps the heads changes the clock speed to 4MHz during head stepping and changes it back afterwards. Neat eh??

Anyway, back to the point. It takes typically something like half a second to locate and read any point on the disk, so a random access record could be found in that time. The only problem would be knowing which record to access, which brings us to the knotty problem of indexing. The obvious way of finding the required record is to look up the key and its appropriate record number in an array. So given that K$(X) contains the keys, R(X) contains the record numbers and N is the number of records a search for I$ would go something like this:

```
10 OPEN "R",£1,"FRED.DAT"
20 FOR R=0 TO N
30 IF K$(R) = I$ THEN 50
40 NEXT
50 GET £1,R(R)
```

This would acceptably fast as the search of the arrays would only take a second or so, but the limitation would be the space available for the arrays. Even if the contents of the key array were kept small, and the record number array made integer you could only keep track of a couple of thousand records before the dreaded OM error would occur, the reason being that although the array may only be a few K long in total, BASIC has to keep pointers to the arrays which naturally gobbles space.

Now the contents of the arrays must have come from somewhere originally, and we may presuppose that they would have been constructed as sequential files on disk and on start-up read directly into the arrays. So the information in the arrays is already available from disk. A variation on the theme would be to perform the search for the record number directly from disk, thus:

```
10 OPEN "R",£1,"FRED.DAT"
20 OPEN "I",£2,"FRED.NDX"
30 IF EOF(2) THEN PRINT "Not found": END
40 INPUT£2,J$
50 IF LEFT$(J$,4) <> I$ THEN 30
60 GET £1,VAL(RIGHT$(J$,4))
```

Now this would be somewhat slower, but has the advantage that any number of records (within the limits of disk size) may be accomodated. Notice that the key string is the first four bytes of J$ and the record number is held as a string in the last four bytes of J$. The major snag is updating the index.

A further thing about indices is that they may of course alter the order of a random access file. For instance, we wish to look at the random records in alphabetical key order. It would be a major job to sort any sizeable random database into alphabetical order, whereas sorting the index into order and then reading the random records in sequential index order would be a lot less time consuming albeit somewhat slower on display as each random record would have to be read separately and there would inevitably be lot head shifting to be done. The problem arises in sorting the sequential index, or getting it into order in the first place. Sequential files are not easily modified. Any change requires the file to be copied entirely which becomes a bit tedious (if not time comsuming). For instance, let us suppose we are keeping FRED.NDX in alphabetical order and wish to insert a new key entry, I$ with its appropriate record number I. We must look through FRED.NDX sequentially, test if I$ is greater or lesser that the current input J$, if it is not write out the current J$ to a new file. If the current J$ is greater than I$ then write out the current I$, followed by the current J$, and then continue copying J$ to the end of the file. Something like this.

Another aside, I'm writing these little routines straight off the top of my head, so they are untested and there are liable to be bugs or syntax errors, so, for any mistakes, you have my apologies, and if you care to write a complete database handler (without mistakes) from these routines we may well publish it. Anyway the sequential file alphabetical update:

```
 10 OPEN "I",£1,"FRED.NDX"
 20 OPEN "O",£2,"FRED1.NDX"
 30 IF EOF(1) THEN 60
 40 INPUT£1,J$
 50 IF J$ <= I$ THEN PRINT£2,J$: GOTO 30
 60 I1$=SPACE$(4): LSET I1$=STR$(I): I$=I$+I1$
 70 PRINT£2,I$
 80 IF NOT EOF(1) THEN PRINT£2,J$
 90 IF EOF(1) THEN 110
100 INPUT£1,J$: PRINT£2,J$: GOTO 90
110 CLOSE
120 KILL "FRED.NDX": NAME "FRED1.NDX" AS "FRED.NDX"
```

Now a number of different ways of doing this is possible. This routine uses two loops, but it is entirely possible to make the routine smaller using one loop and a flag, or flag out the J$ <= I$ test after the insertion has occurred. There are lots of ways of doing this, it just depends on what grabs you at the time. The nett result is a new file FRED.NDX with the new I$ inserted at the appropriate place.

The final aside for this time, I've just phoned Paul and he says he wants this stuff by tomorrow lunch time (otherwise he won't pay for lunch), and as it's five past midnight I'l have to draw the line here:

The line ---> --

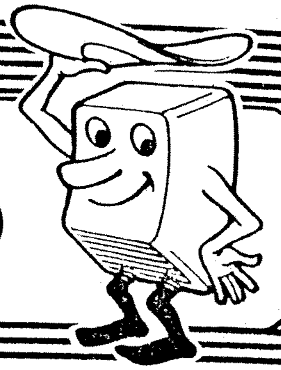
CUBEGATE LIMITED
Trading as

HENRY'S

404-406 Edgware Road, London W2 1ED

COMPUTER DEPT.
01-402 6822

OFFICE
01-724 3565



2 IVC HI-RES

Specially written for the Gemini Multiboard computers, the Gemini Galaxy range, the Quantum 2000, the Kenilworth Personal Computer and CP/M based Nascoms fitted with the Gemini GM812 IVC. This software provides pseudo high resolution graphics on a 640 x 250 matrix. This is achieved by reprogramming the video control processor and mapping the programmable character set generator to the screen.

Commands provided:
Select mode (48 or 80 column)
Clear graphics screen
Select decimal or binary coordinates
SET, RESET, INVERT and TEST point X, Y
LINE SET, LINE RESET and LINE INVERT line X, Y to X1, Y1

PRICE £15 PLUS VAT (£17.25 INC VAT)

3 DISKPEN VERSION III
LOW COST WORD PROCESSOR

DISKPEN is the answer to all small word processing needs. This latest version of DISKPEN has been developed as a result of extensive research into the user requirements, and the major enhancements have been introduced in response to feedback from the previous versions. In its standard configuration it is intended for use with the Gemini Galaxy range of computers, the Quantum 2000 range, the Kenilworth Personal Computer and Nascom CP/M based computers using the Gemini GM812 Intelligent Video card. Versions for the British Microcomputers Mimi G803 and the Superbrain, as well as many other popular computers, will follow shortly.

DISKPEN, used with these relatively inexpensive general purpose computers, provides a cost effective alternative to the larger, often under-utilized word processors. It is ideal for the preparation of letters, drafts, and documents of all kinds. DISKPEN is particularly easy to use as it is designed for users familiar with the normal office typewriter but otherwise untrained in the use of word processors.

DISKPEN also incorporates a number of optional extensions for more specialist roles. These provide an integrated approach to the requirements of not only the small office, but also the home and the laboratory. The MAXiFILE extension gives DISKPEN the ability to be used as a free field database controller for the keeping of stock records, staff records, for use as a day book, home record keeping, cross reference index, logbook, etc. The only limitation on the use of MAXiFILE is the users imagination. The SPooLER extension is particularly useful in the preparation of repetitive letters, allowing the printing of one document whilst the next is being prepared. MULTiFORMAT is a further extension for printing columns of text. This may be used for the presentation of price lists, etc.

Two other extensions are undergoing field trials, one, SPeLLAID is a proof reading spelling correction extension, whilst CALcPAC adds arithmetic capability to DISKPEN and MAXiFILE for use as an accounting and general invoicing package.

DISKPEN	price **£50.00** + VAT (£57.50 incl VAT)
CALcPAC	price **£20.00** + VAT (£23.00 incl VAT)
MULTiFORMAT	price **£15.00** + VAT (£17.25 incl VAT)
MAXiFILE	price **£20.00** + VAT (£23.00 incl VAT)
SPooLER	price **£20.00** + VAT (£23.00 incl VAT)
SPeLLAID	**price to be announced**

All UK Post Free

MDIS, IVC HI-RES & DISKPEN Available from sole worldwide distributors

Cubegate Ltd T/A Henrys 404-406 Edgware Road, London W2 1ED
Phone 01-402 6822 (Computer dept.)

HENRY'S COMPUTER SHOP
OPEN 6 DAYS A WEEK
WELL WORTH A VISIT

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