

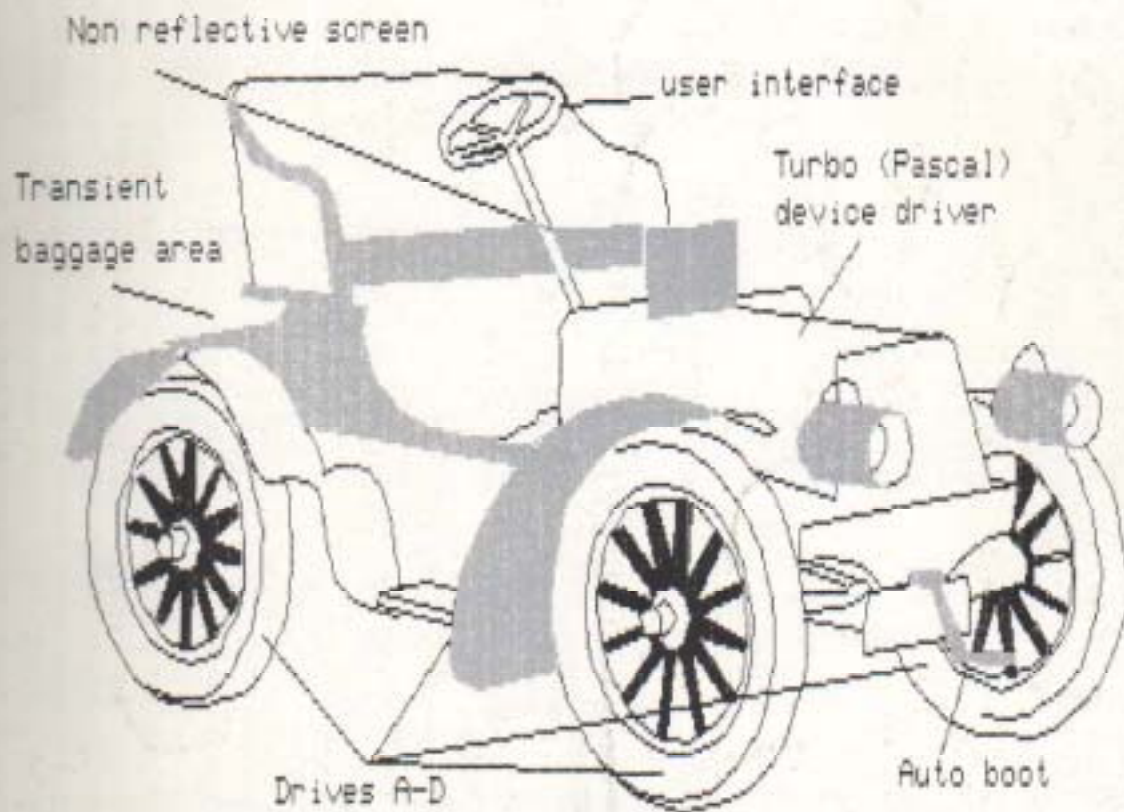
80-BUS NEWS

NOVEMBER-DECEMBER 1984

VOL. 3 ISSUE 6

- **DIY EPROM Programmer**
- **RP/M Version 2.3**
- **A look at 'C'**

ANNOUNCING THE NEW GM999 - THE GEMINI C6



**The Magazine for
GEMINI & NASCOM USERS**

£1.50

November–December 1984.

80-BUS NEWS

Volume 3. Issue 6.

CONTENTS

Page 3	Editorial
Page 4	Letters to the Editor
Page 7	Doctor Dark's Diary - Episode 23
Page 9	RP/M - Version 2.3
Page 11	DIY EPROM Programmer for Nascoms
Page 23	More Dave Hunt Bits
Page 27	A 'C' Programmer Exposes Himself
Page 34	What the General Public have said about Lawrence
Page 35	Sinclair Speaks to Nascom
Page 37	The Questionnaire Prizes
Page 38	Interrupt Driven Printer for Gemini
Page 41	DIY Economy Logic Probe & Nascom Monitor Pager
Page 42	1797 FDC Technical Note
Page 43	Put a Real Time Clock in your CBIOS
Page 49	Private 'Wanted' Ads
Page 52	ASCII Character Codes Chart
Page 52	Advertisement
Page 53	Lawrence and the Polydos User Group
Page 54	Advertisements

All material copyright (c) 1984/1985 by Gemini Microcomputers Ltd. No part of this issue may be reproduced in any form without the prior consent in writing of the publisher except short extracts quoted for the purposes of review and duly credited. The publishers do not necessarily agree with the views expressed by contributors, and assume no responsibility for errors in reproduction or interpretation in the subject matter of this magazine or from any results arising therefrom. The Editor welcomes articles and listings submitted for publication. Material is accepted on an all rights basis unless otherwise agreed. Published by Gemini Microcomputers Ltd. Printed by The Print Centre, Chesham.

SUBSCRIPTIONS

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

Subscriptions to 'Subscriptions' at the address below.

EDITORIAL

Editor : Paul Greenhalgh

Associate Editor : David Hunt

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

ADVERTISING

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

PRIVATE SALES

Free of charge to Subscribers by sending to the address below

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

EDITORIAL

Spelling

From time to time people write in and complain about the spelling mistakes in the mag. Well I'm sorry if they bother you, perhaps you would like to volunteer to do the proof-reading? Even if I do say it myself (as I am sure that no one else will) my own spelling is pretty good. It's just that on the computer screen it is easy to miss various errors, the resulting print out goes to the printers, the finished magazine comes back, I open up a copy to see what it looks like, and lo and behold the first page that I look at has an error staring at me. This could probably be solved by me re-reading everything a couple of times after I've printed it out, but what, with a more-than-full-time job to hold down, a small amount of socialising to keep me sane, a house to run, these magazines to produce, and still only 24 hours in a day well, let's just say that a few spelling mistakes here and there are right at the bottom of my priority list, and I hope yours too. (You can now put your hankies away!)

Disks not tapes

What would help a lot would be if those of you with disk systems who submit articles could send them in on disk. I'm not sure why, but some articles about disk related topics, and thus obviously from disk system owners, arrive on cassette (and sometimes just on paper!). Now I recognise that cassettes (and paper) are cheaper than disks, but I do send disks back, honest! And it is much easier to read a disk than it is to find a cassette recorder, a lead, a program that will read the particular tape format being submitted, etc, etc. And as for having to type the whole lot in from scratch.... So please, if you are sending in an article and can send it in on disk, in virtually any format or size, then please do so. It makes life so much easier, and improves the chances of the mags. being out on time.

Bulletin Boards

In an article in this issue Dave Hunt mentions the possibility of a telephone bulletin board being set up by a dealer. Well one has been set up since he wrote the article. Try 0272-421196 between 5.30 pm and 9 am, 300 baud.

Reading the small print

A little while ago I tried an experiment with small print with a view to reducing our increasing costs, and asked for your reaction to this. (See Vol. 3, Issue 3, pages 31-33.) Well a number of people have kindly responded. Some said that they thought that it was fine, but the majority wondered whether I was being paid by some opticians' society! The reaction, therefore, was not favourable, and many said that they would rather see an increase in subscriptions than see the magazine reduced to this level (pun intended!). As the price of the mag. has never changed since it started three years ago this seems quite reasonable to me! Another reaction was that some people would prefer to see the magazine appearing regularly, even if this meant less frequently, and that the size could be increased so that the overall quantity of material per annum remained constant. This would have the advantage of reducing postage cost (fewer magazines) and printing costs (fewer larger magazines are cheaper to print than more smaller ones).

I am therefore considering a combination of all of these comments, and you will see the first signs of the new format, whatever that may be, with the arrival of the first 1985 issue. (Hands up whoever added "in 1987".)

Letters to the Editor

Nascoms, Polydos, Graphics & Forth

The recent arrival of 80-BUS News (July-August 1984) and some of its comments both served to remind me that a letter to the Editor was overdue on at least two counts: I have yet to confess to being a Nascom/POLYDOS user and have also failed to submit a completed Questionnaire.

I am not quite sure why I chose POLYDOS other than that I needed (wanted) to replace tape storage with disks and that, since I have no business use for my Nascom, POLYDOS seemed friendly, had a nice editor, an adequate assembler and some useful extensions to ROM-BASIC in the bundle. (It is also easy to convert POLYDOS files to P-system files.) However, like Mr Webber of Lewisham, I wanted something other than BASIC as a language. This was not simply because I wanted to replace BASIC with a structured and/or compiled language but also because, when one's Nascom memory map has POLYDOS, POLYDOS-BASIC, ROM-BASIC, NAS-SYS and the Nascom AVC G32 code in it, there is precious little room for any BASIC code. The SET-CHAIN command of POLYDOS-BASIC helps but it struck me that something altogether more compact was required for non-trivial graphics programs. I have tackled this problem on two fronts: adapting Hi-Soft PASCAL and installing a home-brew FORTH.

Again like Mr Webber, I found that, even when armed with the Alteration Guide, adapting a tape version of Hi-Soft PASCAL to disks was far from straight-forward. Nevertheless, with the expenditure of more time than I care to recall, it has been accomplished to some extent. The modified version handles Hi-Soft 128 byte record files for the tokenised source code and will convert files output by the POLYDOS editor to this form. (I have yet to write the de-tokenising program but that is surely trivial.) As for data files, I have implemented three primitives OPEN, CLOSE and BLKIO which, with the PASCAL routines GETC(har) and PUTC(har), enable me to read text files. Naturally, using BLKIO one could read numeric data in binary form but I don't usually want to. To drive my Nascom AVC, I have set up a number of types, especially COLOUR(!), and routines, such as LINE, to drive the G32 code. It all works but even now there is a space problem in that I cannot develop graphics programs of any size interactively. My last program, which designs MAT-arrays for the AVC on a grid, was compiled as two lines of code each of which was an include statement!

FORTH, like APL, fascinates me because of its extensibility and, unlike APL, ease of implementation. I put up the first version using R G Loeliger's book "Threaded Interpretive Languages". (This was reviewed in 80-BUS News a little while ago and is the only computing book I have thumbed to scruffiness.) Doing this was hard work on a tape Nascom before getting the ZEAP assembler, and then POLYDOS. Version 2.0 is nearly complete but the luxury of disk-based screens means that I can tinker forever with, for example, the ASSEMBLER or EDITOR. Like PASCAL this is used to drive the AVC but in a style more reminiscent of LOGO and TURTLES.

More generally on the AVC, I have started to think about driving it directly rather than through G32. This is not to gain more speed on the standard geometrical constructs but for SCROLLING, PANNING and sprites. The first two are easily done for the whole screen using the MC6845's own commands though not, I suspect, in the middle of a FILL sequence if one wants to use the edges to make the picture appear continuous. For similar windows the Z80 command LDIR would have to be flogged to death. Sprites should just be a matter of copying blocks onto the AVC memory.

At the moment, I have a problem with the system's memory on which I would be grateful for help. I have RAM socketed on the N2 at D800H-DFFFH and the

POLYDOS ROMs at D000H-D7FFH with the links at LKS1 placing MROM, VRAM and BROM as normal. Pin 10 for D000H-DFFFH is connected to both pin 7 (XROM) for the POLYDOS ROMs and pin 4 (BLOCK A) for the RAM chips in sockets IC37 and IC38. This all worked with a 64K Gemini RAM Card until the AVC was installed. The memory card was amended as the AVC manual demanded but I did nothing to the N2. As a result (?), there is a gap in memory from D8000H-DFFFH. I suspect /RAMDIS is the culprit but am not sure how to use the advice in the AVC manual to recover the lost RAM. Eventually, I want to use the spare parallel ports on the Gemini GM816 (see below) to handle this linking - then I could switch ROM BASIC in or out.

I am sorry about the delay in returning the Questionnaire, but writing letters to the Tax Inspector and even editors of computer journals comes a long way after programming. Specific answers to most of the questions are appended to this letter. In general terms I very much enjoy reading 80-BUS News (though inclined at times to mutter about too much of the software stuff being CP/M orientated) but am becoming alarmed at the numbers of Nascoms now on the second-hand market.

Disloyal it might seem, but I have acquired a Sinclair QL so that I can get my hands on something closer to a proper operating system and am now setting up an RS232 link to the Nascom using a Gemini GM816 and a GM818. Does this make the QL an expensive printer buffer or the Nascom an extremely expensive disk drive?

Yours truly, C J Cave, South Croydon, Surrey

Reading the small print

Reaction to layout of Dave Hunt's ramblings, 80-BUS News, Vol 3 - Iss 3:- **"Aaarrrrggghh!"**. Frantic scrabble for magnifying glass. This layout is not a good idea. Reducing the number of pages is also not a good idea. We don't want a return to INMC Issue 1. Besides, how would you have room to publish the design of my 80-BUS Orgone Accumulator, if you cut down the length of the magazine.

The 80-BUS News has cost £1.50 since January 1982. This is £9.00 a year. I must have spent quite a bit more than £300 on my beloved Uriah in the past year. Compared to this expense, £9 for 6 issues of 60-odd pages of information, including reviews of hardware and software (which could well prevent a costly mistake) is nothing. As even 2000AD has had its price put up by 20% in the past year, are all the 80-BUS freaks out there REALLY too stingy to shell out a few extra quid?

I have been trying for more than a year to start a tape circle, but none of my offers have yet been published in the 80-BUS News. How do you expect us "dodos" to be anything else if you suppress us?

Yours truly, Kevin Wood, Canterbury, Kent

80-BUS & BBCs

About Mr Young, the person who wishes to attach a BBC to the 80-BUS (what a repulsive thought!). What does he mean by the 1 MHz tube?

There is a thing called the 1 MHz bus. This buffers the lower 8 bits of the address bus out of the BBC and the data bus bi-directionally. There are also two select signals provided (which each map the bus onto one page of memory, from £FC00 to £D00), reset, the Read/Write line, a 1 MHz clock and an interrupt line. There is also an input to the BBC's internal amplifier (which is responsible for the "frying eggs" sound of the BBC). The 1 MHz bus is totally putrid.

There is also a thing called "The Tube". This is even more putrid, and is smothered in registered trade mark signs, although why anyone would want to

claim responsibility for this is beyond me. It consists of the lower 7 address bits, the data bus, reset line, interrupt request, 2 MHz clock, Read/Write line and a select line. All the lines are unbuffered, and are connected directly to the 6502 wherever possible. Incidentally, there is less buffering on a BBC than on a Nascom 1.

The 1 MHz bus may be thought of as a set of uncommitted ports, and the Tube as a way of talking to the BBC, when the BBC can be bothered with you. A second processor for the BBC uses it to talk to the world. Imagine a Gemini IVC/SVC with optional networking and disk controller.

Perhaps if you were to tell us why you want to interface a computer with a toy, more information could be provided. Try the BBC advanced user guide, as this should be able to help you considerably, assuming that you can write controlling software.

On the subject of BBCs, I would like to say that Dave Hunt put his comments on BBC "BASIC" very well, and the only fair criticism that I feel could be made of his article, is that he didn't criticise the BBC "BASIC" enough. Perhaps Mr Hellen would be able to appreciate this more if he had bought Pascal.

Finally, in my last letter, I suggested putting up the subscription to maintain the quality of 80-BUS News. As I have just paid my subscription, I realise that I am clear of this prospect for another year. However, I still feel that this is an appropriate step, considering the reduced length of the July/August issue.

All the usual garbage, Kevin Wood, Canterbury, Kent.

More on BBCs

I am writing regarding Chris Hellen's letter on BBC BASIC. I have used this system on my BBC/TORCH system and with dual disks and 64K of memory it serves all my needs.

However, I still have my old CRASH AT 2MHz NASCOM 2 with built-in amnesia and have been having thoughts that a transfusion via the RS232 port would be a better bet than the bus/tube system proposed, as timing will be difficult.

If anyone wants to join the insanity trail, please ring 0992-25386 and join the club.

Yours truly, G S Chatley, Cheshunt, Herts.

No Improvement

Thank you for publishing my letter last year, it is a pity that you felt unable to answer it. [Ed. - see Vol. 2, Issue 4, page 4 where Mr Brown complained about a distinct lack of 80-BUS News, but kindly enclosed his resub. all the same!!]

This year I received a subscription reminder in April, dated May/June, whereas the last copy of last years subscription did not arrive until December. Is there any hope of improvement or are you assuming that no one has noticed?

This time as an act of my good faith [Ed. - ?????????] I have delayed sending my cheque until actually having received my full compliment of issues from last time.

Yours hopefully, J. Brown, 21 Mowbray Rd., Didcot, Oxon. OX11 8ST.

[Ed. - the receipt of this issue (assuming it returns from the printer in a reasonable time) should put us only half->one issue in arrears. As for 1985, you will have noticed that for some time the subscription has been stated to be for 12 months, and not for a specific number of issues.]

DOCTOR DARK'S DIARY – Episode 23.

NASIO Nasties!

Thanks to Adrian Perkins' article in Volume 3 Issue 2, I have stopped a mysterious fault that was occurring on my system. I would go into great detail about the various combinations of boards that would or would not work with the two different CP/M's I use, but it is so complex to explain that I prefer not to go on about it in too much detail. Suffice to say that I was blaming my Belectra board when it was not to blame. On putting the old Nascom I/O board back into use, I found no more trace of the fault. I suspect, and that is as far as I get with my limited hardware knowledge, that the GM809 disk controller board I am using, which I thought was supplying NASIO, is not doing it in the recommended way. I don't understand all this open collector business too well, so I may be wrong. Perhaps an expert on that board will enlighten us?

MONITOR.COM NASTIES!

Nice to see that Adrian has also fixed the bugs in MONITOR.COM for us. I knew about the fault where the last block of anything was loaded twice, but didn't bother to find out why it was happening. The reasons for this slapdash approach? Well, it was doing what it was supposed to do, and that was load in the Nas-Sys based games I wanted to run, and the extra block was doing no harm. My children, I caution you against spending all your time improving the operating system software, when you could be writing application programs for money, or even better, playing adventure games! And no, I didn't do my kludges in 8080 code, Adrian. I did them on paper in Z80 assembly mnemonics, because the program was too small to bother with using assemblers, linkers and the like. I like programming in Z80, and when there were only Nascom 1's with 1K, I used to do all my programming directly. Don't be misled by the fact that the block moves are not used where the two sections of RAM get interchanged. If there was a Z80 operation that would swap two blocks of memory, I would have been delighted to use it. Incidentally, if anyone is interested, I have a labelled, commented disassembly of my code on disk. I sent it out on the Ring of Iron, but that seems to have gone into hyperspace. If anyone wants it for the purpose of making improved versions of Nas-Sys for themselves, or indeed everyone, then send me a disk (Pertec drives, DDDS).

More thoughts about monster programs.

Having written up my ideas on chaining programs in Hisoft Pascal, I was thinking about ways of writing groups of large programs, when I realised that straightforward chaining may not be enough. Suppose you wanted control to be returned by a program to the program that had activated it, no matter what that program was called? It seemed to me that what was needed was some sort of equivalent to GOSUB in BASIC, or procedures in Pascal. It would be nice, I decided, if recursion was possible. All this thinking made my brain hurt, so I did a bit of drinking instead, and soon realised that a file that behaved like a stack was needed, along with a couple of procedures called "call" and "return".

What "call" would have to do would be as follows. Create a file called "\$\$\$\$.SUB" and put the name of the program to be called in it, complete with all the extra, unexplained spaces I mentioned last time. Add the name of the currently running program to the "top" of the stack file. End the current program.

Similarly, "return" would behave as follows. It would remove the name of the program to return to from the top of the stack, and put it in a file called (surprise) "\$\$\$\$.SUB". Then it would end execution of the current program.

These two procedures would be included in each of the programs in the system of programs, so it is fortunate that Hisoft have put a "library file" facility in their compiler. You only have to type them once, into files with ".LIB" extensions.

(Just a thought, in the midst of the white heat of Pascal monster programming, is all this stuff interesting to YOU, or are you bored by it? Do you wish I would go back to writing about hardware? (Not what I am best at!) Help shape the magazine by answering these questions, if you feel like it. Otherwise I could ramble on for ages with no audience! (I wrote this before seeing the recent survey, but it isn't necessarily redundant, as you can always write to me as well as doing the survey. No prizes for writing to me though!))

The other thing each program in the system needs to know, Pascal being what it is, is the names of all the programs that are likely to get called. At least, this is so the way I have done it. There may well be lots of better ways, and if you know them, you can save me some time by telling me where I can find them! Again, use the library facility to include the names in each program. Then when you add a program to the system, you only need to edit the library file, and recompile all the programs.

As another aside, and by way of partial explanation of the reasons why I want to be able to produce systems of programs the size of my disks, rather than sticking to the 64K limit of the Z80, I have a question I hope one of you can answer. The run-time routines for HP4 are quite a lot larger than those of HP5. So the same program, compiled by each of the compilers in turn, should be smaller when HP5 is used. Well it isn't. I noticed this some time ago, and have just retried it, with the result that an 8K source file compiled to 54 records with HP4, and HP5 made 63 records of it. The effect is even more marked with some programs. Anybody know why? I would ask Hisoft, but I don't like to disturb them in case their super programmer, David Link, is working on a compiler for Modula 2, the language I want to learn next! Ada? Don't make me larf...

Loops, Rings and Mobile Rust.

Nascom 1 users who are interested in setting up a tape based software swapping group should write to Kevin Wood, 24 Hudson Close, Sturry, Canterbury, Kent, CT2 0HX. He seems to be a hyperactive programmer, so there should be some good programs going round that loop.

A disk called Son of Ring of Iron arrived here not so long ago, with some interesting new utilities on it. There was a version of Pacman for 80 column displays, which looked very strange on my old 48 column tube. A program to enable modem users to access Prestel, looked interesting, and I would have liked to read the source file for that one, but it wasn't there! A program called 512X512.COM would do nothing on my system, probably something to do with a board I have not got.

And that's all you get this time! Sorry, but I am very busy with some software for fucre, and anyway, who knows which year this will get to be printed?

RP/M VERSION 2.3**By Richard Beal**

RP/M is one of the two ROM monitor programs supplied with Gemini GM811 and GM813 CPU cards. It is designed for use with or without disks and is particularly suited to users who wish to develop and test software under CP/M which is to be placed in ROM and used in dedicated systems which do not have disks. It is however able to boot a floppy disk based system and is therefore suitable as a general purpose boot ROM. (It can boot a hard disk based system via a floppy disk, but cannot directly boot a hard disk.) It also has a variety of simple commands which allow many hardware problems to be analysed if the disk system fails to operate.

The other ROM monitor program supplied by Gemini is SIMON which is specifically designed for use in disk based systems and has the ability to boot both floppy and hard disk drives. It too is designed to allow diagnosis of hardware problems.

RP/M was originally issued as version 0.1 and this was later replaced by version 2.0, which had many improvements. See 80-BUS NEWS Vol 2 Issue 1 for details. This issue also specifies a small patch to convert version 2.0 to version 2.1.

This article provides a series of patches to version 2.0 or 2.1 so that if you have the ability to program 2732 type EPROMS then you can upgrade your copy of RP/M to an enhanced version, called 2.3. (To avoid confusion with CP/M V2.2 there is no RP/M version 2.2.)

Version 2.3 allows the Boot command to be followed by an optional value which makes RP/M attempt to read in and execute a boot sector which is on the first track of any disk in the system. It is possible to have the boot sector on a 5.25 inch or an 8 inch floppy disk. It may be either single or double density. Also the sector number may be specified. For example a normal Gemini format 48 or 96 tpi double density system disk with a 512 byte boot sector in physical sector 0 in drive B could be booted. So could a standard single density 8 inch disk with a 128 byte boot sector in physical sector 1.

A particular advantage of this ability is to check out the correct operation of the disks should there be a problem with booting the system. Before this if the system failed to boot with a "Bad disk" message then it was impossible to determine if the problem was in the disk controller or the disk drive without dismantling the drives and altering their physical addresses.

Note that since the boot sector on any disk is likely to contain code which loads in CP/M from drive A, it is still not possible without a special version of CP/M to actually boot CP/M and use it if drive A is out of operation. Even if the boot sector is patched so that CP/M is loaded from the other drive, as soon as it executes it will attempt to log in drive A.

Also note that RP/M loads and executes only the first 128 bytes of a boot sector, regardless of its length. For this reason it is not possible to load a double density boot sector from an 8 inch disk, for timing reasons associated with the Z80 code. Also, as before, if the boot sector does not start with the Gemini identifying characters then the "Wrong disk" message is displayed.

RP/M will still attempt as now to boot a standard Gemini disk in drive A on power-on if there is a disk card in the system. Also the normal B command with no values entered after it works exactly as before.

The format of the Boot command is now as follows:-

B Boot drive A, double density, sector 0, 5.25 inch
 or
 B sscd Where sscd is a single hexadecimal value.
 Leading zeroes may be omitted.
 ss is the physical sector number to be read. Normally this is 0 or 1.
 c is the command digit which has the following possible values:-
 0 5.25 inch, double density
 1 5.25 inch, single density
 3 8 inch, single density
 d is the drive select digit which has the following possible values:-
 1 drive A
 2 drive B
 4 drive C
 8 drive D

Examples of the enhanced Boot command

B 2 Sec 0, 5.25 inch, DD, drive B
 B 111 Sec 1, 5.25 inch, SD, drive A
 B 134 Sec 1, 8 inch, SD, drive C

PATCHES REQUIRED

First make the patches to bring V2.0 to V2.1, if needed.

<u>Address</u>	<u>V2.0</u>	<u>V2.1</u>
F068	30	31
F108	20	ED
F109	04	41
F10A	ED	20
F10B	41	02

Then make the patches to bring V2.1 to V2.3.

At F068 change 31 to 33
 At FB08 change 3A 60 00 to CD EF FF
 At FC03 change 11 99 FC to 3D 20 FD
 At FC11 change 20 27 3E D0 CD 8C FC 3E 01 D3 E4 3E 5B
 to 11 99 FC 20 24 3E D0 CD 8C FC CD DE FF
 At FC42 change AF D3 E2 to 00 00 00
 At FFDE there are FF characters to the end - change these to
 3A 61 00 B7 20 01 3C D3
 E4 3A 62 00 D3 E2 3E 5B
 C9 3A 60 00 B7 20 06 21
 00 00 22 61 00 FE 02 D0
 AF C9

These changes have been carefully checked and the results fully tested, so you should have no difficulty. You can check that you have typed in the changes correctly by using the standard CRC calculating program. For your convenience this program has been run separately on the first 2K of RP/M V2.3 (CRC = EF 24) and the second half (CRC = 58 CD). Therefore you can easily confirm that your new RP/M 2.3 is correct. The CRC calculating program is called CRC.COM and is the subject of another article in this issue.

EPROM PROGRAMMER FOR NASCOMS

By Stephen Weir

This article describes the construction of a very cheap EPROM programmer for the Nascom owner who, like me, has to run his Nascom on a budget. It can be built for less than £20 including the cost of a zero insertion force socket.

Most inexpensive programmers that use only one PIO generate the address lines from a CD4040 12-stage counter giving a range of 4096 (2^{12}) addresses, i.e. the largest EPROM that they can handle being the 2732. However, the 2732 is no longer cost-effective, the best value as far as bits per pound at the moment is given by the 2764, and the 27128 will no doubt fall in price.

I wanted to get the programmer built and working as quickly as possible and the following points were the main constraints on the design:

1. Low cost
2. Minimum hardware
3. Simple software
4. Capable of programming up to 27128

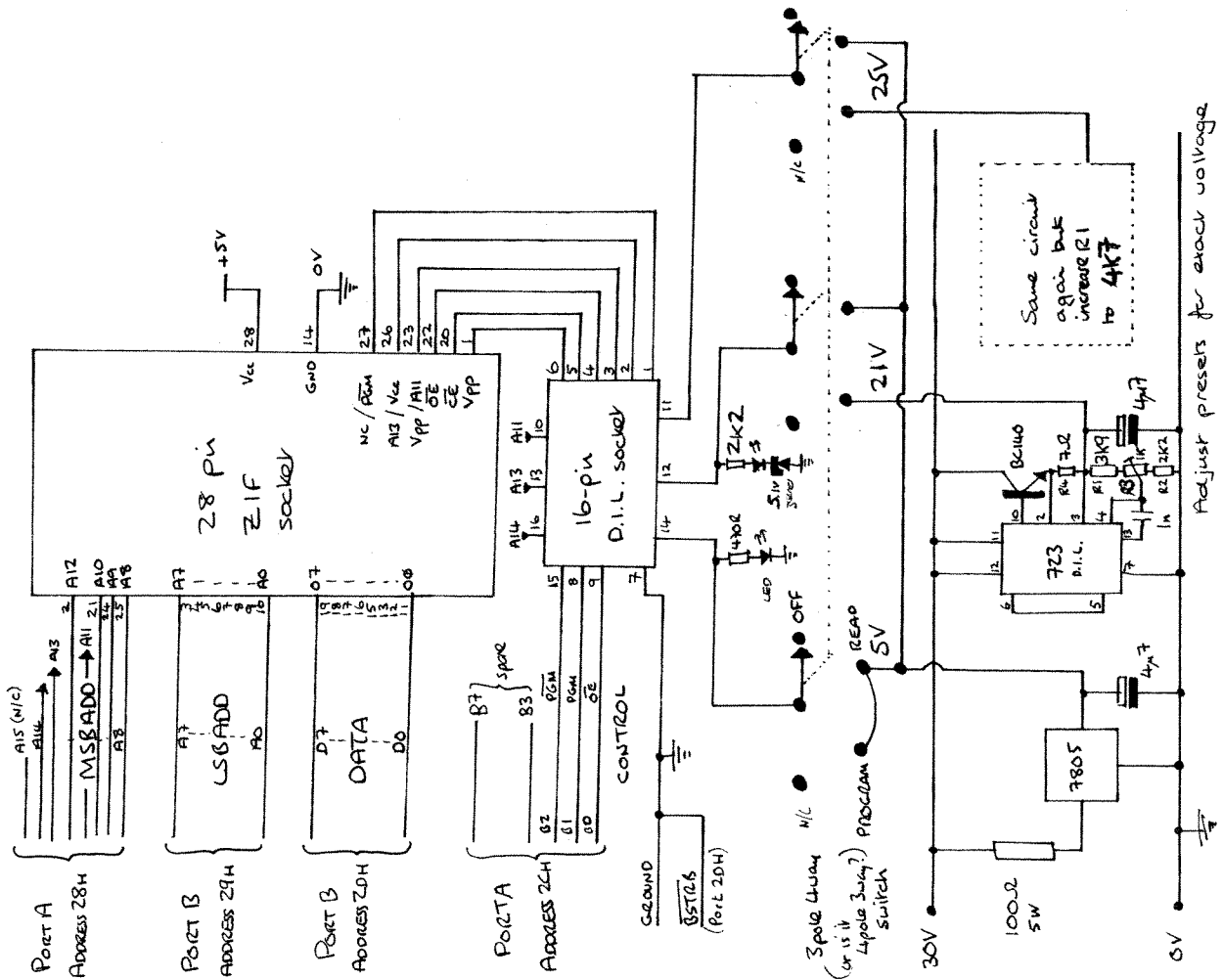
With low cost being the main priority, points 2 and 3 naturally follow. Minimum hardware means that there is no automatic switching of power supplies or selection of EPROM type. Power supplies are switched manually and the EPROM type is selected by inserting a suitably wired header plug into the selector socket. Since all switching is manual the software can be very simple. However certain 'awkward' EPROMs do not follow the usual pattern, particularly the 2732 which does not have a 'VERIFY' mode and uses a common pin connection for /OE and Vpp. This would require additional circuitry to do the switching so I just left the 2732 out. I don't find this a problem since I doubt if I will ever use a 2732, unless somebody gives me some! Its brother, the 2532 is also a bit awkward since it does not have a 'VERIFY' mode either. In this mode the contents of the EPROM may be read with Vpp at +21V (or +25V) so that each location may be verified immediately after programming without having to switch off Vpp. The original control program I wrote uses this feature so that any faulty locations are reported immediately. However this meant that the 2532 was excluded also. So, in order to include at least one 4K EPROM, I modified the program to do all verifying when programming is complete so that Vpp can be switched off before verification if a 2532 is being programmed. Thus the following EPROMs can be programmed :- 2716, 2516, 2532, 2764, 27128 (and perhaps the 27256, but I don't have any data on this one and I think it uses a different Vpp).

Leaving everything to the human operator is, of course, asking for trouble, but with prompts on the Nascom screen together with warning lights to show when power is applied to the EPROM I have found it easy to get into a routine when programming an EPROM. So as long as you take some care you shouldn't have any problems - although it's probably best to keep off the home-brew while you are using the programmer!

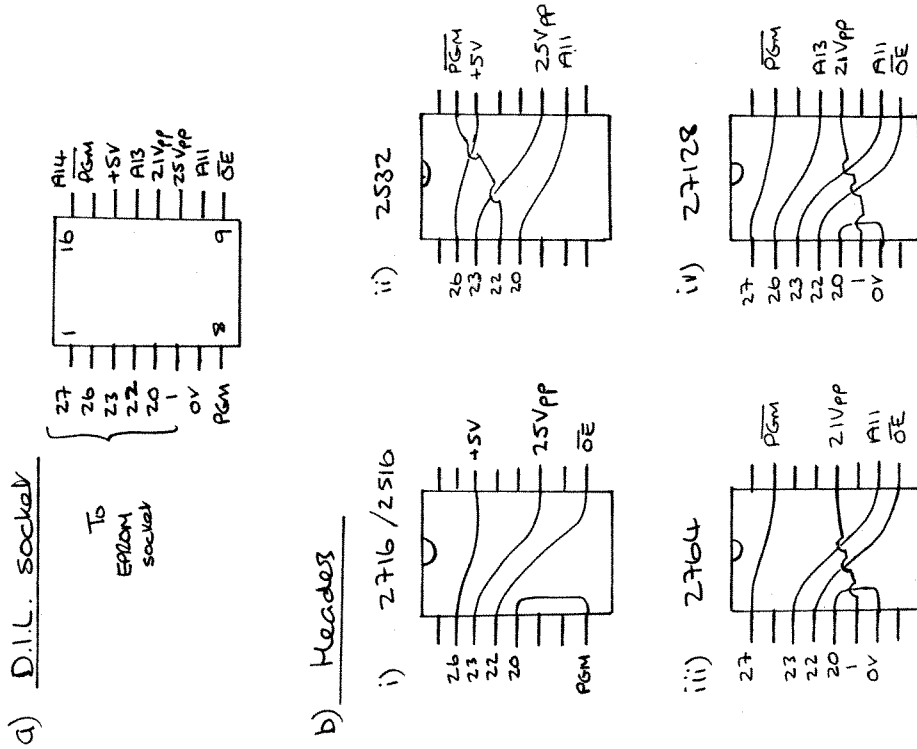
Circuit Description

Referring to fig 1, the 14 address lines are supplied from a PIO. The 8 data lines and 3 control lines are supplied from another PIO. The data lines and most of the address lines are taken directly to the pins of the 28-pin ZIF socket, since these connections are common to all EPROMs. Address lines A11

EPROM PROGRAMMER Fig1 Circuit Diagram

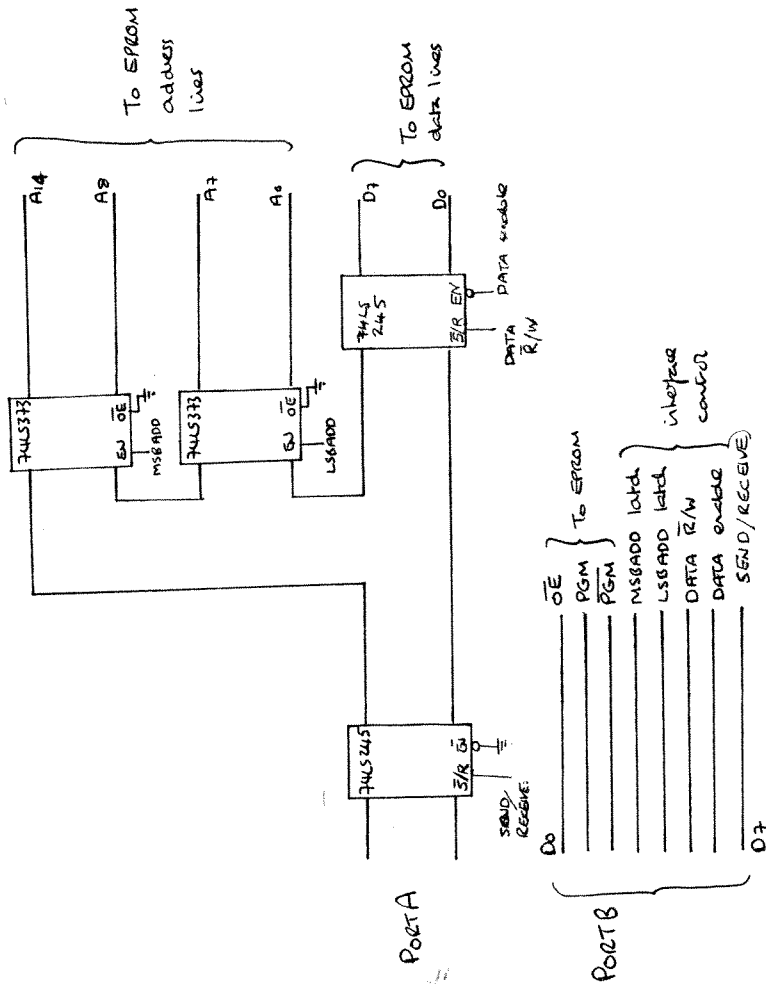


EPROM PROGRAMMER Fig2 16-pin header connectors



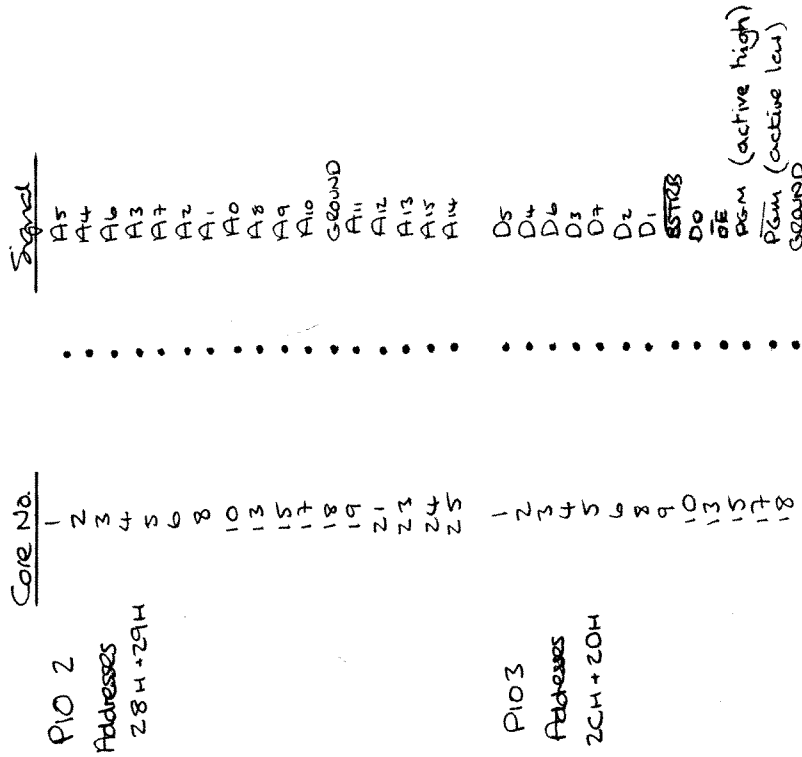
EPROM PROGRAMMER

fig3) Using one PIO



EPROM PROGRAMMER

fig4) PIO Cable Connections



and A13 (also A14 for future expansion), control lines /OE, PGM and /PGM, and power supplies Vcc (+5V), Vpp (+21V and +25V) and GND (0V) are taken to a 16-pin dip socket. ZIF socket connections that vary depending on the EPROM being used are also taken to this socket. By inserting a header plug with the required connections the appropriate EPROM may be programmed. See fig 2 for header connections. Note that some EPROMS require a low to high programming pulse whereas others require a high to low pulse - hence the use of two signals PGM and /PGM respectively.

The +5V supply is provided by a 7805 5V regulator from an unregulated 30V supply. The two Vpp voltages are supplied from two separate circuits built around two 723 regulators. Although the 723 can deliver the 30ma Vpp current I included a bypass transistor to take most of the dissipation away from the 723. The function switch has 3 positions- OFF, READ (Vcc and Vpp at +5V) and PROGRAM (Vcc at +5V and Vpp at +21 and +25V). Two LEDs indicate when Vpp and/or Vcc is applied. The 5.1V zener stops the Vpp LED lighting when in READ mode (Vpp=+5v).

Control Program

By restricting the EPROM types that can be programmed, and routing the signals via the header socket, the control program has been made very straightforward and, apart from checking to see if a 2532 is being programmed, it does not really need to know which EPROM is in the socket since the routines and control signals are the same. However, for convenience of the user you are requested to enter the EPROM type so you will be told if you are trying to program a 4K block of memory into a 2716! It also uses the EPROM type information to work out the number of locations to check for erasure and to read in the EPROM contents to RAM.

On entering the program a menu gives the options:-

1. Verify EPROM fully erased
2. Read in EPROM to memory
3. Program EPROM (and verify)
4. Return to NAS-SYS

The basic structure of each operation is a simple loop which outputs an address, sets the control lines to perform the operation, either writes data from RAM to the EPROM or reads data from the EPROM into RAM, increments the EPROM address and RAM pointer, and continues until the operation is complete. In each case you will first be asked to enter the EPROM type. The previously entered type is displayed and if this does not need to be changed just press NEWLINE.

1. Verify

This routine reads in every location from the EPROM and compares it with FFh which signifies that it is erased. If the location is not erased, a message is displayed together with the address of the offending location. If there are a lot of unerased locations you can press ESCAPE to abort the verification. Then you may try another EPROM or return to the menu.

2. Read in to RAM from EPROM

You are first asked for the start address of the RAM block where you want the contents of the EPROM to be dumped. A default address is shown so just press NEWLINE if this does not need to be changed. The whole of the EPROM will

be read in and when this has been done the NAS-SYS Tabulate command is used to display the contents of the EPROM (note: the addresses shown are RAM addresses).

3. Program

After entering the EPROM type you enter the start and end address (inclusive) of the RAM block which holds the data to be blown into the EPROM. Next, you are asked for the base address of the EPROM. The default value is 0000. If you wish you can change the base address to the actual address at which the EPROM will reside in memory. When this has been entered the start and end addresses of the EPROM will be displayed, assuming programming is to start at the first location in the EPROM. If you are programming from a different address then change the values accordingly. The next prompt tells you to switch the programmer to PROGRAM and then to press key `P`.

When programming is complete the locations just programmed will be verified against RAM by reading in each location and comparing it with the corresponding byte in RAM. If a discrepancy is detected a message is given which displays the EPROM location concerned. If there are a lot of errors press ESCAPE to abort the verification. Then you can try the same EPROM again or a fresh one.

The signals used to control the EPROM are /OE which is low to read and high to program, and the programming pulses PGM (active high) and /PGM (active low). The duration of the pulses is 50ms +or- 5ms and to implement this I used a simple delay loop. However a program crash during the middle of the delay loop would leave the PGM signals permanently active and the EPROM would not like this! So a safer alternative would be to use a 74LS121 monostable to provide the 50ms pulse and this would simply need to be triggered from the PIO.

As I already had a Nascom I/O board which holds up to three PIO's I took advantage of this by using two PIO's to provide all the necessary signals and therefore simplify the hardware. If you only have the on-board PIO but still would like to build the programmer have a look at fig3. This is something off the top of my head (yes, I know it's a funny place to keep circuit diagrams!) and has not actually been tried but something along these lines should work, although the control program will need modifying.

Construction

The circuitry can be built on a piece of Veroboard and mounted in a plastic box with the ZIF and 16-pin dil sockets poking through holes cut in the top of the box, together with the switch and LED's. Connection to the PIOs on the I/O board is made using two 26-way ribbon cables with IDC headers. See fig 4 for pin connections (but note that I use different addresses for the PIOs from those recommended in the manual). The power supply could be included in the programmer box or use a separate one. It should be able to deliver 30V at about 300ma. Wire up the header sockets as per fig 2 and I strongly suggest you clearly mark each one with the EPROM type!

```

9CA.2 22290C
9CA5 EF4F5054
9CA9 494F4E53
9CAD 00
9CAE 219B08
9CB1 22290C
9CB4 EF2D2D2D
9CB8 2D2D2D2D
9CBC 00
9CBD 210A09
9CC0 22290C
9CC3 EF202020
9CC7 20202020
9CCB 20202756
9CCF 272D2056
9CD3 65726966
9CD7 79204550
9CDB 524F4D20
9CDF 65726173
9CE3 65640D0D
9CE7 20202020
9CEB 20202020
9CEF 20275227
9CF3 2D205265
9CF7 61642069
9CFB 6E204550
9CFF 524F4D20
9D03 746F2052
9D07 414D0D0D
9D0B 20202020
9D13 20275027
9D17 2D205072
9D1B 6F677261
9D1F 6D204550
9D23 524F4D0D
9D27 0D
9D28 20202020
9D2C 20202020
9D30 20275127
9D34 2D205175
9D38 69740D0D
9D3C 00

          PRS \ DB "OPTIONS",0
LD HL,LINE2+64+17
LD (CURSOR),HL

PRG \ DB "-----",0
LD HL,LINE2+64*3
LD (CURSOR),HL

          -V~- Verify EPROM erased",NL,NL,NL
PRS \ DB "
DB "
          -R~- Read in EPROM to RAM",NL,NL,NL
DB "
          -P~- Program EPROM",NL,NL,NL
DB "
          -Q~- Quit",NL,NL,NL
DB "

;Wait for selection
RIN
MENU2 CP "v"
      JP Z,VERIFY
CP "R"
      JP Z,READIN
CP "p"
      JP Z,PROGRA
CP "Q"
      JP NZ,MENU2
SCAL MRET
9D3D CF
9D3E FE56
9D40 CA549D
9D43 FE52
9D45 CA259E
9D48 FE50
9D4A CAll19F
9D4D FE51
9D4F C23D9D
9D52 DF5B

```



```

9E69 19      ADD HL,DE
9E6A EB      EX DE,HL
9E6B DF64    SCAL NUM
9E6D 38CF    JR C,TRYAG9
9E6F 2A210C  LD HL,(NUMV)
9E72 E5      PUSH HL
9E73 DDE1    POP IX
9E75 2238A3  LD (RANTAB),HL
;Load arguments ready for Tabulate command
9E78 220C0C  LD (ARG1),HL
9E7B ED5B40A3 LD DE,(ROMSIZ)
9E7F 19      ADD HL,DE
9E80 220E0C  LD (ARG2),HL
9E83 210E00  LD HL,0EH
9E86 22100C  LD (ARG3),HL
;Wait for keypress
9E89 EF0D5377
9E8D 69746368
9E91 20746F20
9E95 52454144
9E99 20616E64
9E9D 20707265
9EA1 73732027
9EA5 52272077
9EA9 68656E20
9EAD 72656164
9EB1 7900
9EB3 DF7B
9EB5 FE52
9EB7 20FA
;Set up regs ready for loop
9EB9 210000  LD HL,0
9EBC 0E28    LD C,MSBADD
;Read in each byte
9EBE CD19A3  LOOPRD CALL RDBYTE
;Put into RAM
9EC1 DD7700  LD (IX+0),A
;Inc pointers and check for end
9EC4 23      INC HL
9EC5 DD23    INC IX
9EC7 E5      PUSH HL
9EC8 B7      OR A
9EC9 ED52    SBC HL,DE
9ECB E1      POP HL
9ECC 3BF0    JR C,LOOPRD
;Loop back if not finished
9ECE EF0D00  TABULT PRS \ DB NL,0
9ED1 DF60    SCAL ARG3
9ED3 DF54    SCAL "T"
9ED5 EF0D5072
9ED9 65737320
9EDD 27542720
9EE1 746F2054
9EE5 4142554C
9EE9 4154450D
9ED 6F722061
9EF1 6E79206F
9EF5 74686572
9EF9 2086579
9EFD 20666F72

PRG \ DB NL,"Press 'T' to TABULATE",NL,"or any other key
for MENU",0
SCAL BLINK
CP "T"
JP Z,TABULT
JP Z,BEGINA
;*****
; * Program EPROM *
;*****
;Clear the screen and write title
PROGRA CALL CLSTIT

9F11 CD71A2
9F14 EF50524F
9F18 4752414D
9F1C 0D
9F1D 2D2D2D2D
9F21 2D2D2D0D
9F25 0D00

9F27 3E05
9F29 D32C
;Set OE high PGM low and PGM high
LD A,101B
OUT (CONT D),A
;Set DATA lines to output
LD A,0FH
OUT (DATA C),A
;Get EPROM type
CALL TYPRM
;Get RAM addresses
TRYAG1 LD HL,LINE2+64*4
LD (CURSOR),HL

9F32 214A09
9F35 22290C
9F38 EF537461
9F3C 72742061
9F40 6E642065
9F44 6E642061
9F48 64647265
9F4C 7373206F
9F50 66205241
9F54 4D3F2020
9F58 00
9F59 DF63
9F5B 211E00
9F5E 19
9F5F EB
9F60 DF64
9F62 38CE
9F64 2A210C
9F67 223AA3
9F6A E5C1
9F6C DF64
9F6E 38C2
9F70 ED5B210C
9F74 C5E1
9F76 13
9F77 B7
9F78 ED52
9F7A 30B6
9F7C ED533CA3
9F80 EB
9F81 2A40A3
9F84 19
9F85 3841

PRG \ DB "Start and end address of RAM? ",0
;Wait for entry
SCAL INLIN
LD HL,30
ADD HL,DE
EX DE,HL
SCAL NUM
JR C,TRYAG1
LD HL,(NUMV)
LD (RAMSTA),HL
PUSH HL \ POP BC
SCAL NUM
JR C,TRYAG1
LD DE,(NUMV)
PUSH BC \ POP HL
INC DE
OR A
SBC HL,DE
JR NC,TRYAG1
LD (RAMEND),DE
EX DE,HL
LD HL,(ROMSIZ)
ADD HL,DE
JR C,SIZOK
;Jump if RAM <= ROM size

;Try again if end
;address < start
;DE = -length of block
;Try again if invalid
;DE has end address
;HL has start address
;Load value into workspace
;and save in BC
;Get end address
;Try again if invalid
;DE has end address
;HL has start address
;Inc DE
OR A
SBC HL,DE
JR NC,TRYAG1
LD (RAMEND),DE
EX DE,HL
LD HL,(ROMSIZ)
ADD HL,DE
JR C,SIZOK
;Jump if RAM <= ROM size

```

```

;Otherwise print message
INC DE
OR A
SBC HL,DE
SCAL TBCD3
CALL RETCUR
CALL RETCUR
SCAL INLIN
LD HL,30
ADD HL,DE
EX DE,HL
SCAL NUM
JR C,TRYAG3
LD HL,(NUMV)
LD BC,(ROMBAS)
OR A
SBC HL,BC
JR C,TRYAG3
LD (ROMSTA),HL
SCAL NUM
JR C,TRYAG3
LD HL,(NUMV)
OR A
SBC HL,BC
LD DE,(ROMSTA)
OR A
SBC HL,DE
JR C,TRYAG3
ADD HL,DE
EX DE,HL
OR A
SBC HL,DE
LD (ROMEND),DE
;Check ROM and RAM block lengths are same
LD DE,(RAMLEN)
DEC HL
OR A
SBC HL,DE
LD A,H
LD A,H
OR L
JR Z,CARYON
;Carry on if OK
;Otherwise print message
PRG \ DB NL,"Block lengths different!",0
SCAL TDEL
CALL DELETE
JP TRYAG1
;and try again
;Wait for keypress
A03F 13
A040 B7
A041 ED52
A043 DF66
A045 CD07A3
A048 CD07A3
A04B DF63
A04D 211E00
A050 19
A051 EB
A052 DF64
A054 38B6
A056 2A210C
A059 ED4B42A3
A05D B7
A05E ED42
A060 38AA
A062 2244A3
A065 DF64
A067 38A3
A069 2A210C
A06C B7
A06D ED42
A06F ED5B44A3
A073 B7
A074 ED52
A076 3894
A078 19
A079 EB
A07A B7
A07B ED52
A07D ED5346A3
A081 ED5B3EA3
A085 28
A086 B7
A087 ED52
A089 7C
A08A B5
A088 2823
A08D EF0D426C
A091 6F636B20
A095 6C656E67
A099 74687320
A09D 64696666
A0A1 6572656E
A0A5 742100
A0A8 DF5D
A0AA CD2CA3
A0AD C3329F
A0B0 EF0D5377
A0B4 69746368
A0B8 20746F20
A0BC 50524F47
A0C0 52414D20
A0C4 616E6420
A0C8 70726573
A0CC 73202750
;Wait for input
;DE points to first digit
;Get Hex value
;Try again if invalid
;HL = STA - BAS
;Try again if STA < BAS
;Store in workspace
;Get ROMEND
;HL = END - BAS
;HL = END - BAS - STA
;Try again if END < STA
;HL = END - BAS
;DE = END - BAS, HL = STA
;HL = - ROM block length
;Store in workspace
;Check ROM and RAM block lengths are same
;Carry on if OK
;Otherwise print message
PRG \ DB NL,"RAM block larger than ROM - change EPROM
type!",0
SCAL BLINK
PUSH AF
CALL DELETE
POP AF
CP "Y"
;If yes start again
;Else reenter RAM addresses
;Store block length in wspace
;Get ROM start and end addresses
LD HL,LINE2+64*6
LD (CURSOR),HL
TRYAG2
9FC0 21CA09
9FCF 22290C
9FD2 EF426173
9FD6 65206164
9FDA 64726573
9FDE 73206F66
9FE2 20524F4D
9FE6 3F202020
9FEA 20202020
9FEE 20202020
9FF2 00
9FF3 2A42A3
9FF6 DF66
9FF8 CD07A3
9FFB DF63
9FFD 211E00
A000 19
A001 EB
A002 DF64
A004 38C6
A006 2A210C
A009 2242A3
A00B (ROMBAS),HL
A00C 214A0A
TRYAG3
A00F 22290C
A012 EF537461
A016 72742061
A01A 6E642065
A01E 6E642061
A022 64647265
A026 7373206F
A02A 6620524F
A02E 4D3F2020
A032 00
A033 2A42A3
A036 DF66
A038 2A42A3
A03B ED5B3EA3
PRG \ DB "Start and end address of ROM? ",0
LD HL,(ROMBAS)
;Display default values
SCAL TBCD3
LD HL,(ROMBAS)
LD DE,(RAMLEN)

```

```

A0D0 27207768
A0D4 656E2072
A0D8 65616479
A0DC 00
A0DD DF7B
A0DF FE50
A0E1 20FA
A0E3 EF0D0D50
A0E7 726F6772
A0EB 616D6D69
A0EF 6E672E2E
A0F3 2E0D00
A0F6 DD2A3AA3
A0FA ED5B3EA3
A0FE 2A44A3
A101 0E28
A103 ED61
A105 0C
A106 ED69
A108 0D
A109 DD7E00
A10C D32D
A10E 3E03
A110 D32C
A112 C5
A113 060A
A115 FF
A116 10FD
A118 C1
A119 3E05
A11B D32C
A11D 23
A11E DD23
A120 13
A121 7A
A122 B3
A123 20DE
A125 3E4F
A127 D32F
A129 3A48A3
A12C FE32
A12E 2043
A130 EF0D4550

CARYON PRS \ DB NL,"Switch to PROGRAM and press 'p' when ready",0
WAIT2 SCAL BLINK
CP "p"
JR NZ,WAIT2

PRGR2 PRS \ DB NL,NL,"Programming...",NL,0
;Initialise registers
LD IX,(RAMSTA)
LD DE,(RAMLEN)
LD HL,(ROMSTA)
LD C,MSBADD
;Output address
LOOPX OUT (C),H
INC C
OUT (C),L
DEC C
;Output data
LD A,(IX+0)
OUT (DATA D),A
;
;Keep OE high (inactive)
;Set PGM high (active), and PGM low (active)
LD A,011B
OUT (CONT D),A
;50ms delay
PUSH BC
LD B,10
;Change to 20 for 4MHz
DELAY RDEL
DINZ DELAY
POP BC
;
;Keep OE high (inactive)
;Set PGM low (inactive) and PGM high (inactive)
LD A,101B
OUT (CONT D),A
;Do next byte
INC HL
INC IX
INC DE
LD A,D
OR E
JR NZ,LOOPX
;When finished,
;verify correct programming
;Set DATA port to INPUT
LD A,4FH
OUT (DATA C),A
;Check if 2532 EPROM
LD A,(ROMTYP)
CP 32H
JR NZ,RESTOF
;Jump if not 2532
;If it is, print message
;and wait for keypress

```

```

A134 524F4D20
A138 69732061
A13C 20323533
A140 3220D20
A144 0D537769
A148 74636820
A14C 746F2052
A150 45414420
A154 616E6420
A158 70726573
A15C 73202743
A160 27207768
A164 656E2072
A168 65616479
A16D DF7B
A16F FE43
A171 20FA
A173 DD2A3AA3
A177 ED5B3EA3
A17B 2A44A3
A17E 0600
A180 0E28
A182 CD19A3
A185 DBE00
A188 281F
A18A EF50726F
A18E 6772616D
A192 6D696E67
A196 20666169
A19A 6C656420
A19E 61742000
A1A2 DF66
A1A4 EF0D00
A1A7 06FF
A1A9 DF62
A1AB 3004
A1AD FE1B
A1AF 2808
A1B1 23
A1B2 DD23
A1B4 13
A1B5 7A
A1B6 B3
A1B7 20C7
A1B9 3EFF
A1BB B8
A1BC 202F
A1BE EF0D5072
A1C2 6F677261
A1C6 6D6D696E
A1CA 67206661
A1CE 696C6564

PRS \ DB NL,"EPROM is a 2532 - ",NL,"Switch to READ and press 'c' when ready",0
SCAL BLINK
CP "c"
JR NZ,WAIT8
;Initialise registers
LD IX,(RAMSTA)
LD DE,(RAMLEN)
LD HL,(ROMSTA)
LD B,0
;Clear not-programmed flag
;Read in byte
LD C,MSBADD
CALL RDBYTE
;Verify byte
CP (IX+0)
JR Z,NEXT1
;If OK
;Otherwise print message

PRS \ DB "Programming failed at ",0
SCAL TBOD3
PRS \ DB NL,0
LD B,OFFH
;Set failed-flag
;If ESCAPE pressed then quit
SCAL INP
JR NC,NEXT2
CP ESC
JR Z,ESCAP2
;Otherwise do next byte
NEXT2 INC HL
INC IX
INC DE
LD A,D
OR E
JR NZ,LOOPZ
;If not finished
;If any failures go back to program again
ESCAP2 LD A,OFFH
CP B
JR NZ,PROGOK

```

```

A1D2 202D2054
A1D6 72792061
A1DA 6761696E
A1DE 3F2000
A1E1 DF7B
A1E3 FE59
A1E5 CAE3A0
A1E8 EF0D00
A1EB 1864

A1ED EF50726F
A1F1 6772616D
A1F5 6D696E67
A1F9 20737563
A1FD 63657373
A201 66756C0D
A205 0D
A206 4C617374
A20A 206C6F63
A20E 6174696F
A212 6E2000
A215 2B
A216 DF66
A218 EF0726F
A21C 6772616D
A220 6D656420
A224 77697468
A228 2000
A22A D02B
A22C DD7E00
A22F DF68
A231 EF0D0D00
A235 EF416E6F
A239 74686572
A23D 20455052
A241 4F4D203F
A245 2000
A247 DF7B
A249 FE59
A24B C29C9C
A24E C3119F
A251 EF0D5472
A255 7920616E
A259 6F746865
A25D 72204550
A261 524F4D3F
A265 2000
A267 DF7B
A269 FE59
A26B C29C9C
A26E C3B0A0

PRGOK PRS \ DB "Programming successful",NL,NL
DEC HL
SCAL TB0D3

DB "Last location ",0
TRYAG7 LD HL,LINE2+64*2+19
LD (CURSOR),HL
SCAL INLIN
LD HL,19
ADD HL,DE
EX DE,HL
SCAL NUM
JR C,TRYAG7
LD A,(NUMN)
CP 4
JR NC,TRYAG7
;Convert ROM type to ROM size in bytes
;Get "BCD" value
;Put in workspace
;Zero byte count
;First call is a dummy
;Hundreds
;Tens
;Units
;Divide by 2
;Acc now has EPROM size DIV 256
;Check that this size is allowed
LD BC,5
CPIR
JR NZ,TRYAG7
;If OK, load value into ROMSIZ
LD H,A
LD L,0
LD (ROMSIZ),HL
RET
;BCD to binary conversion
;Shift next most significant digit
;into reg D
BCDBIN LD D,0

A27D EDB0
A27F C9
;Get EPROM type

A280 EF547970
A284 65206F66
A288 20455052
A28C 4F4D203F
A290 20203237
A294 00
;Print default value
LD HL,(ROMTYP)
LD A,H
CALL PRLSB
LD A,L
RRA
RRA
RRA
RRA
RRA
RRA
CALL PRLSB
LD A,L
CALL PRLSB
;Read in decimal value
TRYAG7 LD HL,LINE2+64*2+19
LD (CURSOR),HL
SCAL INLIN
LD HL,19
ADD HL,DE
EX DE,HL
SCAL NUM
JR C,TRYAG7
LD A,(NUMN)
CP 4
JR NC,TRYAG7
;Convert ROM type to ROM size in bytes
;Get "BCD" value
;Put in workspace
;Zero byte count
;First call is a dummy
;Hundreds
;Tens
;Units
;Divide by 2
;Acc now has EPROM size DIV 256
;Check that this size is allowed
LD BC,5
CPIR
JR NZ,TRYAG7
;If OK, load value into ROMSIZ
LD H,A
LD L,0
LD (ROMSIZ),HL
RET
;BCD to binary conversion
;Shift next most significant digit
;into reg D
BCDBIN LD D,0

A2ED 1600
BCDBIN LD D,0

```

```

A2EF 0604      LD B,4
A2F1 CB25      LOOPB
A2F3 CB14      SLA L
A2F5 CB12      RL H
A2F7 10F8      RL D
                DJNZ LOOPB
                ;If it is zero then return
                PUSH AF
                LD A,0
                CP D
                JR Z,SKIPPY
                ;Add 'D' decimal weightings to byte count
                POP AF
                LD B,D
                ADD C
                DJNZ LOOPW
                PUSH AF
                SKIPPY POP AF
                RET
                ;Return cursor (move back 5 spaces)
                RETCUR LD HL,(CURSOR)
                DEC HL \ DEC HL \ DEC HL \ DEC HL \ DEC HL
                LD (CURSOR),HL
                RET
                ;Print LSB of accumulator
                ;suppress leading zeroes
                PRLSB AND 0FH
                RET Z
                SCAL BIHEX
                RET
                ;Read in byte from EPROM
                RDBYTE OUT (C),H
                INC C
                OUT (C),L
                DEC C
                ;Set OE low, keep PGM low (inactive)
                ;
                ;
                ;
                LD A,100B
                OUT (CONT D),A
                ;Read in data byte
                IN A,(DATA_D)
                ;
                ;Set OE high, keep PGM low (inactive)
                ;
                ;
                ;
                PUSH AF
                LD A,101B
                OUT (CONT D),A
                POP AF
                RET
                ;Delete line
                DELETE LD HL,(CURSOR)
                SCAL CPOS
                LD (CURSOR),HL
                LD A,ESC
                ROUT
                RET
                ;HL points to LHS of line
                ;Send ESCAPE
                RET

A338 00D8      RAWTAB DW 0D800H
                RAMSTA DS 2
                RAMEND DS 2
                RAMLEN DS 2
                ROMSIZ DS 2
                ROMBAS DW 0
                ROMSTA DS 2
                ROMEND DS 2
                ROMTYP DW 16H
                ;Tabulate address
                ;Start of RAM block
                ;End of RAM block
                ;Length of RAM block
                ;Size of EPROM in bytes
                ;Base address of EPROM
                ;Start address for programming
                ;End address
                ;Type of EPROM (default 2716)

A342 0000
A348 1600
                ;Allowable EPROM sizes (DIV 256)
                A34A 08102040
                A34E 80
                EPRTAB DB 8,10H,20H,40H,80H
                ;Title
                A34F 4550524F
                A353 4D205052
                A357 4F475241
                A35B 4D4D4552 TITLE DB "EPROM PROGRAMMER"
                END
                Workarea - 95D3 to 9B14
                ORG end - A35F
                LOAD end - 9C8C

```

MORE DAVE HUNT BITS

80-BUS News

So the July-August issue has at last emerged (at the end of January?). Oh well, only six months late, but then I haven't been exactly productive either. All sorts of things have been getting in the way; work, family, earning money, and many others things even more sordid. In the end most of the issue was a solo production by Paul, so I can hardly complain about the irregularity of 80-BUS. Anyway, I've just finished reading it (most of which I hadn't seen before publication) and feel a few comments are in order. (Is this to be a review of the reviews (of the reviews.....), etc, no way.) On the subject of irregularity of 80-BUS, Dr. Coker suggests an increase in the editorial staff, a reasonable idea, but whoever is interested should live reasonably close to North West London, obviously know something of what they are talking about and have plenty of free time. A quick scan through the 80-BUS distribution database reveals about 200 to 300 names in the area, if they feel they qualify in the other respects, perhaps they'd like to give me a buzz.

Nascom

One predominant feature of the letters is the treatment that some contributors feel they have received from Nascom, particularly regarding their helpfulness (or lack of). I feel the letters as printed represent a fair balance, as being in a position where I see a number of Nascom owners, many report the same sort of things. The latest issuing from Nascom has been a new trade/retail price list, dated November 1984. Now this is the first price list that I have seen since May 1982 (shows how well they keep in touch with their dealers) and the joke is the 10% increase in the retail price of most of the hardware, and the drastic reduction in dealer margins for those who don't buy enough these days. The nett affect of this is that Henry's Radio at least will no longer stock Nascom products. A pity, as many bits of Nascom gear are still viable and the new things like Lotti are superb. But it seems the Lucas is no longer interested in the home user, but rather the dedicated system user working for a company which can afford to pay for the complete system in one go.

Gemini

Mind you, Gemini could not be accused of pandering to the home user either, as their prices are usually worse on the wallet than Nascom. But at least they continue to supply the home user and allow enough margin in their products to allow their dealers to support their smaller customers.

Cheap Video

A thought for some enterprising person not too keen on paying out for the Gemini video card. Have a serious look at the 80 x 25 card manufactured as an add-on to the Einstein, it costs 50.00 (including VAT) and looks very like a cut price version of the MAP VFC. Is got on board RAM, a 6845 video processor and all the I/O to fit the Einstein 'PIPE' which looks just like a buffered Z80 bus (and what is 80-BUS if it's not a buffered Z80 bus?). Mind you the superficial simliarities between the Einstein and Gemini/Nascom product don't stop there. The software 'feels' very familiar, and anyone used to using NasSys wouldn't need the Einstein software manual for the machine code operating system! Do I qualify for my personally signed Lawrence picture for suggesting that one and thereby saving our hero some loot?

80-BUS Independence

There seem to be two contrary views being expressed at present (no, expressed is the wrong word, it's only an impression I get) with regard to the editorial independence of 80-BUS. Some of these impressions have been aired in the letters columns, some I've gleaned as a result of the survey. The first view is that 80-BUS is becoming the 'official' Gemini mouthpiece, and from this I get the feeling that this is mildly disapproved of. The second is that as 80-BUS IS the Gemini mouthpiece, why don't Gemini take a firmer hand in the organisation and running of the magazine, particularly with regard to getting it out on time. Whilst 80-BUS indeed owes a debt of gratitude to Gemini for use of office facilities and to a small degree, for subsidizing the magazine by making use of the distribution list to mail price lists, etc; no demands are made on 80-BUS to reciprocate with any sort of 'plug Gemini' deal. Personally I do not like the periodicals related to some other computer manufacturers which spend their time fawning over their favourite toy to the exclusion of almost everything else. I think a reasonable balance now exists, where, if I have something rude to say about Gemini (or anything else for that matter), then I can say it without much fear of the offending piece having the blue pencil put through it provided it isn't likely to land us all in court.

The Survey

I understand that the survey went well, I've read some of the replies and these in the main are productive and interesting. How to improve the magazine from the results of the survey is more of a problem; certainly one statistic is that the large majority of people who responded have various sorts of disk systems. As the response has been a bit less than 30% of the distribution list, does this mean that the response is representative and nearly all readers have disk systems, or perhaps it means only those with disk systems reply to surveys. I don't know, and it will be a problem to discover how representative the survey has been. None the less the survey has been revealing both in the equipment used by the readers and also comments about 80-BUS in general.

IMP Ribbons

A quick commercial: Mr. Cooper mentions refilling Imp ribbons with Epson refills, nice one. For those who break the cartridge trying to get the top off, Henry's have new Imp ribbons (at a price).

BBC BASIC

Oh dear, I knew I would be skating on thin ice having a go at the BBC computer and BASIC and the way they are used in education. I have had several letters (not for publication), variously addressed to me at home, work and through 80-BUS which generally endorse my view that all is not well when it comes to computer education in junior schools. A number of people I have met have also mentioned the subject with similar views to my own.

As to my comments about the BBC BASIC, and the defence of by Mr. Hellen. Yes, I feel my comments were somewhat scathing, although I feel my remarks to a degree justified by Mr. Hellen's own comment about the BASIC, in that he, "Found it easy to use, powerful and challenging." The operative word I feel is, "Challenging". At the age that young people are introduced to the BBC BASIC, a challenge is perhaps not the best thing they need. Rather it is the need to be able to grasp quickly and soundly what the language in the computer is capable of doing. BASIC is recognised by those who are supposed to know, as being a bad first language for young people, or at least, so I'm told. I am in no position to judge, as I've never tried any that are recommended, (LOGO, et

al). Nor for that matter am I child psychologist, or at least no more of a child psychologist than the average father of three daughters who wishes to retain some self-respect in his own household needs to be. Perhaps the matter of the suitability from an educational point of view of this particular version of BASIC ought to be left to those far more qualified to judge. I suspect they'll still be thrashing that one around even when we achieve biological computers which don't require languages.

As to the BBC language itself. Yes I am well aware of the Z80 CP/M version of it, having had an early copy for about eighteen months. I agree entirely with Mr. Hellens quoted comment (above), and particularly like the 10 digit maths precision for normal use. It is also comparatively cheap. The onboard assembler (Z80 mnemonics in the CP/M version) gets round the sort of problems encountered with video cards as discussed in my last piece, very neatly. However, I still have reservations about the way in which it uses procedures as well as (and/or) subroutines. This sort of structure is very 'unBASIC' and far more reminiscent of Pascal and 'C' hence my renaming the language. This is not intended to be derogatory, simply an expression of the fact that I feel the language to be a hybrid calling upon the better bits of others and therefore justifying a new name. After all, Comal has been recognised as a separate language, and yet is not as radical as BBC BASIC.

As a language, I find it easy to program (when thought of as simply BASIC), and a little bit harder when some of the clever frills are used, which in turn promote the use of structures in programming which are both compact and easy to read. Its most commendable feature is that it is very fast indeed. At some point I must persuade Steve Hanselmann to publish his BBC tape reading and writing programs which he has written to transfer stuff to and from a BEEB, as the BEEB at least uses the Kansas City (CUTS) tape interface as do the Gemini and Nascom 2. BBC BASIC is available in Gemini and Nascom CP/M formats from OFF RECORDS of Battersea.

Not to me at work!!

On the subject of letters addressed to me, please send them to 80-BUS or my home address. Please DO NOT send them to me at work, I get embarrassed when I have to explain to the boss why my personal mail is larger than his!!

EPRM Eraser

Paddy Coker's article on erasing EPRMs was very good, but I note a couple of variations from a chat I had with a guy from TI a long time ago, and that I seem to remember were published. Firstly keeping EPRMs cool during erasure. Now this fella told me that EPRMs slow up as they are repeatedly erased and reprogrammed, and that the trick here was to cook them for about 20 minutes at 100 deg C, this brought them back to speed. There was a time when we used an 80 watt quartz sun ray lamp tube (you know the type, those cheap heat & light type sun ray lamps with the heater elements removed and replace by a 80 watt fluorescent light choke) at about an inch from the EPRMs. They got darned hot, but didn't seem to suffer any fatalities (either short or long term), so I don't know. These days we use an 8 watt 12" fluorescent tube specially made for EPRM erasure. This keeps the chips nice and cool, and I go along with that, but Paddy is right about TI chips, they take ages. Perhaps TI ones need the heat?

One other point, safety. I must re-emphasize Paddy's remarks. UV light is dangerous!!! It is carcinogenic, that is, it is known to be a contributory cause of cancers. I have a friend who has recently undergone operations for cataracts in both eyes and the attributable cause is UV light. He used to work for a PC production company exposing the resist coated boards. He was in the

habit of defeating the interlock switch on the machine as he liked to watch the resist change colour, he says he could judge the exposure far more accurately than by using a timer. Ok, but at the expense of ruining your eyesight? He wasn't in the job long, and the dosage was not much greater than that needed to erase a few hundred EPROMs, but it was enough.

CP/M Users' Group

So it seems the CP/M Users' Group hasn't seen enough coverage in this magazine. I can't think why it hasn't been plugged before, perhaps it was assumed that everyone knew about it. Anyway, I'll mention it again. A very creditable organisation for CP/M users. As was mentioned by Dr Plews, there are about 250 volumes in the library and all of it costing next to nothing. There's even some bits of mine in there, perhaps that's why they give it away. Be that as it may, to get volumes from the library you send a disk and £2.00 to cover the copy charge and return postage. Mind you, you've got to be a member first, that costs £7.50 per annum for an individual and you get a quarterly magazine for that. It must be remembered that the volumes refer to 8" disks with about 250K each, so if you're using a Henelec/Gemini G805 system then you'll require two disks for each volume.

Over the years I've acquired a fair number of volumes, and should add that about 50% of the stuff I have is pretty useless, or to be more charitable, very old and written for EBASIC. None the less the remainder is marvelous stuff, all very useful and relevant. About half of the 'Henry's Incredible Utilities Disk' is extracted from various volumes of the User Library. Unfortunately the library is not over endowed with 'heavy weight' programs, so large database controllers, high level languages, assemblers, etc, may still have to be purchased when required. But as for the smaller utilities, well you can't go far wrong by joining and getting the library index volumes and seeing what is available. Contact: CP/M Users' Group (UK), 72, Mill Road, Hawley, Dartford, Kent. DA2 7RZ. Phone (0322) 22669

MODEMS

Mr. Brown mentioned UKM715, the UK version of the CP/M Library Program MODEM7, and that Henry's have a copy. True. They don't even charge for it (they are not allowed to), but they'll charge you for a new disk to put it on even if you bring along your own disk! Which brings me round to the subject of bulletin boards. These can be great fun but not all use the same protocols. Many are Prestel like, other simply assume a dumb terminal at the other end. Some use 1200/75 BAUD others use 300/300 BAUD. A few will accept either, and one uses 1200/1200 BAUD. Likewise, the range of modems available at present is confusing in the range of options offered. So what ever modem and software you may have, it is unlikely that you will be able to access all of them.

In the short term at least three suites of software will be required, a clever one like UKM715, a dumb one like Richard Beal's TERMB and a 'Prestel like' one such as Dave Ryder's PRETZEL2. Richard Beal is currently beavering away on a new program called DIAL (which will probably be finished when this lot appears in print). This is neat because it will contain a library of telephone numbers, access codes (yours not his) and a list of the appropriate software to run each bulletin board. Basically you select the service you require, DIAL then loads up the appropriate software, dials the service for you, sends any access codes required and then jumps into the terminal software. Hopefully, DIAL will be patchable for the more usual modems, and it may be possible to wrap this lot up as a package for sale at a reasonable price. As to Gemini or one of the dealers setting up a bulletin board, all I can say at present is, it's possible!

A 'C' PROGRAMMER EXPOSES HIMSELF

By Dave Russ

Never once flinching from that hard cruel stare, the lovely mantissa threw off her flimsy wrap and pressed her heaving bosom into the dark matting that was his chest... What was that Paul?... 80-BUS still 4 months behind... an article on 'C'?... But I was just!!... Oh alright then.

Well since my debut article on GSX I have been harbouring the thought that my as yet untapped talent for generating copious blurb could be put to better use in writing popular smut, however the frontiers of science will never be pushed much further if I adopt that attitude so I had better move on to more serious matters.

Have you ever heard the conversation that went like, "'C', yes I've had a look at it but could never get the hang of it somehow, I just stick to BASIC and assembler now". I used to hear this quite often, and indeed have had to suffer the guffaws of non 'C' afficianados when confronted with a seemingly incomprehensible source listing. It did worry me at one time, and I was considering having a complex about the fact that I actually liked using 'C'. Now, however, my confidence is restored as I have heard a buzz that the mighty Digital Research is writing BIOS material in 'C', and it is whispered in the hushed cloisters of Gemini Microcomputers that your own favourite agony aunt David Parkinson uses it. So, in such esteemed company, I will finally come out of the closet and expose myself in public once and for all.

Really folks once you get used to the language it seems like all your prayers have been answered; in essence 'C' is a high level assembler which also allows you to structure your programs according to the high ideals of the computer academics. It was originally derived from BCPL I'm told but never having seen that I don't know, however, I suppose that the best comparison I can make is to Pascal, but with a lot less typing involved. Concerning the pedigree of this language I should say that 'C' and UNIX go hand in hand, more so in the past as a large proportion of the UNIX operating system is written in 'C' with only the nitty gritty way down there in the UNIX kernel being directly coded in assembler. Considering present company though I will drop the subject of UNIX as several 'C' compilers are available for the CP/M enthusiast.

I have in the past programmed in BASIC, COBOL and FORTRAN and I must say that in terms of program structure and flexibility 'C' beats these hands down for the type of applications that I have been concerned with. Of course there is no getting round the fact that for certain parts of programs assembler code is just going to have to be used for speed, but this is no problem as 'C' will allow you to either directly include assembler mnemonics in a source listing or link in external modules using L80 or the suchlike, depending on your compiler. This allows you to create the controlling software without a dollar terminated string or BDOS call in sight and when necessary call the assembler section to do its stuff and even pass parameters to it if required.

Now let me say at this stage that I bypassed Pascal for 'C' and thus I'm not very familiar with it at all, so all you Pascal freaks can sit down now and stop shouting 'But I can do that', I would never deride Pascal as I would be using it myself now had I not taken to 'C'.

I do not intend to give a detailed description of the language here as it would be too long and boring but if I can try to conjure up the flavour of "The `C` programming environment" as the yanks call it, I think that would serve my purpose better. If by some miracle these documented mumblings generate some enthusiasm then I recommend that you go out and buy the standard text book for any `C` programmer, that is "The `C` programming language" by Brian Kernighan and Dennis Ritchie published by Prentice-Hall. It is quite good for a computing textbook, and what is more, defines the standard to which all compiler writers strive, and all available compilers are compared. Better still borrow it from a friend as it probably costs about £19 by now.

Ok, enough of the preamble, lets get our hands dirty and walz straight into a `C` program. This will not shock the computing world with its complexity but is a program to open a disk file and write the value of a loop counter into it.

```

#include "stdio.h"                                /* include standard IO header file
                                                    from disk */

int count;                                        /* declare a 16 bit integervbl */

FILE *fd;                                         /* declare file descriptor */

main()
{
    printf("A `C` prográm\n");                    /* print opening message */
    openit();                                     /* call function to open file */
    wrt_nums();                                  /* call function to write to file*/
    printf("Program terminating\n");             /* print end message */
    fclose(fd);                                  /* write buffers & close file */
    exit(0);                                     /* call exit routine */
}

openit()                                         /* function to open trivia file */
{
    fd=fopen("A:TRIVIA.DAT","w");                /* pass name & mode to fopen func*/
    printf("Trivia file: ");                     /* display file id */
    if(fd == NULL)                               /* is the file open sucessful */
    {
        printf("open error");                    /* No - so print error message */
        exit(0);                                 /* ... and terminate program */
    }
    else printf("open ok\n");                     /* file opened ok, so tell em */
}

wrt_nums()                                     /* writes value of count to disk */
{
    for(count=0; count<=100; count++) putw(count, fd);
}

```

So having looked at it what do you think, I know that I have been using `C` for a while now and I'm pretty used to it, but is it really as mumbo jumbo-ish as some people say?

The file `STDIO.H` is supplied with a compiler and is `'included'` at the beginning of most programs that require IO capabilities. It is itself `'C'` source code and it declares items to the program such as file control blocks and IO buffers that will be used later on. It also declares things like `'NULL'` which can be seen in `openit`, this is analagous to the M80 equates that are seen at the beginning of a program. Examples from `stdio.h` are typically:

```

#define NULL 0
#define TRUE 1
#define FALSE 0

```

You can also use the `#define` facility to define equates of your own in the identical way. The `'C'` macro preprocessor that most compilers have looks through your code and replaces all occurrences of `'NULL'` with `'0'` and so on. `#define` entries can also be used for inserting macros into your code.

`'C'` data types

The next two lines declare permanent storage for the variables `'count'` and `'fd'`, don't worry about the `'*'` in front of the `'fd'` it means that it is an integer pointer, more on that later. These variables defined in this way outside of any functions declare then as global to all functions in this source file and also to any modules that are to be linked in later. Being globals they will reside in memory and maintain their set values throughout the run of the program. They may be accessed and altered by any of the program functions.

`'C'` also allows you to use local or, to give them their proper name, automatic variables, these are declared inside the body of a function and reside in the stack area until a return from the function occurs. Automatic variables are declared as follows.

```

functionx()
{
    int var_a, var_b;
    char array[16];
}

```

The above declares two 16 bit integers and an array of 16 bytes in the stack area. So you can see that you are able to create space for storage and arrays that you can process inside a function, store a result and discard on exit. Once the function has done its stuff the stack pointer is restored to its previous value before the routine was called and the stack space made available for other uses.

The data types available to you sometimes depend upon the compiler that you have but a typical `'C'` compiler will have the following types.

short or char	- These usually occupy a single byte.
int	- Integer data type, usually two bytes long.
long	- Long integer, usually four bytes long.
float	- Single precision FP variable usually 4 bytes long.
double	- Double precision FP usually 8 bytes long.

Mixed mode arithmetic between these data types is made easy for you as the compiler should sort your assignments out for you. I have in the passed assigned the result of a calculation which was held in a double precision FP variable directly into an integer simply by entering:

```
int_var = double_var;
```

Of course the compiler uses library functions to achieve this transformation which are also available to you for use if required, but it is nice to think that the compiler is intelligent enough to sort this out for you.

Pointer variables

'C' is also very adept at handling pointers and doing arithmetic with them. You declare pointer variables using the syntax, `char *ch_ptr` or `int *int_ptr`, where `ch_ptr` and `int_ptr` are variable names. These variables will be used as pointers to character or integer fields and are typically used for processing arrays. An example of how pointers may be put to good use is the alteration of the IO byte under program control. We know that the iobyte lives at 0003h so here is how we get at it:

```
char oldiob;                /* storage for old iobyte value */
char *iob=3;                /* pointer to 0003h - the iobyte */

setiob(value)               /* changes iobyte to 'value' */
char value;
{

oldiob = *iob;              /* store contents of 0003h in oldiob */
*iob = value;               /* set 0003h to new value passed as arg */
}
```

The character pointer `'iob'` is declared as a pointer to 0003h it will be a 16 bit field of course and contains the value 3. If during processing an asterisk is placed in front of `iob` as in the above example, the contents of the byte that `iob` is addressing is used.

Should we now wish to do something with the contents of 0004h all we have to do is increment `'iob'` using `'iob = iob + 1;'` or more simply `'iob++;'`, `iob` now points to 0004h.

Pointers can also be used for integer fields, and interestingly, if an integer pointer is incremented as above, it points to the start of the next integer ie. address+2, the compiler has also sorted this out for you.

As you may be unsure as to where in memory your data will reside in its final .COM version 'C' provides you with a mechanism whereby you can locate the address of a variable or data structure. This is normally used to initialise the values of pointers. Take the example:

```
functiony()
{
char arrayx[100];
char *point;

point = &arrayx[0];

..... and so on
```

Here we have declared a hundred byte array (0 - 99) and set up a character pointer to the start of it. `&arrayx[0]` is the syntax used to return the address of `array[0]` which is then placed into the pointer `point`. For `C` fans everywhere that have noticed that I could have achieved the same by saying `point = arrayx;`, fear not, I realise this but it is a good example of `C`'s addressing capability.

Pointers to functions are also available and are useful in applications that require overlay sections. Executable code is read into an area of memory with a known base address and using function pointers control is passed to it. You are also capable of passing arguments to this function if required.

`C` functions

`main()` is the name given to the controlling function of any program and always receives control when the program is entered. Of course you can do what you like as soon as you have entered `main`, but devotees of structured programming will realise that this is where the controlling calls of the overall program should reside.

In `main` of the first sample program you can see that 4 functions have been called, namely `openit`, `wrt_nums`, `printf` and `exit`. You can see where the first two come from, they are in the source listing, but what about the others? Well it may come as a bit of a shock but `C` contains no IO capability of its own, there are no intrinsic functions such as in BASIC or FORTRAN. What this means is that if you compiled this program as it stands into a `.COM` file the linker would declare `printf`, `exit`, `fopen`, `fclose`, and `putw` as undefined. Pretty useless on its own eh? But this is where `C` wins over its rivals in some respects. The functions such as `printf`, which is capable of displaying a formatted string on the screen, have to be loaded from function libraries supplied to you in REL format on your master disk. The required code is extracted from these libraries by using the `L80 /s` option for example and linked into your program. This means that the runnable program contains only the code necessary and no unused portions.

These function libraries are usually pretty extensive and contain not only the routines defined by the `C` standard but also some which have been added by the compiler writers to make your life easier. You will have guessed from this that you are also able to create your own libraries and place them into your programs in a similar fashion. I have in the course of my duties at Gemini created function libraries for both the SVC and Pluto which are written in assembler.

As a point of interest the `C` standard dictates that arguments passed to a function are pushed onto the stack in reverse order before the function is called. The value of the argument is pushed and not the address of the variable ie. pass by value. So from within your custom written assembler routine you can retrieve these values from the stack and send them out to your SVC or Pluto as data using the Z80 OUT opcode. In practice though, I have noticed that the ECO C compiler only passes integer and character arguments by value and any arguments longer than 2 bytes are passed by address. There are often idiosyncracies involved with the method by which arguments are passed so I'm afraid that there is no substitute for reading the manual.

Control mechanisms

All the usual controlling mechanisms are available to you these are:

```
if<condition> ... else...
```

```
for(<loop start> ; <loop end condition> ; <loop step> )...
```

```
do ... while<condition>
```

```
while<condition>...
```

and the very handy `switch` multiple condition mechanism.

if.. else and do.. while require no explanation but I would just like to run through the for loops and switch statements.

The for loop in the example program went like this:

```
for(count=0; count <= 100; count++) putw(count, fd);
```

This translates as set count to 0, WHILE count is less than or equal to 100 call the putw function to output the value of count to the disk file, and at the end of each iteration increment the value of count by 1 (count++).

This is the normal format of any for loop but I don't see any reason why you could not put fred<=100 instead of count<=100 to make the end condition dependant upon some other variable and give you added flexibility.

The switch statement is a pearl and saves you the agony of entering multiple if conditions. It is ideal for menu driven applications. A typical example may be:

```
switch(value)                /* value is a variable */
{
    case 1:
        function1();
        break;

    case 2:
        function2();
        break;

    case 3:
        function3();
        break;

    default:
        printf("ERROR - illegal option");
        errflag = TRUE;
}
```

Tests are done on `value` and if it is found to be 1, 2, or 3 then the appropriate function is called and on return the case structure is exited via the `break` statement. If none of the valid cases are found to be true the

default routine is executed, in this case the output of an error message and the setting of an error flag.

Now we have introduced the ``break`` statement, I think that we should give it a further airing along with its bedmate the ``continue`` statement. Although ``C`` allows the use of the ``goto`` statement, it should never be necessary to use it. Let us say that we are in the body of an executing loop and we come across the following:

```
if( vbl_1 == 0 ) break;
```

`vbl_1` is tested for a zero value and if it is found to be true the loop is exited and control is passed to the statement immediately following the loop statements. A nice clean way don't you think to conditionally exit an executing loop.

if the following was encountered:

```
if( vbl_1 == 0 ) continue;
```

``C`` would interpret this as, if `vbl_1` is found to be 0 then forget about the remainder of the looping statements and continue with the next iteration. This is not an exit from the loop but the forcing of a premature ``NEXT``, to give a comparison in BASIC.

It did occur to me a while back that if you used ``#define goto structured_repass`` the preprocessor would replace ``structured_repass`` with `goto`, which means that you could have as many `goto`'s in your code as you wanted and no one would be any the wiser, especially if you put it in `STDIO.H` where it could not be found in a source listing. If you want to get your own back, using methods like that, ``C`` could be the ideal medium for ``drop through`` spaghetti programmers!

Arithmetic operators

All the normal operators are supported and usually conform to the standard rules of operator precedence. As you may have already noticed there are two ways that arithmetic syntax can be entered. I call these ``longhand`` and ``shorthand`` for convenience. Here is some equivalent syntax.

longhand	shorthand	desc
<code>x=x+1</code>	<code>x++</code>	increments x by 1
<code>x=x+3</code>	<code>x+=3</code>	increments x by 3
<code>x=x*3</code>	<code>x*=3</code>	multiplies x by 3

In addition to these you will also have bitwise logical operators at your beck and call, some examples below.

<code>x>>=4</code>	shifts x four bits to the right
<code>x<<=6</code>	shifts x six bits to the left

<code>x&=value</code>	logically ANDs x with value and stores back to x
<code>x =value</code>	" ORs " " " " " " " "
<code>x^=value</code>	" XORs " " " " " " " "
<code>x=~x</code>	stores the ones complement of x to x

It is probably this sort of thing that causes so much confusion to the non initiated as experienced 'C' programmers prefer to use the shorthand as it is easier to type in, but doesn't do much for the clarity of the source code.

The 'C' programming environment

As you can see it can be used at many levels depending on the application in hand. If it is tight assembler like code that you require, you can develop a whole program using bit fields and logical operators, never once calling on one of the library functions. If however you wish it to be a higher level language all the facilities are there for you to use.

You may find that some of the supplied functions are not to your liking. For example the inclusion of the standard printf function from a library tends to add 4k onto the program even if it was only a simple message. If this does not suit you then all you have to do is write your own with less formatting capability and so less memory required.

This ability to create libraries of functions that can easily be linked in the final version adds to your programming capability. As you develop you will acquire libraries that suit your applications and programming style and perhaps discard the standard libraries altogether. As mentioned I have created a library of Pluto functions and with these I have in effect a high level Pluto programming language. If you were to look at a Pluto program you would barely recognise it as 'C' at all, as it consists of calls to my custom written routines. This must be the biggest advantage in that you mould the language to suit yourself.

Anyway, there it is, I was hoping to include a section on compilers, what is available and how they work, but I think that I will save that for the next issue (If I ever get invited to write again), besides I must get back to my novelette or I'll never get that house in Palm Beach California.

Right.. Where was I... 'She sighed deeply. The entangled mesh of arms and legs that was their lovemaking.....'

What the general public have said about 'Lawrence the Long Haired Weirdo'

I laughed until I stopped - Rauf Denktash

I wish I could have lived to see it - R.K. Maudling

We smirked a lot - Queen Elizabeth II

All my idea - Richard Nixon Not his idea - J. Magruder

You cannot close an article on economic grounds - Arthur Scargill

If you think that you are getting paid for this, you're wrong! - Ed.

Sinclair speaks to NASCOM – A snippet on interfacing

By Michael Hendry

Stimulated by Dr Dark's suggestion in the March-April 1984 issue (Vol 3 Issue 2) I decided to rake out this program, first written in January, and submit it for publication.

The Spectrum is a remarkable little machine, but most of the interesting games available are in machine code, and difficult to patch (i.e. cheat!) without decent tools. I was too mean to buy a machine code package and a printer interface for my son's Spectrum, having perfectly good facilities on the NASCOM, and I decided to interface the Spectrum to my NASCOM 2 without using any extra hardware so that I could disassemble, list, and generally tinker with code dumped from the Spectrum.

The Spectrum uses one of its ports (OFEH) for output to the MIC socket and to provide the Border colours on the screen, bit 3 for the MIC and bits 0-2 for the three primary colours. Its built-in tape routines use a non-standard format, but it is possible to use the output bit provided in any format you wish, in this case the CUTS standard used in the NASCOM, which outputs 1 cycle at 1200 Hz for a zero bit, and 2 cycles at 2400 Hz for a non-zero bit when it is set up for 1200 Baud output. I used 300 Baud values (4 cycles and 8 cycles) initially, but found that 1200 Baud was more reliable!

The crucial routine in the program is labelled SndBit; each bit to be output to tape is passed to this routine in the carry flag, and the MIC bit in port FE is toggled accordingly, with appropriate delays for the frequencies involved. The initial counts for the delay loops have to be calculated from the known clock frequency of the Spectrum and the number of T-states used by each instruction, and are included in the EQUates at the beginning of the listing.

SndBit may be regarded as a software emulation of the NASCOM's cassette interface, and is sent data byte-wise by the UART emulator routine BytOut, which is in turn fed by the NAS-SYS "Write" emulator routine, which is the bulk of the remainder of the program. The analogy is not complete, in that the NASCOM hardware is continuously sending out a stop bit when the cassette output is not in use. The loop labelled PLO is used to provide a short burst of this "pilot tone".

For convenience, the start and end+1 addresses are passed to the program at 8000 and 8002 (ie 32768 and 32770) and the program itself runs at 8004 (RANDOMIZE USR 32772). I use a second copy assembled at F000 if code in the range 8000 to 80C0 is of interest. You may wish to write a short BASIC program to prompt for the necessary addresses and remind you to start the tape, but remember that this may encroach on code under examination.

The output follows exactly the format of a "W XXXX YYYY" command under NAS-SYS with all the usual checksums, and may be saved on cassette and then loaded directly into memory on the NASCOM using the "R" command. If NAS-SYS 3 is available, it may be loaded at any address by giving the appropriate offset as an argument. With suitable amplification it may be connected directly into the tape input on the NASCOM.

IMPORTANT NOTE: This program will not operate properly at addresses below 8000 because the Video circuitry is constantly reading this block of RAM, and upsets the timing loops. As the Spectrum keyboard is read under interrupt, this must also be disabled while the program is run (and enabled afterwards of course!).

I have also written code using similar techniques which uses the Spectrum LLIST and LPRINT commands to speak to the NASCOM through the cassette interface, using the NASCOM as a printer buffer to my Epson. It doesn't handle screen dumps yet, but one of these wet Sundays...


```

8085 CD9080      CALL SndBit
8088 C1          POP BC
8089 10F7        DJNZ B01
808B 37          SCF ;send 2 stop bits
808C CD9080      CALL SndBit
808F 37          SCF
808F            ;Output the bit in the carry register
808F            SndBit JR C HiBit ;jump if logic 1
8090 3817        LD B 1 ;else 1 cycle at 1200Hz
8092 0601        PUSH BC
8094 C5          LD A 1
8095 3E01        OUT (OFEH) A ;set MIC bit, blue border
8097 D3FE        LD B DEL1
8099 066F        DJNZ SB1
809B 10FE        LD A OEH
809D 3E0E        OUT (OFEH) A ;reset MIC, yellow border
809F D3FE        LD B DEL2
80A1 066C        DJNZ SB2
80A3 10FE        POP BC
80A5 C1          DJNZ SB0
80A6 10EC        RET
80A8 C9          LD B 2 ;2 cycles at 2400Hz
80A9 0602        HiBit LD B 2 ;2 cycles at 2400Hz
80AB C5          HB0 PUSH BC
80AC 3E01        LD A 1
80AE D3FE        OUT (OFEH) A
80B0 0637        LD B DEL3
80B2 10FE        DJNZ HB1
80B4 3E0E        LD A OEH
80B6 D3FE        OUT (OFEH) A
80B8 0634        LD B DEL4
80BA 10FE        DJNZ HB2
80BC C1          POP BC
80BD 10EC        DJNZ HB0
80BF C9          RET
80BF

```

THE QUESTIONNAIRE PRIZES

Without further ado here are the winners in the Questionnaire draw.

The ten free one year subscriptions go to:

D. Atkinson, Chesham, Bucks
 R. De Brown, Acomb, Yorks
 A.R. Green, Calverton, Notts
 M.D. Hendry, Cupar, Fife
 D.M. Lord, Bacup, Lancs

S.G. Minshull, Borrowash, Derby
 J. Turner, Pontefract, W. Yorks
 W.H. Turner, New Malden, Surrey
 J. Washington, Woking, Surrey
 A.P.R. Watt, Crawley, Sussex

And the voucher for £100 worth of Gemini goodies goes to:

C. Waller, Luton, Beds.

Congratulations to the above, and thanks to everyone else who sent in their completed form. Steve and I have spent many hours compiling the results, and these will be published in the next issue.

INTERRUPT DRIVEN PRINTER FOR GEMINI 2

By Bill Turner

Introduction

I hear so much these days about the advantages of having a 16 or even 32 bit computer that I start becoming depressed when sitting down to my humble 8 bitter. However, with so much Z80 software I have made considerable investment which I can't throw away lightly. So now the Z80 fights back. I'm not going to fit a Z80B because I don't know if the peripherals will support it and I don't know how to get the 6 MHz clock rate required.

Instead I am going to use interrupts to handle some of the devices. The first is going to be the printer which is after all attached to a PIO making the whole thing pretty simple.

The Hardware

My humble 8 bitter is a Gemini Galaxy 2 running CP/M 2.2 with an Epson FX80 connected via the PIO using the Centronics protocol. The printer is reputed to have a 2K internal buffer but this doesn't seem to make printing a smooth operation. No hardware changes are required except to the IVC as described in 80-BUS News Vol 2 Issue 3. (If the system has an SVC then no changes are required at all.)

The Theory

The new printer driving routines are built around a 256 byte buffer which with single byte pointers provides automatic wrap around making the buffer appear circular. There is an input pointer which points to the next free entry for the next character to be printed and an output pointer which points to the next character to be output to the printer. The buffer is empty when the input pointer equals the output pointer and it is full when the input pointer is one behind the output pointer (i.e. the input pointer has gone right round and caught up with the output pointer). The printer generates an interrupt when busy goes low provided that interrupts are enabled.

The Software

The full listing is somewhere else in the magazine and shows a test program which illustrates the theory. The routine labels correspond to those in the Gemini BIOS (section 3.6 of The Galaxy Computer User's Manual) where applicable and these routines are intended to replace the existing ones. INIT has to replace the existing printer initialisation and the interrupt service routine must also live somewhere in the BIOS.

Where to locate the interrupt vector is a bit of a problem. Wherever it goes defines the 256 byte page of interrupt vectors and for this reason I have put it in the BIOS scratch area in page zero but people with 64K RAM boards used as a RAM disk will find the inter-page transfer code there.

Finally, the biggest problem. Accessing the floppy disk drives is very time critical and they will lose data if interrupted during data transfers so it is necessary to find some way of disabling and re-enabling interrupts around disk accesses. I think that a couple of NOP instructions strategically placed in the disk driver routines would have been the answer, as they could be patched to DI and EI if required. This addition would also make it possible for users to attempt multi-tasking.

IVC Mods

The first thing I noticed when the code started to run was that the IVC was being reset every so often. I found the answer in Vol 2 Issue 3 of 80-BUS News and I am sure no-one will mind if I repeat it here.

First remove your IVC card. Then remove IC35 and lift up pin 14 so that it no longer engages in the socket and replace the chip. Connect pin 14 of IC35 to pin 4 of IC5 which is on the far edge of the board. Then turn the board over and connect pin 3 of IC5 to /M1, line 25 on the edge connector (not too much solder here or the card may not go back in). Now replace the card and check that it works normally.

Conclusions

It took a while to get the whole thing running as I hadn't realised that I only get one interrupt when the busy goes low. (I believe that I have read somewhere that this is due to the interrupt being generated by the high to low transition of busy).

I've been using my interrupt driven printer for over a month now without any real troubles. If you list a file to the printer, make sure that the power is turned on and that it is online or the system will hang up until it is. It should be possible to detect a permanent printer not ready and output a message but I have not yet attempted this.

After all the work I have achieved a 10% speed improvement in printing files (mainly due to there being no printer pause while reading the next sector off disk). This seems a modest improvement, but people with slower drives and/or printers should get more.

```

; Test program for interrupt driven printer
; W. H. Turner 16-OCT-84
;
; .z80
;
; PIO interrupt vector address
;
; lsvect equ 0040H
;
; Printer buffer
;
; lstin: defb 0 ;Input pointer
; lstout: defb 0 ;Output pointer
; lstbuf: defb 256 ;Printer buffer
;
; defs 6
;
; lststk:
; usersp: defs 2
;
; ; Printer port number (as in Bios)
;
; pport: defb 0B4H
;
; ; Printer PIO initialisation
;
; init: ld hl,lstin ;Clear pointers
; xor a
; ld (hl),a
; inc hl
; ld (hl),a
; ld (hl),a
; ld hl,lstvec ;Get vector address
; ld de,lstint ;ISR address
; ld a,h ;Load I reg with msb of vector
; ld i,a
; ld a,(pport) ;Get port address
; add a,02 ;PIO port A control
; ld c,a
; ld (c),l ;Load lsb of vector
; ld (hl),e ;Store msb of ISR address in vector
; inc hl ;Point to msb
; ld (hl),d ;Store lsb of ISR address in vector
; ld b,0FFH ;Set mode 3 (control)
; out (c),b
; ld d,0FDH ;Set all input except bit 1
; out (c),d
; ld a,l7H ;Interrupt control word
; out (c),a
; ld a,0FEH ;Monitor bit 0 only
; out (c),a
; inc c ;PIO port B control
; out (c),b ;Set mode 3 (control)
; xor a ;Set all output bits
; out (c),a
; dec c ;PIO port B data
; dec c

```

```

out      (c),a
dec      c
nop
out      (c),b
out      (c),d
out      (c),b
im       2
ei
reti

; List status
; on exit A = 00 if not ready
; FF if ready
;
;
centst: ld hl,lstin
ld a,(hl)
inc hl
sub hl
inc a
ret z
ld a,0FFH
ret

; List output
; on entry C = character
;
centro: call centst
or a
jr z,centro
ld hl,lstin
push hl
ld a,(hl)
inc hl
inc hl
add a,l
ld l,a
jr nc,centri
inc h
centri: ld hl,(hl),c
pop hl
inc (hl)
ld a,(pport)
add a,02
ld c,a
ld a,83H
out (c),a
ret

; Interrupt service routine
;
lstint: ld (usersp),sp
ld sp,lststk
push af
push hl
push bc

;Output null
;PIO port A data (strobe/busy)
;settle
;Strobe high
;Strobe low
;Strobe high
;Interrupt mode 2
;Enable interrupts
;Clear spurious interrupts

(c),a
c
(c),b
(c),d
(c),b
2
im
ei
reti

; If input pointer - output pointer
; is -1 (FF) then buffer is full
; (i.e. not ready)
; Otherwise return ready
;
;
; Wait for ready
;
; Get input pointer
; save input pointer address
;
; Get buffer address
;
; Get next address in buffer
;
; Store character there
; retrieve input pointer address
; increment input pointer
; Get port address
; PIO A control
; Enable PIO A interrupts
; Done
;
; save user Sp
; use own stack
; save regs
;
;

```

```

ld hl,latout
ld a,(hl)
inc hl
inc hl
add a,l
ld l,a
jr nc,intl
h
intl: ld b,(hl)
ld a,(pport)
add a,02
ld c,a
ld hl,lstin
ld a,(hl)
inc hl
sub hl
nz,int2
ld a,03H
ld (c),a
out c
int2: dec c
out c
dec c
ld a,0FDH
out (c),a
ld a,0FFH
out (c),a
bc
pop hl
pop af
sp,(usersp)
ei
reti

;
cr
lf
equ OAH
;
text: defb 'This is a test message',CR,LF
defb 'Test finished',CR,LF,0
;
; Test program
;
start: call init
ld hl,text
ld a,(hl)
inc hl
or a
ret z
ld c,a
push hl
call centro
pop hl
jr loop
;
end start

```

```

;Output pointer
;Get output pointer
;Point to next
;Get buffer address
;Get address of output character
;
;
;Get output character
;Get port address
;PIO A control
;
;Get pointer address
;If input pointer = output pointer
;then buffer is empty
;
;Not empty
;Disable PIO A interrupts
;
;PIO A data
;Output character
;PIO A data (strobe/busy)
;Strobe low
;Strobe high
;Restore regs
;
;restore user SP
;Re-enable interrupts
;Return from interrupt

```

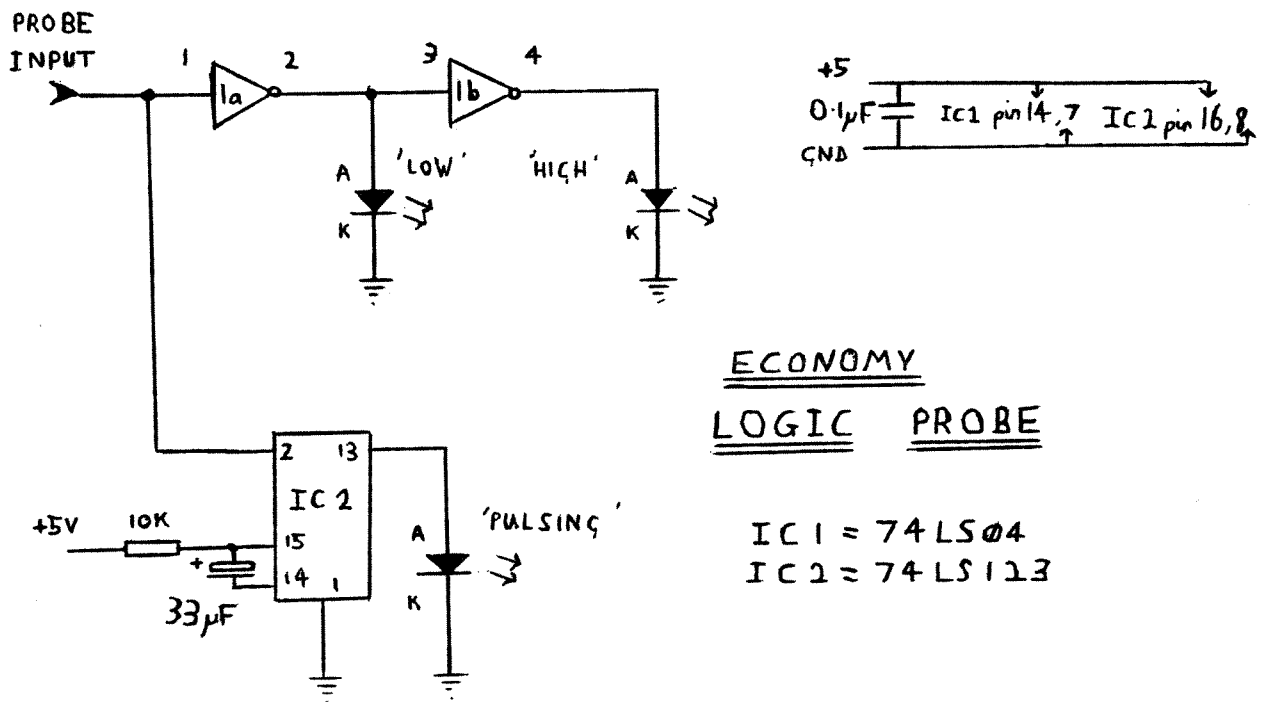
THE LONG HAIRED WEIRDO'S LOGIC PROBE

By Kevin Wood

Recently, while debugging a new board for my beloved Nascom 2, I needed to test the outputs of some chips in a manner that really required a logic probe. As the cheapest probe I could find was a £12 kit in an old Technomatic catalogue, and there was no way I would spend that kind of money if I could help it, I set about designing my own.

Now, I don't know what they mean by all the "pulse train" stuff that they go on about in adverts, but my little probe costing 70p (roughly) has seen extensive use, and done everything that I have needed it to.

The probe has a display of three LEDs, which are used to show a high logic level, a low logic level, and a pulse. A single pulse (e.g. the /RESET line) will cause the "pulsing" LED to light for about a quarter of a second. A series of pulses (e.g. the CLOCK line) will cause the LED to light brightly. This gives a far clearer indication of what's going on, than trying to judge the brightness of a dimly lit LED.

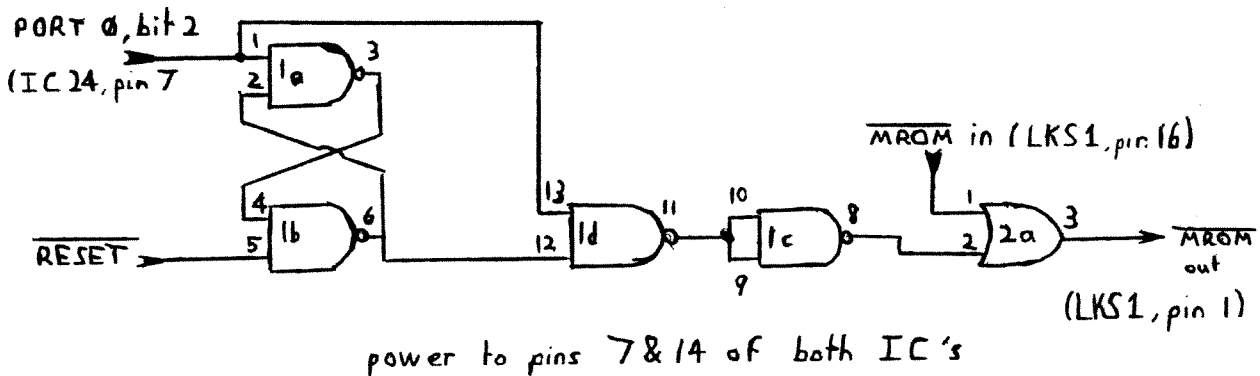


I constructed my probe on a piece of vero-board, about 25 holes by 10 holes (2.5 inches by 1 inch), as this is a reasonable size to hold in the hand. The three LEDs were mounted at one end, in a line with the "pulsing" LED in the middle. At the other end I put the actual "probe". This was simply a piece of single core wire, threaded through the vero-board a couple of times, to make it stick out in a suitably probe-like manner. Power leads are about 18 inches long, as this allows sufficient length to reach anywhere on my computer, and twisted together for convenience. Be certain to put a

decoupling capacitor of 0.1uF on the board, as it really is necessary with power leads this length. If you thread the power leads down through the vero-board, then this will make them far less prone to coming off due to metal fatigue, after they have been moved about a bit (this is a point worth remembering whenever you are connecting flying leads to vero-board). Connect the +5V lead to bus line 75-78, and the GND lead to bus line 67 (which SHOULD be a ground, if your motherboard is wired properly). Alternatively, connect the power leads to the power test points on the CPU board.

The circuit should be fairly self-explanatory.

I also include a circuit for Nascom 2 owners to page out Nas-Sys, as no-one seems to have worked this out. Nas-Sys is paged in after RESET, and when port 0 bit 2 goes low.



IC1 = 74LS00

IC2 = 74LS32

MONITOR PAGER

TECHNICAL NOTE ABOUT 1797 DISK CONTROLLER CHIP

By Richard Beal

During the development of RP/M V2.3 an interesting feature of the 1797 was investigated. In order to determine if there is a disk card present in the system, RP/M outputs a value to the track register of the 1797 and then reads the value back. If it is not correct then RP/M assumes there is no disk card and gives the "No disk" message.

On page 20 of the Western Digital technical note there is a comment in microscopic print that the data cannot be read back correctly for 4 microseconds. In fact RP/M 2.1 waited 5 microseconds and operated correctly. However the delay needed depends on several factors and careful tests have revealed that the following number of NOPs (for example) are needed between the OUT and IN instructions for the correct value to be read back reliably. This value corresponds to the number of microseconds since each NOP uses 4 T states and the processor operates at 4 MHz.

Disk size	Density	Microseconds
5.25	DD	5
5.25	SD	13
8	DD	2
8	SD	6

PUT A REAL TIME CLOCK INTO YOUR CBIOS

By C. Bowden

This article explains how I managed to alter my CBIOS so that I was able to display the Date and Time on screen, using a Gemini GM816 I/O card which contains a REAL TIME CLOCK. Readers may already have an R.T.C. available on either the Gemini GM822 or GM816 card, or some other commercial or homebrew card. Many readers will also either have a `source` listing of their CBIOS, or be using one of the `SYS` CBIOS's, in which case, the method described can be used with appropriate modification. If no listing is available, I have given at the end of the article some suggestions that may help, although I have not tried them.

The idea of having a real time clock on my computer has appealed to me for some time. I have used a couple at work. One was a simple scheme, using a 58174A hung onto the PIO of a Nascom 2 that is running all the time carrying out a control job. The other one, in a Gemini GALAXY 2, used the card produced by E.V. Computing that attaches to the Gemini GM812 IVC card. The latter solution is particularly nice as it does not interfere with the main CPU at all, and the time updates irrespective of any task that the main CPU is doing. The card offers several other features too, but the cost is of course higher.

Having an RTC in the machine is of course all very well, but it is obviously an advantage to be able to see the date and time displayed on the screen at all times, and also to be able to `call` up the clock from user programs. A number of example programs have been printed in this magazine from time to time, describing how to build and/or program such clocks, but little has appeared on how to incorporate one into CP/M (versions 1.4 and 2.2 - version 3 supports a clock).

After some thought and a bit of trial and error, I succeeded in doing this with reasonable success. In this article I will describe how I did this with CP/M 2.2, using a Gemini GM816 I/O card and SYS version 16. Much of the methods used will be applicable to other software and hardware.

When considering the use of a Clock, there are several questions that must be answered before it can be decided whether a particular method is satisfactory. The first consideration is one of accuracy. How accurate must the clock be, and how accurate must the display be? The two are not necessarily the same. The Clock hardware used might be highly accurate, with errors of less than a few seconds in a year, but if the clock display is only updated every hour, then that is the limit of accuracy that can be obtained from the display. (A program might still read the clock much more accurately.) For most purposes, it would probably be acceptable for the clock to be accurate to better than a few seconds a day, with a display that updates at about once every second. This was my target, and it was easily attained, but with one minor restriction. There may be occasions when the Screen clock display will `freeze` for a minute or two, although it will then be corrected at the next update.

This restriction can best be explained by investigating how a clock might be incorporated into CP/M. Most clocks are capable of causing an Interrupt, and the clocks mentioned are no exception. There is however a distinct problem in using an Interrupt. Firstly, none of the simple clock chip systems generate a vector (although one used via a PIO can use the PIO vector). Use of most of the RESTART vectors is also not available for the same reason, unless

additional hardware is used, and in any case both CP/M and NAS-SYS use page one of memory, which is where the `RST` locations are operative. Assuming a vector can be generated, there is still the problem of where to allow interrupts to be active within the program. It would be necessary to use EI and DI instructions carefully, avoiding the whole of the CCP and BDOS areas, and most of the CBIOS. The area diminishes until one is left with a few I/O areas such as the Keyboard. Fortunately this conclusion greatly simplifies the task, because computers spend nearly the whole time waiting for keyboard input.

There are occasions when the machine is not awaiting Keyboard input, as during Disk I/O, Formatting text, Compiling or Assembling a Source File, or perhaps copying files. Some of these activities can last for a few minutes. If one is content to allow the clock display to `freeze` for these few times, but update regularly at all other times, then the solution to the problem becomes much simpler. The difficulty of implementing interrupts has been mentioned. Since clock updating is restricted to such a `small` area of the program, in terms of code, then there is no real advantage in using Interrupts, and Polling will give the required performance.

The method that I have used is therefore based on integrating the clock `polling` code into the keyboard scan code.

A number of short listings of source code are given. These show in full the extra code that I have added to SYS vers. 16. to permit conditional assembly for the Clock with options for a couple of extra features. Any code shown in lower case is `SYS` code, included to show the position of my additions, while Upper case code is my added code. (Except for the `headings` enclosed in "*" characters.) Modification for other SYS's or the Gemini BIOS's should also be relatively easy, if the Source Code is to hand. In the example listings in this article, it is assumed that the system is using a keyboard attached to the Gemini GM812 IVC card, and that the RTC chip is addressed as 16 Ports from address 20H, as on the Gemini GM816 card. Modification of the clock software for PIO format should be along the lines of previous articles in the 80-BUS News, and other magazines. I have not looked at the problems that might arise in outputting the Clock to a NASCOM format screen. If a NASCOM keyboard is used the clock code can probably be fitted into that area, but I have not tried this method.

Code extract 1 lists additional `EQU`ates that need to be inserted into module one of SYS. If CLOCK is not true, then all clock related code will be omitted in subsequent modules. If Clock is TRUE, then all related clock routines will be included in the assembly. Setting LOCK to TRUE will cause code to be included into the WARM BOOT routines that will `lock` the top line of the IVC screen display. This will stop the clock display from scrolling `through the top` of the screen. I usually have both NASCOM and Gemini keyboards on my machine, but the RTC code in this article is relevant to the keyboard on the IVC only.

Some of the clock code is a straight copy of the examples given in the GM816 manual and I make no claim as to its originality. The variable NMRREG is set to the number of RTC registers that are to be read. The variable OFFSET can be chosen such that the Clock display on the Screen is updated every 1 Sec., 10 Sec., or 1 Minute as desired. The RTC chip is of course updating its own registers every .1 Second. There is only about 10 Bytes left in SYS at

present if I include all of my normal Hardware and Software options, so there is not enough room to allow inclusion of the extra code and tables needed to display the day name unless I omit something else or alter MOVCPM so that CP/M leaves room for an even larger CBIOS. Examination of the code listed later will show that this option will only be allowed if NKBD is set FALSE. (This is the most suitable routine for me to omit, since I normally use the Gemini GM827 keyboard.)

In some cases, there may well be enough room for the extra code and data anyway, since some other features such as Virtual Disk may not be incorporated in the CBIOS, and it may not be necessary to omit routines or modify MOVCPM.

The final part of this extract shows the RTC ports added to the relevant section of SYS.

The second set of extracts are the extra code that has been added into the Workspace/Boot/High Level I/O routine section of SYS (which is module 4 of SYS 16).

In order to allow user programs to easily access the Clock, it is necessary to set up some method of allowing the 'calling' program to find the clock routines. The simplest way of doing this is to add an extra 'Jump to Clock Routine' to the jump table at the start of the CBIOS. Since the offset of this jump from the start of the CBIOS is known, it can be used as the access point. Some details of how to do this are given in App. 1. A look at the code for the clock will show that on 'RETURN', HL holds the start address of STRING (or STRNG1), which holds the clock data for subsequent processing.

A marker is also placed after the additional jump in the 'jump' table. The calling program can check for this marker before attempting to read the clock, and can thus be made to avoid trying to read a non-existent clock. This can sometimes cause the program to lock into a permanent loop from which the only exit is 'RESET', unless additional software is used to 'time out' any such loop.

Additional space is reserved at the end of the 'SYS' workspace for clock related data and variables. A space of 11 bytes is allocated to 'REGS'. The RTC chip registers are stored in this space as they are read. After some further processing, REGS will hold the ASCII equivalents of the clock register data. Once the clock has been successfully read, the data is moved to 'STRING/STRNG1', being formatted as it will be printed on the screen in the process. Two sizes of output string are possible, depending whether or not the day name is to be included, as previously described. If the name is to be included, the store starting at 'STRING' is used, but if no name is to be output, the space starting at 'STRNG1' is used. The space UDSTOR is used to hold a copy of the Seconds, Tens of Seconds, or Minutes register of the RTC, to be used as described later.

The other bit of code added to module 4 will conditionally allow the top line of the GM812 IVC card to be locked, to prevent the time/date from scrolling. I added the necessary code into the 'WARM BOOT' routines, just before the 'SYS' code to restore the current default drive. I do not think the actual position of the extra code should be very critical. I chose this place as it was near the end of the warm boot.

In order to keep my extra code to a minimum, I used a number of inbuilt `SYS` routines to carry out certain tasks. These are:

```

SETCUR      ;Set cursor on IVC from values in HL, adding any
             ;offsets needed. H holds ROW: L holds COLUMN
CRTX        ; Outputs Char. in `A` to IVC card.
SAVCUR      ; Saves current cursor location in SYS workspace.
             ; ( In a Store `LABELLED` - `CURSOR` )

```

The first action needed in the lock top line routine, is to store the current position of the IVC cursor, which will be well down the screen on `COLD BOOT`, having just printed the sign-on messages. SETCUR is then called to set the cursor to the start of line two, and ESC "M" is sent to the IVC to lock the top line. (see IVC software manual). The Cursor is then replaced at its original location down the screen. Note that CLOCK and LOCK must both be `TRUE` for this code to be assembled.

At this point, I ought to mention the meaning of the `R` that appears in the source code, for those not accustomed to SYS BIOS listings. SYS is designed to load from disk, where it is stored as a .COM file. It loads to low memory, and then a `Relocation` section loads SYS over the original CBIOS, and adjusts all references to memory locations that are position (address) sensitive. The only way that SYS knows which bytes to adjust is by being told by the person writing the code. The addition of the `R` after ALL memory references that need to be adjusted will ensure that SYS will include them in a table of such addresses, and will carry out the necessary relocation.

The final section of SYS that has code added is the Low level I/O routines. (Module 5 of SYS 16.) The actual Clock routines are added into the IVC Keyboard routines in my SYS if CLOCK is set `TRUE`.

The first part of the code sets HL to point to UDSTOR which will be loaded with the ASCII value of Seconds, Tens of Secs. or Minutes (depending on OFFSET). This value is then read in from the clock, checked for validity, converted to ASCII, and then placed into UDSTOR. The corresponding data stored in REGS at the last clock update is then compared with UDSTOR. If the values are the same, then the clock has not changed, and no clock processing occurs, but the program goes on to scan the keyboard. If the values are different, the clock has changed and so it must be updated. By making the comparison on Data that alters either by the Second, ten Seconds, or Minute the clock can be made to update accordingly as the operator desires.

The keyboard is being scanned many times a second, and if the clock is allowed to update every time, the net effect is to greatly reduce the speed of many operations. Since the check just described only takes some twenty or thirty microseconds, it has no noticeable effect on normal operations, and prevents the clock from intruding into them. I have not calculated the time that the clock routine takes, but I would guess that it takes about one millisecond, and so there is no noticeable effect on operations when it updates.

If the clock has changed, a call is made to READIT, which is the actual clock read and display subroutine. This subroutine may be called from external programs via the extra jump that was added to the `jump table`. This module starts by reading the number of clock registers defined in NMRREG into the

memory store REGS. The 'C' register of the Z80 points to the first clock port, and they are read in via an 'INI' instruction, which automatically increments 'C' and 'HL', decrements 'B', and loops until 'B' is Zero. The values read are then 'scanned' in turn, and 'ANDed' with 0FH to mask off the four high order bits, which are not determined by the R.T.C. chip. The data is then compared with 0FH. If it is equal to this value, then the clock register in question was updating when it was read. If the clock was updating, then REGS contains at least one invalid reading, so the registers are all read again. If the data is valid, it is converted to ASCII by adding 30H, and then it is put back into the same memory location in REGS again. All bytes in REGS are checked in this way. When all data is valid, the program then just has to set the data up in a printable form. Unfortunately, the order of the registers as read from the clock does not match too well the type of format that we normally use in the U.K. How the data is presented is a matter of preference, but I have chosen output in the format :-

Sat 25/04/84 18:33:55 i.e. - Day Name - Date - Time

If the day name is to be processed, the 'DE' register is pointed to the start of the name table, 'BC' is set to three, since the names are three characters long, and the 'day of week' value stored in REGS is read. The value read is then 'multiplied' by three and added to 'HL'. Finally 'DE' is added to 'HL' so that the latter points to the correct name. If the value read back is either 0 or greater than 7, due to some error, the blank name of DAY0 is used. 'DE' is then set to point to the start of STRING, which will hold the formatted data for printing. The three byte name is then copied into STRING using 'LDIR', and 'DE' is incremented once more to point to the units of the day of the month. (i.e. To STRNG1.)

If the day name was not processed however, (i.e. NKBD was TRUE), then 'DE' is set to this value as a definite instruction. Two very similar routines follow, that move the two byte data values associated with Day of month, Month, Hour, Minutes and Seconds into the appropriate place in STRNG1. The 'DE' registers act as a pointer into the string, and 'HL' points to the data source in REGS. 'B' and 'C' act as counters. The code is commented and fairly straightforward, so it will not be described in greater detail here.

On completion of this code, the data will be in STRNG1, and possibly also STRING, ready to be printed on the screen. Before printing, the cursor must be set to the right place. SYS provides a store in its workspace called CURSOR that can be used to save the location of the cursor. To play safe, I decided to save whatever was in this store before I used it, in case it was holding data relevant to some other program that might be running when the clock is updated. The data in 'CURSOR' is copied to HL and then 'PUSH'ed on to the stack. The current cursor location is then saved, using the 'SAVCUR' routine. The cursor is then temporarily turned off, using an IVC command, to stop it flashing as the clock display is updated.

The 'SETCUR' routine is then used to place the cursor at the clock display position in the top line of the screen. This position depends on whether the Day name is to be displayed. The 'H' register holds the row data, and 'L' the column. The data is then printed on the screen, using 'HL' to point to the source of the data to be printed from STRING/STRNG1, and the SYS 'CRTX' routine is used to send each byte to the IVC. The routine stops when it finds the '0' character that is used to mark the end of the output string.

The cursor is then restored to its original screen place using `SETCUR`, the original contents of CURSOR are `POP`ed off the stack and then restored, and then the cursor is then turned on again, using the appropriate IVC command.

Finally, `HL` is loaded with the address of STRING, or STRNG1 as appropriate, so that if the clock read module was called from an external program, that program would be able to find the address of the clock data for subsequent use.

This completes the code needed to modify SYS Version 16. I am using `SYS` modified as described while typing this article in with GEMPEN, and the display is showing the Day name, full date, and time, updating every second. There is no noticeable effect on the clock display during typing, even if the repeat keyboard feature is used. During formatting of this text, which is about 28K long, however, the display stops for about a minute and a half. The `freezing` of the clock display can be made use of, to time activities such as formatting, disk I/O and assembly.

Appendix 1. Finding the Clock Routine through the added `JUMP`

Locations 0002 and 0003 in memory hold the `WARM BOOT` address, which is 3 bytes into the BIOS. The `JP READIT` instruction is 30 bytes further on, and the marker is another 3 bytes on. For example, on my machine, locations 0002 and 0003 hold 0EE03H, so the `clock jump` is at 0EE33H, and the marker "RTC" starts at 0EE36H. The clock can therefore be accessed by a CALL 0EE33H instruction. The clock access can be performed in a better way as follows :-

```
LD      HL,(0002)
LD      DE,0030H
ADD     HL,DE
CALL    CLOKJUMP
;
;      etc; Routines to process clock data
;
;
;
CLOKJUMP:      JP (HL)
;
;
```

Appendix 2. Patching in a Clock without Source Code.

This will not be very easy, especially for persons unused to getting into CP/M with Disassemblers and Debuggers. I think that one feasible approach is to carry out the following steps.

1. Look at CP/M in memory with DDT/ZSID/GEMDEBUG to find the CBIOS start, and to see if there is any room at the `top` of memory for extra code.

2. If there is not enough room, use MOVCPM to make a new CP/M that uses a little less memory, or alter the MOVCPM program to reserve a little bit more CBIOS space. This can be done by changing location 023DH in MOVCPM. By adding 1 to this value an extra 256 bytes is added to the BIOS space reserved, and so on.
3. If this method is being tried, I assume that the BIOS Source Code is not available. I hope that I am not disclosing too much information in the following suggestions, which I am basing on my original CBIOS.
 - a) The 'Signon' messages are at the end of the CBIOS.
 - b) The Workspaces are just before this.
 - c) The PUTVID and GETVID routines are just before this.
 - d) Keyboard and Editing routines come just before this.
4. If you can find these routines and data areas, then you can try out the following suggestions. Choose an address just AFTER your CBIOS, where you will place the Clock AND the IVC Keyboard routines. The latter are not very long. If you can locate the equivalent IVC routines, it will be possible to 'CALL' them. If not then they must be added in.
5. Assemble the Clock routines and include the Keyboard code, with 'ORG' set for your chosen address. Any Clock workspaces will have to reside here as well.
6. Patch this code into the end of the CBIOS, and place a 'JUMP' at the start of the code of the original keyboard routine, to the new code.
7. If this all works, use DDT/ZSID/GEMDEBUG to move a copy of CP/M down over MOVCPM and save it as CLOKBIOS.SYS or similar name. You can then put it on to the SYSTEM tracks of your disk with SYSGEN. Note that this is a fixed size BIOS, and if you alter memory size, and regenerate CP/M with MOVCPM, you will have to re-assemble and patch the new code.
8. Top line locking might be achieved by jumping to extra code at the end of the CBIOS, from the warm boot routines, or by adding the command into the 'update' part of the routine. Since it would then be executed every time that the keyboard is scanned, there would be an extra time penalty.
9. One final problem might remain. SYSGEN and COLD BOOT may need altering to READ/WRITE the increased code in the system tracks.

My previous article in the 80-BUS NEWS, on how to insert 'SYS' into the CBIOS gives quite a bit of information on some of these problems, particularly 7 and 9 .

WANTED

Wanted, Nascom IMP Printers, need not be in working order. Please write with details to D. Torrens, c/o 51 St Martin, Marlborough, Wilts

Extract 1. Code added to SYS16. Module 1.

a) Conditional Equates

```

; Output devices
CLOCK EQU TRUE ;EQUATE FOR R.T.C.
LOCK EQU TRUE ;LOCK I.V.C. TOP LINE. CLOCK MUST BE TRUE AS WELL
NRREG EQU 11 ;11 R.T.C. REGS. TO READ
OFFSET EQU 0 ;0 FOR 1 SEC, 1 FOR 10 SEC, 2 FOR 1 MINUTE UPDATE

```

b) Additional Ports.

```

; ***** PORTS *****
; *****
; *****
; *****
IF CLOCK
    EQU 20H ;R.T.C. Base address for CM816 Card
ENDIF

```

Extract 2.

a) Addition to Jump Table

```

jp sectrn ; Sector translation
;
IF CLOCK
    JP READIT ;CLOCK ROUTINE
;
DEFB "RTC" ;Marker to say RTC exists in this SYS.
ENDIF

```

b) Extra Reserved Space and Messages at End of the Workspace

```

;
REGS: DEFS 11 ;TEMP. STORE FOR REGS
IF NOT NKBD
    STRING: DEFB " "
    ENDIF
STRING1: DEFB " / 84 : : ",0
UDSTOR: DEFS 1 ;COMPARISON DATA FOR UPDATING CLOCK
;
IF NOT NKBD
    DAY0: DEFB " " ;THESE CAN BE PLACED IN ONE LONG
    DAY1: DEFB "Sun" ;STRING IF DESIRED, AS ONLY LABEL
    DAY2: DEFB "Mon" ;DAY0 IS REFERRED TO IN SOURCE CODE.
    DAY3: DEFB "Tue" ; eg :- " SunMonTueWedThuFriSat"
    DAY4: DEFB "Wed"
    DAY5: DEFB "Thu"
    DAY6: DEFB "Fri"
    DAY7: DEFB "Sat"
    ENDIF
ENDIF

IF CLOCK
    LD HL,UDSTOR ;POINTER TO TEMP .STORE
    R
    IN A,(RTC+2+OFFSET) ;READ APPROPRIATE CLOCK REG.
    AND OFH
    CP OFH ;SEE THAT IT IS VALID
    JR Z,AGN ;LOOP IF NOT
    ADD A,30H ;MAKE ASCII
    LD (HL),A ;SAVE IN UDSTOR
    LD A,(REGS+OFFSET) ;GET VALUE OF CORRESPONDING
    R. ;DATA ON LAST CLOCK UPDATE
    CP (HL) ;ARE THEY THE SAME ?
    JR Z,TEXTIT ;IF EQUAL, NOT TIME TO UPDATE YET
    CALL READIT ;NOT EQUAL SO UPDATE CLOCK
    R
    TEXTIT: ENDIF
;
ld a,esc ; Test for character
call crtx

```

c) Code to Lock top line of I.V.C. Screen

```

IF CLOCK
IF LOCK
    CALL SAVCUR ;SAVE CURRENT POSITION OF CURSOR
    R
    LD HL,0100H ;START OF SECOND ROW.
    CALL SETCUR ;SYS- SET CURSOR ROUTINE
    R
    LD A,01BH ;ESC-
    CALL CRTX ;O/P IT TO IVC
    R
    LD A,"M" ;LOCK TOP LINE
    CALL CRTX
    R
    LD HL,(CURSOR) ;NOW SET CURSOR TO ORIG. POSITION
    R
    CALL SETCUR
    R
    ENDIF
ENDIF

```

-----***-----

```

*****
***** KEYBOARD ON VIDEO CARD *****
*****

```

```

IF CLOCK
    LD HL,UDSTOR
    R
    IN A,(RTC+2+OFFSET)
    AND OFH
    CP OFH
    JR Z,AGN
    ADD A,30H
    LD (HL),A
    LD A,(REGS+OFFSET)
    R.
    CP (HL)
    JR Z,TEXTIT
    CALL READIT
    R
    TEXTIT: ENDIF
;
ld a,esc ; Test for character
call crtx

```

```

;
; scf
; ret
;
; endif
;
; IF CLOCK
;
; MAY BE CALLED FROM EXTERNAL PROGRAMS VIA JUMP TABLE. RETURNS WITH HL
; POINTING TO CLOCK DATA.
;
; READIT: LD C,RTC+1
;          LD B,NRREG
;          LD HL,REGS
;          R
;
;          INC C
;          INC NZ,RDT1
;
;          LD B,NRREG
;          LD HL,REGS
;          R
;
;          LD A,(HL)
;          AND OFH
;          CP OFH
;          JR Z,READIT
;
;          ADD A,30H
;          LD (HL),A
;          INC HL
;          DJNZ SCAN
;
; STORE NOW HOLDS VALID ASCII DATA SO PUT IT INTO A STRING BUFFER FOR SCREEN
; DISPLAY, STARTING WITH DAY NAME IF SUFFICIENT SPACE FOR THE CODE.
;
; IF NOT NKBD
;
; MOVDAY: LD DE,DAYO
;          R
;          LD BC,3
;          LD A,(REGS+8)
;          R
;          AND OFH
;          CP 8
;          JP P,MOVDY1
;          R
;          LD HL,0000
;          LD L,A
;          ADD A,L
;          LD L,A
;          ADD A,L
;          LD L,A
;          ADD HL,DE
;
;          ;POINT DE TO PLACE IN 'STRING' FOR
;          ;THE DAY NAME TO BE INSERTED
;          ;MOVE THE 3 BYTES
;          ;SPACE BETWEEN DAY NAME AND DATE
;
;          ;START OF DATE AND TIME STRING
;          ;IN CASE DAY NAME NOT PROCESSED
;
;          ; POINTER TO TENS OF DAY OF MONTH
;
;          ; 2 BLOCKS TO MOVE (MONTHS AND DAYS)
;          ; COPY TENS & UNITS OF DAYS OF MONTH
;          ; GET BYTE FROM REGS.
;          ; MOVE IT TO STRING
;          ; BUMP DESTINATION
;          ; REDUCE SOURCE POINTER
;          ; LOOP FOR 2 BYTES EACH TIME
;          ; SKIP OVER -/+ CHAR.
;          ; POINTER TO TENS OF MONTHS
;
;          ; IF 'C' NOT ZERO, MONTHS NOT YET
;          ; COPIED TO THE STRING
;          ; MOVE PAST TENS OF YEAR
;          ; DITTO - UNITS
;          ; NOW SKIP THE SPACE BETWEEN DATE & TIME
;          ; POINTER TO TENS OF HOURS
;
;          ; HOURS, MINS., & SECS. TO MOVE
;          ; 2 BYTES TO MOVE EACH TIME ROUND
;          ; SAME AS ABOVE
;
;          ; CLOCK DATA NOW IN STRING/STRNG1, SO MOVE CURSOR AND PRINT IT
;          ;
;          ; CSRSV: LD HL,(CURSOR)
;          ;          R
;          ;          PUSH HL
;          ;          CALL SAVCUR
;          ;          R
;
;          ; CSROFF: LD A,01BH
;          ;          CALL CRTX
;          ;          R
;          ;          LD A,"D"
;          ;          CALL CRTX
;          ;          R
;
;          ; TURN OFF CURSOR WHILE PRINTING CLOCK

```

```

; POSCUR: IF NOT NKBD
;           LD HL, 003BH
;           ELSE
;           LD HL, 003FH
;           ENDIF
;           CALL SETCUR
;           R

; PRITIM: IF NOT NKBD
;           LD HL, STRING
;           R
;           ELSE
;           LD HL, STRNG1
;           R
;           ENDIF

; PRTIM1: LD A, (HL)
;          CP 0
;          JR Z, RESCSR
;          CALL CRTX
;          R
;          INC HL
;          JR PRTIM1

; RESCSR: LD HL, (CURSOR)
;          R
;          CALL SETCUR
;          R
;          POP HL
;          LD HL, (CURSOR), HL
;          R

; CSRON: LD A, 01BH
;         CALL CRTX
;         R
;         LD A, "E"
;         CALL CRTX
;         R

; IF NOT NKBD
; LD HL, STRING
; R
; ELSE
; LD HL, STRNG1
; R
; ENDIF

; RET
; ENDIF
; page

```

```

; POSITION ALLOWING FOR DAY NAME
; POSITION WHEN NO DAY NAME
; SET CURSOR TO POSITION

; NOW COPY 'TIME' STRING TO SCREEN
; HL POINTS TO SOURCE STRING.

; START HERE IF NO DAY NAME

; GET NEXT BYTE FROM THE STRING
; END OF STRING MARKED BY A '0'
; SKIP WHEN A '0'
; SEND BYTE TO I.V.C. CARD.
; 'BUMP' POINTER
; LOOP TILL END OF STRING

; NOW PUT CURSOR TO PREVIOUS POSITION

; RESTORE ORIG. VALUES TO 'CURSOR'
; STORE

; TURN CURSOR ON AGAIN

; NOW SET 'HL' TO POINT TO START OF
; TIME/DATE STRING TO ALLOW ANY
; EXTERNAL PROGRAM TO FIND DATA.

; BACK TO CALLER

```

page

ASCII Character Codes

Dec	Hex	CHR	Dec	Hex	CHR	Dec	Hex	CHR
000	00H	NUL	043	2BH	+	086	56H	V
001	01H	SOH	044	2CH	,	087	57H	W
002	02H	STX	045	2DH	-	088	58H	X
003	03H	ETX	046	2EH	.	089	59H	Y
004	04H	EOT	047	2FH	/	090	5AH	Z
005	05H	ENQ	048	30H	0	091	5BH	[
006	06H	ACK	049	31H	1	092	5CH	\
007	07H	BEL	050	32H	2	093	5DH	]
008	08H	BS	051	33H	3	094	5EH	^
009	09H	HT	052	34H	4	095	5FH	_
010	0AH	LF	053	35H	5	096	60H	`
011	0BH	VT	054	36H	6	097	61H	a
012	0CH	FF	055	37H	7	098	62H	b
013	0DH	CR	056	38H	8	099	63H	c
014	0EH	SO	057	39H	9	100	64H	d
015	0FH	SI	058	3AH	:	101	65H	e
016	10H	DLE	059	3BH	;	102	66H	f
017	11H	DC1	060	3CH	<	103	67H	g
018	12H	DC2	061	3DH	=	104	68H	h
019	13H	DC3	062	3EH	>	105	69H	i
020	14H	DC4	063	3FH	?	106	6AH	j
021	15H	NAK	064	40H	@	107	6BH	k
022	16H	SYN	065	41H	A	108	6CH	l
023	17H	ETB	066	42H	B	109	6DH	m
024	18H	CAN	067	43H	C	110	6EH	n
025	19H	EM	068	44H	D	111	6FH	o
026	1AH	SUB	069	45H	E	112	70H	p
027	1BH	ESCAPE	070	46H	F	113	71H	q
028	1CH	FS	071	47H	G	114	72H	r
029	1DH	GS	072	48H	H	115	73H	s
030	1EH	RS	073	49H	I	116	74H	t
031	1FH	US	074	4AH	J	117	75H	u
032	20H	SPACE	075	4BH	K	118	76H	v
033	21H	!	076	4CH	L	119	77H	w
034	22H	"	077	4DH	M	120	78H	x
035	23H	#	078	4EH	N	121	79H	y
036	24H	\$	079	4FH	O	122	7AH	z
037	25H	%	080	50H	P	123	7BH	{
038	26H	&	081	51H	Q	124	7CH	
039	27H	'	082	52H	R	125	7DH	}
040	28H	(	083	53H	S	126	7EH	..
041	29H	)	084	54H	T	127	7FH	DEL
042	2AH	*	085	55H	U			

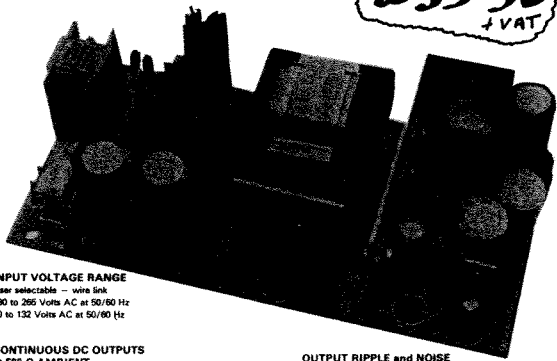
Dec=decimal, Hex=hexadecimal (H), CHR=character, LF=Line Feed, FF=Form Feed, CR=Carriage Return, DEL=Rubout



DATA 2250

MAINS ISOLATING SWITCH MODE POWER SUPPLY

£39-50
+VAT



INPUT VOLTAGE RANGE
User selectable - wire link
180 to 266 Volts AC at 50/60 Hz
90 to 132 Volts AC at 50/60 Hz

CONTINUOUS DC OUTPUTS
at 50° C AMBIENT
+ 5 V DC at 3.0 Amps
+ 12 V DC at 1.3 Amp
- 12 V DC at 0.3 Amp

OUTPUT NOTES:
(a) Total power output at power supply terminals not to exceed 36 Watts
(b) A minimum load of 0.5 Amp on +5 V output is required for specified regulation
(c) Output ratings can be increased under controlled ambient conditions. (Contact VMS Technical Services Department)
(d) The following individual continuous DC outputs at 50° C ambient are the maximum permitted ratings under power "trade-off" conditions
+ 5 V DC @ 4.0 Amps
+ 12 V DC @ 2.0 Amps
- 12 V DC @ 0.3 Amp

LINE REGULATION
0.5% maximum over specified input voltage range

LOAD REGULATION
+ 5 V output (over range 0.5 to 3.0 Amps) 1%
+ 12 V output (over range 0.25 to 1.3 Amp) 10%
- 12 V output (over range 0.0 to 0.3 Amp) 5%

OUTPUT RIPPLE and NOISE
+ 5 V output 50 mV peak to peak maximum
+ 12 V output 100 mV peak to peak maximum

OUTPUT HOLD UP
> 1 msling cycle at 50 Hz

OVERCURRENT PROTECTION
All outputs are protected against accidental short-circuits

OVERVOLTAGE PROTECTION
Operates on the +5 Volt output at a nominal level of 6.2 Volts

OPERATING TEMPERATURE
+ 50° C maximum

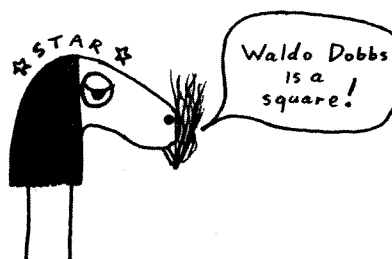
SAFETY and ISOLATION
4.0 kV input to output (relevant components)
2.2 kV DC input to output and earth (complete unit)
0.7 mA maximum earth leakage

FUSES
A Mains fuse (2.0 Amps TD) is provided on the PCB

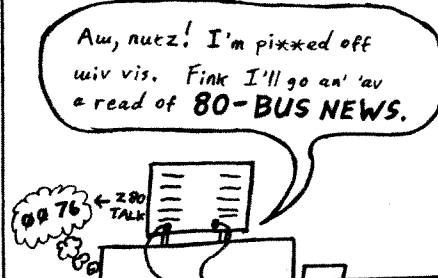
AMERSHAM COMPUTER CENTRE LTD.

Tel: 02403 22307. Telex: 837788
18 Woodside Road, Amersham, Bucks.

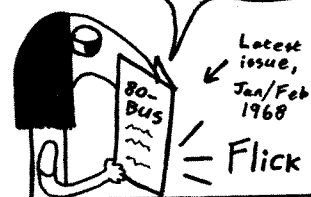
Lawrence and the PolyDos
Users Group.



Lawrence, hard at work (as usual) zapping
space invaders - when suddenly



Oi! Wot's vis!!?
Vat Dr. wot's-'is-face in
Taunton 'as started a club
for CP/M users. Hmm....



So....

A few days later, a reply.

Dear Doc,
I see you've got a users
group going for vis CP/M fink.
CP/M will NEVER catch on.
Why don't you buy POLYDOS
an' join 'P.U.G.'!!
Lawrence.

Dear Lawrence,
1. I like CP/M.
2. I'm too cheap to buy POLYDOS.
3. I don't like talking to people
who drag their knuckles on
the ground when they walk!
Dr. Dark.

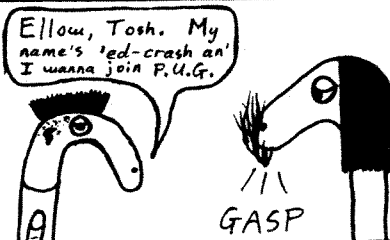
HUH!!

Anyway, s'a well known
fact vat Doc. Dark's
Nascom is a factory-built job!

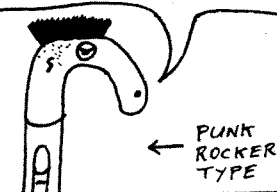


Lawrence, thoroughly pigged-off with
life, the universe and everything (etc. etc.)
decides to get 'zonked-out' on

Meanwhile, word of Lawrence's efforts
at trying to start a PolyDos Users
Group had spread far and wide.



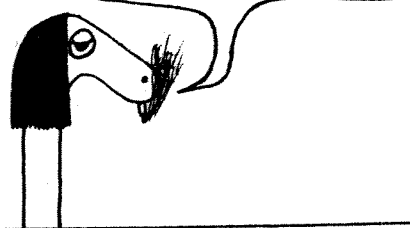
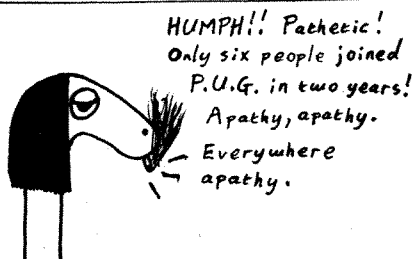
Er, ellow folks. They call me
'ed-crash 'cos vat's wot my
disk R/W heads do most - crash!
I joined P.U.G. to learn more about
PolyDos. Plus, Dr. Dark won't 'av me
in 'is Group - Lawrence'll take anyone!!



Two years later (this forward
temporal jump is surpassed only by the
one in the film '2001'!)

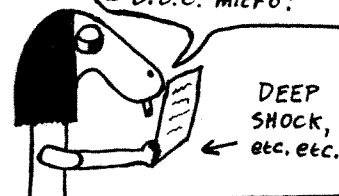
Look lads. FREE
programs, fame, etc. etc.
Also, any PolyDos user wot
don't join P.U.G. is a
diseased mutant - so there!

A few days later - the world ends.



AAAH!!

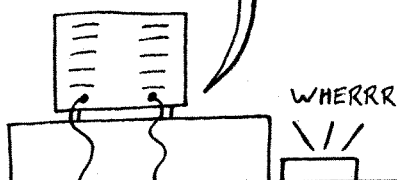
One of ver P.U.G.
members 'as bought
a B.B.C. micro!



I am, to say ver least,
shocked at vis mutinous act.
Fink I'll 'av a pint of
falling-down juice!



PolyDos won't not
never die - an' P.U.G.
will live forever!



POLYDOS USERS GROUP.
29. MARTIN CRESCENT,
TONYRETAIL,
MID. GLAMORGAN.
SOUTH WALES. CF39 8NT.

THE GATEWAY

The author of the highly successful NASPEN and DISKPEN software has now completed his latest masterpiece. Called the GATEWAY, this represents a new concept in database management. The aim is to provide free access to information rather than data restricted within the more normal rigidly defined fields of a conventional database. Designed primarily for use with ASCII text files from any source, the GATEWAY works in a free field format, allowing the GATEWAY to search for logical references from single word length to references contained within text blocks of any size from sentences, paragraphs, etc, with a practical limit of about 20,000 characters (about 5 pages of closely typed A4) per field. GATEWAY searches and constructs 'tag' lists which contain pointers matches found, and then allows storing, merging, printing or copying of the references found.

THE PATHWAY

The output from the GATEWAY can be handled by any text processor, but the PATHWAY text formatting program offers significant advantages as it is designed to make use of the output of the GATEWAY to produce reports, mail shots (complete with mail merging facility), etc. A unique feature is the ability to configure both the input and output formats, where any input character may be converted and similarly any output character can be converted into a different character or string.

Both GATEWAY and PATHWAY are system independent programs, and will run on any CP/M80 machine.

The unique flexibility and power of GATEWAY and PATHWAY is difficult to describe in this restricted space. The complete manuals are available separately and the cost will be credited against the purchase of the suite.

GATEWAY and PATHWAY with manuals	115.00 inc. VAT.
Manuals only	5.00
Carriage & packing	1.00

SUPER DISKPEN (Version PENVG:3)

DISKPEN has been rewritten and revised. This popular text editor/formatter now includes a 'HELP' facility, and new features for the print control of the most popular printers, underline, bold, etc (also user patchable for the less popular types). New features include block delete, better move commands, new cursor control, optional hyphenation, visible indentation setting and lots more. A major enhancement is the ability to handle overlay files so that PEN can use auxiliary packages such as the MAXiFILE free field file searching utility or the print spooling utility.

The new DISKPEN is useable on all Gemini multiboard computers (Galaxy, Kenilworth, Quantum) and Nascom/Gemini hybrids, (MAPPEN is available for users of the MAP video card). DISKPEN is available as an upgrade to earlier DISKPENS and GEMPENS at 17.25 inc. VAT., or to new purchasers at 57.50 inc. VAT. (Please state disk format when ordering.)

MAXiFILE overlay	23.00 inc. VAT. (20.00 + VAT)
SPOOLER overlay	17.25 inc. VAT. (15.00 + VAT)
Carriage & packing	50p

BDOSZ

Yes, you guessed it. Some enterprising person has now 'disconbooberated' the BDOS in CP/M and rewritten it as a Z80 program. Its fully compatible with the original with no bugs found to date. Because it's written in Z80 code it's smaller, this has allowed room for tidying up all the annoying stupidities in the original BDOS so that errors like:

BDOS ERROR ON x: R/O

which usually causes you to lose everything you've just done, becomes the far more helpful:

Disk x: is set R/O

Do it anyway? (Y/N/^C)

Which, of course, means you don't lose anything. It even allows you to change disks when they are full without loss of data. In all, a lovely piece of software. Available in most popular 5.25" formats (please state when ordering) at 11.50 inc. VAT.

Carriage & packing 50p

ON A GEMINI SVC OR IVC

P	
PRE	
PREST	The new Henelec PRETZEL 2 terminal
PRESTEL	software turns your CP/M based Nascom
ESTEL	or Gemini, fitted with the Gemini GM812
TEL	or GM832, into a full blown terminal
L	for use with the BT PRESTEL service.

Automatic download of character sets gives proper PRESTEL style characters and mosaics. Automatic select of 40 by 24 screen size, full PRESTEL style screen wraparound. Message facilities allowing full keyboard entry to the bill-boards. Downloading of PRESTEL pages to disk. Optional auto-dial (with suitable modem) and ID transmit. Suitable for most 1200/75 BAUD modems, but designed around the Minor Miracles 2000, the Telemod 2 and the GEC LTU-11. Supplied on disk as an M80 source file with utilities, documentation and demo pages:

Please state format when ordering:

29.95 inc. (26.04 + VAT). Carriage & packing 50p

(Note: the trade name PRESTEL and the PRESTEL logo are the property of British Telecom.)

TRANSDATA MODEM CARDS.

Brand new, tested and aligned 300 BAUD answer and originate acoustic telephone modem cards manufactured by TRANSDATA for their model 317 acoustic modem. Full CCITT specs. Requires only a few components to complete (no case available). Computer interface is 300 BAUD RS232 I/O. The power supplies of +12 volts at 180mA and -12 volts at 170mA are required. Card size 350mm x 105mm. May be directly connected using the GEC LTU 11 coupler (see below). Full circuit description, drawings and connection data supplied. Suitable for use with computers fitted with 300 BAUD RS232 I/O. Software for Nascom & Gemini published in recent 80-BUS NEWS.

Card only at 29.95 inc. VAT. (26.04 + VAT)

Supplied with LEDs connector, switches and transducers at 34.95 inc. VAT. (30.39 + VAT)

Supplied with LEDs connector, switches and GEC LTU 11 direct coupler at 45.95 inc. VAT. (39.96 + VAT)

Carriage & packing 1.00

GEC LTU-11 75/1200 BAUD MODEM KIT

The GEC LTU-11 75 BAUD transmit/1200 Baud receive modem kit consists of two components, the modem card and the direct telephone coupler. The modem card accepts I/O at TTL levels at 75 BAUD and returns data at TTL levels at 1200 BAUD. The modem card produces the necessary modem tones which are fed to the direct coupler. The coupler consists of the line isolation transformer which sends and receives the modem tones and has TTL level inputs for the control relays for line seize and auto-dial. Power supplies are +5 volts and -5 volts. The units originally had BT approval, but as these are supplied in kit form, the BT approval is void. Supplied complete with connection data and auto-dial software. GEC LTU-11 modem card 14.95 inc. VAT. (13.00 + VAT). GEC LTU-11 direct coupler 14.95 inc. VAT. (13.00 + VAT). Carriage & packing 75p for each unit.

MINOR MIRACLES 2000 MODEM

The Minor Miracles 2000 direct connect answer and originate modem is an all 'bells and whistles' modem capable of Bell and CCITT protocols at 300/300, 600/75, 600/600, 1200/75 and 1200/1200 BAUD receive/transmit with optional auto-dial and auto-answer. Automatic mode select under software control. Requires RS232 I/O for data transmission and an optional 4-bit port for mode control. Uses the AMD 7910 'world series' modem control chip. BT approved (until you remove the link to activate the Bell protocols). Separate 220-240V AC mains powered. Size approx. 140mm x 70mm x 180mm. Supplied complete with lead terminated in BT620 (the new rectangular type) plug, manuals, etc.

Modem 148.35 inc. VAT. (129.00 + VAT).

Auto-dial option 34.50 inc. VAT. (30.00 + VAT).

Carriage & packing 1.00

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

HENRY'S

COMPUTER SHOP
404/406 Edgware Road, London, W2 1ED
Telephone: 01-402 6822

OPEN 6 DAYS A WEEK

Credit Sales available ask for details.
Official orders welcome.



Order by Post with CHEQUES/ACCESS/ATM or you can telephone your orders

AMERSHAM COMPUTER CENTRE

for MONITORS

COTRON SWORD

Cotron's progressive development of high technology TV monitors for professional users, has enabled them to produce computer monitors that few other manufacturers can match in both quality and price.

With the introduction of the Sword range, a new dimension is offered to micro users. All the Sword monitors incorporate 14" 'Black Glass' tubes, producing very high contrast with bright clean colours.

All Sword monitors will accept both TTL and analog input signals, via a 25 pin 'D' type connector.

We think you will agree that Cotron's Sword range is British technology at it's best.

FEATURES INCLUDE:

14" RGB MONITOR * TTL & ANALOGUE *
18MHz BANDWIDTH * INFINITE COLOUR
PALETTE * PRESTIGE CASE * BRITISH
DESIGN & MANUFACTURE * OPTIONAL
TILT & SWIVEL BASE * STANDARD OR
LONG PERSISTENCE

SABRE

The Sabre is a medium resolution monitor that has a horizontal resolution of 650 pixels and a dot pitch of .40mm, bandwidth is 18MHz.

SABRE-1S	£ 455.00
SABRE-1L	£ 480.00
SABRE-2S	£ 542.00
SABRE-2L	£ 570.00

RAPIER

The Rapier is a high resolution monitor with a horizontal resolution of 850 pixels and a dot pitch of .31mm, bandwidth is 18MHz.

RAPIER-1S	£ 550.00
RAPIER-1L	£ 575.00
RAPIER-2S	£ 628.00
RAPIER-2L	£ 650.00

CABLES

Cable - PLUTO mini palette	£ 32.00
Cable - PLUTO TTL/RGB	£ 32.00

KEY

(1) No Stand
(2) With tilt & swivel stand
(S) Standard persistence
(L) Long persistence

MICROVITEC

CUB 452

The CUB-452 is the standard resolution option from the popular Microvitec CUB range of 14" colour monitors. This model offers 452 x 585 addressable pixels.

This monitor has an RGB type input available as either TTL or linear. A special version (MZ) is also available for the Sinclair Spectrum computer.

The (AP) version of this monitor is designed for use with a video recorder as well as a computer and as such is equipped with both PAL and AUDIO inputs.

1431MS (Metal Case)	£ 199.00
1431LS (Structural Foam)	£ 249.00
1431MZ (TTL + Spectrum)	£ 225.00
1431AP (TTL + PAL + Audio)	£ 225.00

CUB 653

The CUB-653 is the medium resolution option in the Microvitec 14" range. This model offers 653 x 585 addressable pixels.

Available in both TTL and Linear versions the CUB-653 provides the ideal specification for the majority of micro's with high resolution colour and 80 column displays.

1451MS (Metal Case)	£ 299.00
1456LI (IBM-PC)	£ 395.00
1451AP (TTL + PAL + Audio)	£ 340.00
1451DQ3 (Sinclair QL)	£ 239.13

CUB 895

The CUB-895 is the high resolution option in the Microvitec 14" range. This model offers 895 x 585 addressable pixels and is ideal for those applications requiring very high clarity and precise colour reproduction.

1441MS (Metal Case)	£ 440.00
1441LS (Structural Foam)	£ 450.00
1446LI (IBM/Programmed Rom)	£ 495.00

PHILIPS

CT2007 TV/MON

The Philips CT2007 is a 14" colour TV receiver/monitor with inputs for both R.G.B. and C.V.B.S. as well as audio. Bandwidth is 20MHz and the display is standard resolution.

CT2007	£ 229.00
--------	----------

SANYO

CRT70

The Sanyo CRT70 is a 14" high resolution monitor in an attractive silver alloy finish. Resolution is 800 pixels and the input is R.G.B.

CRT70	£ 499.00
-------	----------