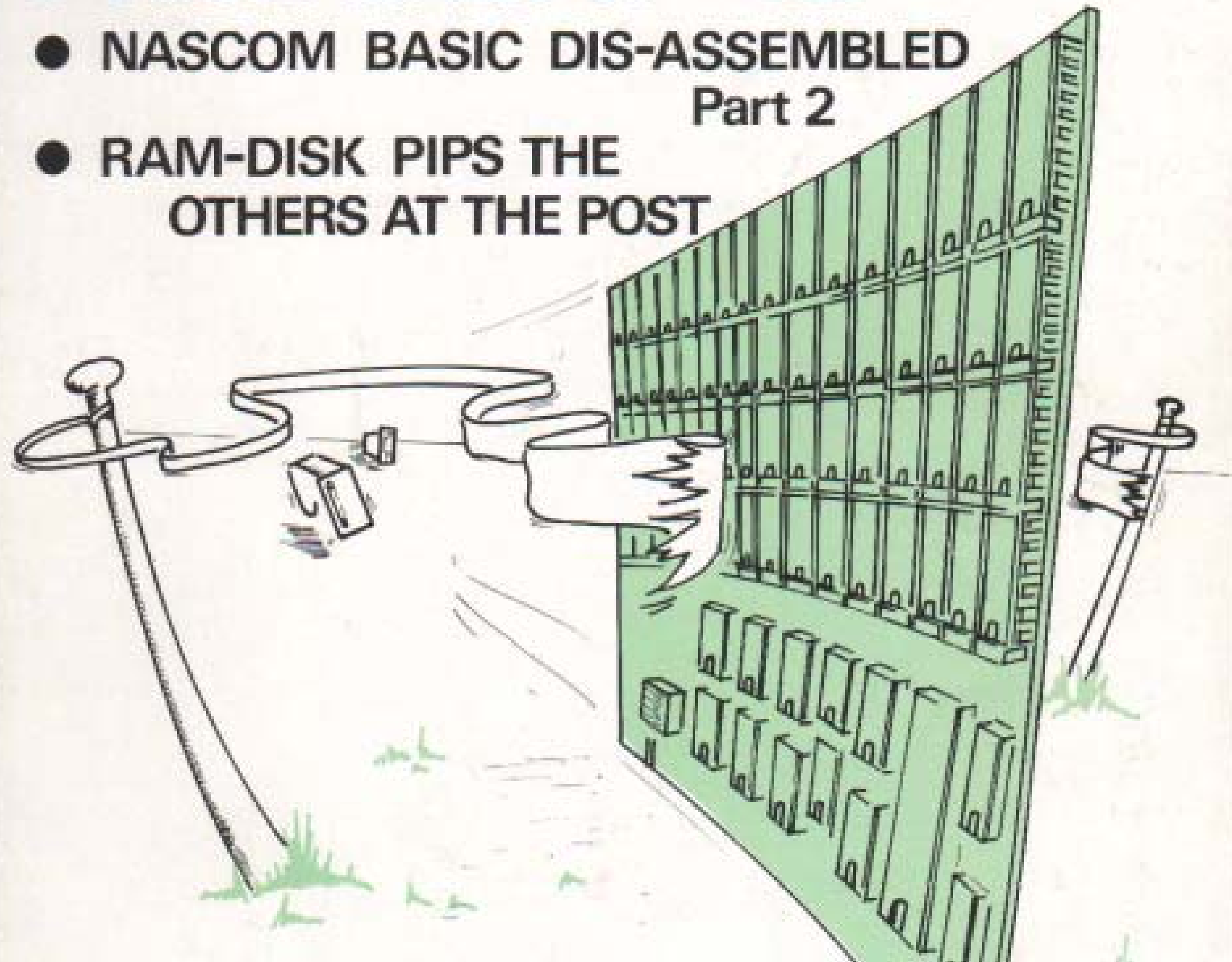
80-BUS NEWS

JULY – AUGUST 1983 VOL.2 ISSUE 4

- COLOUR BOARDS COMPARED
- NASCOM BASIC DIS-ASSEMBLED Part 2
- RAM-DISK PIPS THE OTHERS AT THE POST



£1.50

The Magazine for NASCOM & GEMINI USERS

July – August 1983 **80-BUS NEWS** Volume 2. Issue 4.

CONTENTS

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

SUBSCRIPTIONS

Annual Rates (6 issues)	UK	£9	Rest of World Surface	£12
	Europe	£12	Rest of World Air Mail	£20

Subscriptions to 'Subscriptions' at the address below.

EDITORIAL

Editor : Paul Greenhalgh   Associate Editor : David Hunt

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

ADVERTISING

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

ADDRESS: 80-BUS News,
Interface Data Ltd.,
Woodside Road,
Amersham, Bucks, HP6 5EW.

Letters to the Editor

Re. COMPAS Review

I saw that COMPAS was reviewed by Doctor Dark (your graduate in Voodoo medicine?) in vol. 2 iss. 3, and although I am always pleased to see our products mentioned in articles, in this case I was less excited, since the product reviewed is COMPAS version 1, and not version 2, which is a far better package.

As opposed to COMPAS 1 and Hisoft Pascal, COMPAS 2 is a superset of Standard Pascal, i.e. it supports all features of the standard language, plus many more. The extensions that are most noteworthy are dynamic strings, random access files, structured constants, include files and, in the latest version (called 2.20) also overlays. All extensions are carefully thought out, and to a large extent are compatible with other well known Pascal systems, such as UCSD Pascal and Pascal MT+.

Programs written in COMPAS 2 can be distributed with no royalties payable. We realise that the terms set forward in the COMPAS 1 license agreement were a bit harsh, not to mention the fact that they were impossible to implement. Therefore there are no royalties on programs written using COMPAS 2.

I feel that COMPAS 2 is the ideal package for anyone who wishes to learn about Pascal since it supports the full standard, it is also ideal for experienced programmers since the extensions make this a very powerful implementation of Pascal indeed.
Anders Hejlsberg, Polydata, Denmark.

MBASIC on Disk

I read with great interest the article by Dr. Coker in the last issue of 80-BUS News and it is obvious that he has looked at his BASIC in great detail but has not done sufficient programming in it to find the error in his use of the substitute REM statement. In particular he will find that while you can use the apostrophe (27H) either at the start of the line or in the middle, if used in this latter position it is necessary to include a colon to seperate it from the previous command or that program could crash as BASIC will not know where to stop interpreting and treat the remainder of the line as a REM statement.

His second comment that MBASIC is regularly updated may have been correct but reading the trade press one is lead to believe that MBASIC-80 is a casualty of the CP/M versus MSDOS war, and that Microsoft will not be supporting it for CP/M-80 in the future.

Since the 80-BUS is essentially a Z80 bus which runs CP/M and we are not too interested in "foreign" operating systems such as MSDOS, what is needed for the next 10 years is a UK supported BASIC, starting with the core instructions of MBASIC and having extensions to cover the following:

(a) Use of full Z80 instruction set

(b) Include additional commands such as cursor addressing, clearing screen, reverse video and other terminal attributes

(c) Enabling the BASIC interpreter or compiler to address more than one 64K page of memory where it is available

The principle reason why so called 16 bit machines have gained in popularity is because most of the software available to run under CP/M uses only the 8080 instruction set and not the enhanced instruction set of the Z80 which we all know and love!

It is highly unlikely that I will be upgrading to the Intel series of 8086 or 8088 chips especially as Zilog have announced the Z800 series where the specification reads as if it were tailor made for the 80-BUS. I feel that it is a sufficiently big chip to last me for the next 10 years.

We all know that BASIC is the language that everybody loves to hate but everybody is using it, there is so much software written in it that to abandon it would be very short sighted. Couple to this the ability to run in interpreted mode until the program works and then compile it to run faster and to protect your code and you have a combination of software for developing applications programs that work.

Having said all this in praise of the humble Z80 chip and CP/M systems all that is needed is for some volunteer to come forward to write this program. All serious contenders please call me on 0243 825132.
John Stuckey, Bognor Regis, W. Sussex.

Non-communicating Dodos

It appears that most of you (at 80-BUS News) feel rather upset that Dodos like myself do not communicate more regularly.

I'm sorry about this, but I would point out that a major characteristic of a dodo is its lack of communicable ideas.

Two possible reasons for your upset, it seems to me, are:-

i) you feel that the ratio of contributors to subscribers is far too low
ii) you wonder whether there is any real demand for the mag.

Now it seems to me that, if you take these fears seriously, the solution is easy: cut down the number of subscribers. I have no idea what the number is, but it must be reasonable for your contributors to wonder what we're all doing (fiddling with our appendages Chris, but we call it 'getting on with life').

I think that this approach is silly, and that a large number of subscribers, at £2.25 an issue (£9/6 issues per year = £2.25 if 6=4 as it did last year) is something to be proud of, but a way of implementing it, if you really want to, is to return the £9 if it is not accompanied by usable material for the mag.

If you think that anyone but me will read articles on what I do with my Nascom 2 / GM805 with DCSDOS II / Pluto (baby) / Imp then you are mistaken, as it doesn't work anyway.

If you think that all we dodos are sitting on a goldmine of ideas, you fail to recognise that YOU WROTE THE GOLDMINE!!!

As long as I can keep buying it, I will read it!
Bill Radcliffe, Teddington, Middx.

The Final Word

I wrote to you on 14/4/83 to ask why there were only 4 issues of 80-BUS News 1982 when you clearly stated there would be 6. I have not yet had a reply.

I enclose a cheque for £9 for a years subscription in the final hope that things will improve; although performance so far is not good. There can be few excuses left.
A. Brown, Didcot, Oxon.

Colour and your computer – the choices.

by D.R. Hunt

The Colour Cards.

Some time ago I was asked to give a talk at the East Grinstead Computer Club. I hope a good evening was had by all despite the rambling nature of my chat. I know I enjoyed myself, apart from the fundamental mistake that when I agreed to go, I had firmly fixed in my mind that East Grinstead was reasonably local, being somewhere north of Finchley, instead of being half way to Eastbourne. One of the things to come out of the evening was what was my opinion of all the colour cards around? When we totted them up we found there were seven:

The Winchester Technology Colour card
The Pluto Colour card
The Nascom AVC
The Climax colour card
The R & E W colour card
The Edinburgh University colour card
A N other colour card.

There were definitely seven, but for the life of me, I can not now recall whose the last one was. My opinions weren't much use at that time either, as I had only gained experience of one of the cards, and hadn't even seen most of them in action.

Since then, I have to some degree rectified my lack of experience of colour cards, having seen all but one, and played with most, three have occupied some considerable time over the last couple of months. When I wrote a preliminary note about the Nascom colour card some issues back, I was sceptical as to the uses a colour card could be put. I'm afraid I still remain unconvinced, and I think it would take a radical change in my thinking and uses of my computer to persuade me that any colour card should be given a home in my computer box.

Now lets have a quick look at each card. The Winchester Technology Colour 8" x 8" memory mapped card was by far the earliest and used the Mullard Teletext controller chip set. This made it fully Prestel and Ceefax compatible, but not really high resolution. The card is no longer manufactured, which is a pity with the growing interest in Prestel and Ceefax compatibility, it could be a big sales point these days to have a range of computers which are Prestel and Ceefax compatible.

Next to arrive was the Pluto card in its various forms. Again an 8" x 8" card but this time port addressed. Possibly the fastest and most powerful, certainly the most expensive and the highest resolution. This card uses an on board 16 bit (8088) processor and carries its own bus structure for even further, mind blowing expansion.

After a long wait came the Nascom AVC, an 8" x 10" card built round a 6845 video controller chip. It uses page mode memory mapping to address it, and features comfortably high resolution and a nice range of software bolt-ons to go with it.

Then, quite recently, came the Climax Computers card. An 8" x 8" card built round the Thompson Video controller chip and port addressed. Lower in resolution than either the Pluto or the Nascom, but with limited grey scaling and some of the best explained software primitives I have yet to see.

R & E W, the magazine sponsored by Ambit International, the components people, entered the field by publishing an article on a colour card which was then made available as a kit. This one was either a 10" or 12" x 8" card and was port addressed using the TI colour video controller. This controller features 'sprites', which are preformed patterns which can be manipulated with incredible speed once set up. I have yet to see one of these cards working, but I have heard dark mutterings from various directions about the damn thing not doing what was expected of it.

Some time ago a sample card was shipped down from Edinburgh University for a group of dealers to offer their opinions. Very few details were supplied with it, except that it was a port addressed 8" x 8" card, the resolution was something like 350 x 256, but its most interesting feature was that the demo displayed amongst other things a test card with a 16 step grey scale wedge. Some time during the day the card (which was after all a prototype) crashed and at the end of the day it was shipped back to Edinburgh. That is the last that has been heard of it - a pity, a could be a contender in the colour card stakes.

As I said earlier, I can not recall anything about the A N Other card which was discussed at the East Grinstead meet.

I have also seen the circuits and chat about a Dutch designed colour card which was published in the Dutch equivalent of 'Nascom News'. This one is similar to the Climax card except it uses the 256 x 512 version of the video controller and has an AY-3-8910 sound chip on it as well.

THE COMPARATIVE REVIEW.

Of those listed above only three are in serious production, the Pluto, the Nascom and the Climax. I have been having fun (or tearing what's left of my hair out) with these. I propose to review the features of each of the cards under different subheadings. Where appropriate I shall digress.

THE MONITORS.

For the first digression, and before getting on with the review, let's have a have a look at the display requirements. Colour monitors are not cheap, and the domestic colour TV appears at first sight to a be viable alternative. Unfortunately, the easiest entry into the domestic TV is the RF input, which just happens to be the most unsatisfactory. Some TV's notably of European origin, Normende, Grundig, etc, have composite PAL inputs on the back for connecting Video Cassette recorders, which would be satisfactory with the Climax card. Many TV's have RGB inputs for the inclusion of Teletext panels if you know where to look, but it is highly dangerous to assume that these would make a safe connection as most colour TV's use chassis live to one side of the mains, and connection here is a quick way of taking a ride to the mortuary. So unless the TV has the right connections on the back, or unless you are thoroughly aware of what you are about, DO NOT make connections inside the domestic TV set.

The Nascom and Pluto cards do not have RF outputs on the grounds that the resolution is too high for the domestic TV, certainly true in the case of the Pluto, perahps arguable in the case of the Nascom. The Climax card is fitted with a modulator for connection into a TV. Even so, the results from a TV are very much inferior to the results from even a modest colour monitor.

So to the choice of monitors. These are normally available in three resolutions and with two types of input, analogue and logic. Resolution first. Not all manufacturers make all types, but the information can usually be extracted from the data sheets, albeit, sometimes disguised as something else. Resolution is usually measured in vertical lines. Not to be confused with the number of horizontal lines on the display. Vertical line resolution is taken as the maximum total number of vertical black/white lines the display is capable of resolving across its face, and is a function of the fineness of the shadow mask of the tube and tube size. The bigger the tube, the more lines it should resolve. Some manufacturers quote the vertical resolution at the centre of the tube, meaning that the total resolution across the face of the tube would be this figure if the tube was good enough. It is reasonable to expect some roll off towards the edges, but not too drastic roll off. Checking the deflection angle can be of some help here. 90 degree tubes do not roll off so much at the edges as 110 degree tubes. Most monitors are fitted with 90 degree tubes for this reason.

Don't fall for the old trick of looking for a high input bandwidth, all this means is that the electronics are up to it, but says nothing about the tube characteristics. 24MHz bandwidth might imply that the monitor is capable of 800 line resolution, but what is to stop the manufacturer fitting a low res. tube, whereas, a quoted resolution of 800 lines means that both the electronics and the tube must be capable of the job.

A low res. monitor would typically have a 12" or 14" tube and resolve 400 lines at the centre of the display (less at the edges, although this shouldn't be too much less, perhaps 350 lines). This resolution is about the same as a 14" colour TV (and is probably where the tube came from in the first place) and is adequate for the Climax and just about for the Nascom card. These are typically in the £200 to £300 range.

The next are the medium res. monitors with about 600 lines resolution for a 12" or 14" tube. Entirely suitable for the Nascom card in the normal eight colour mode, and the Pluto at a push. Typical prices are in the £300 to £400 range. The higher price is dictated by the purpose made tube with its finer shadow mask. It is highly probable that the electronics are identical to the equivalent low res. version. This is certainly true between the KAGA Vision - I and Vision - II where the difference in price is solely down to the tube.

Lastly are the real high res. monitors with 800 line or better resolution. These are very costly, between £450 and £2000 depending upon the size of the tube, in fact the major part of the cost is the purpose made tube, so tube guarantees and insurance are well worth exploring.

Monitors will most probably have one of two types of input, logic, where the input is either a colour at full saturation or off (black), or analogue where the intensity of the colour is related to the level of the input voltage. Some monitors are switchable in this respect. Logic inputs are suitable for the Nascom card and the Pluto card (if expansion to the Pluto Palette is not envisaged). The Climax requires an analogue input as does the Pluto Palette.

Various monitors and an old Philips TV were used with the colour cards. Those used were the:

KAGA Vision - I	12" tube	400 line resolution	analogue input
KAGA Vision - III	12" tube	800 line resolution	analogue input
Microvitec Cub	14" tube	400 line resolution	logic input
Luxor	14" tube	800 line resolution	logic input
Luxor	14" tube	800 line resolution	logic/analogue
Philips portable TV	18" tube	600? line resolution	RF input

All were found to satisfactory within their specifications (except the TV, which was awful). The analogue input on the Luxor was found to be quite linear and could double as a TV repeater monitor. The KAGA inputs were not quoted as being analogue, but were found to be reasonably linear although not as good as the Luxor.

SPECIFICATIONS.

The Pluto Card.

The Pluto card is available in a number of different versions and with various options, from the Baby Pluto to the Pluto Palette. The Pluto card I shall concern myself with is the standard Pluto with the 8MHz option fitted. The optional extended command ROM was not fitted. The cost of this version is £450.00. The card is an 8" x 8" card and densely packed. 196K of 4164 RAM is fitted, 192K is used for display. The basic resolution is 640 x 288 in two pages and eight colours. The two pages may be addressed at will, and the use of two pages allows one page to be displayed whilst manipulation is taking place on the other. Switching then allows the two display pages to be swopped instantly making for very fast apparent frame changes. (A Baby Pluto is the same but only has half as much RAM and a single page.)

The Card is port driven, control codes being sent to three ports addressed in page CXH. The port addresses may be redefined. Output is provided in TTL compatible RGB with mixed or separate syncs only, the resolution of the Pluto precludes the use of an RF output.

Most, but not all the RAM is assigned to the physical display. Areas are set aside for the storage of character sets and shapes as desired. These areas are setup dynamically, and are assigned as pages 0 - 255. Pages 0 and 1 are the display pages, setting software switches allows access to any page. The card uses an 8088 16 bit processor for control

Single and multibyte commands are passed to the on-board 8088 and this processor causes the function to take effect. The functions provided in the command ROM as supplied are adequate, but the more complex functions such as circle drawing and filling and complex fills are only provided with the extended command ROM available as an option. Little external software is required to drive the Pluto card.

The Nascom Card.

The Nascom AVC is a 10" x 8" card and is supplied complete with a NAS-DOS disk of add-ons to the Nascom Basic and a very extensive manual. The card is the cheapest of the trio at £149.95.

The Nascom AVC has page moded RAM and uses three overlayed RAM planes each of 16K, which could provide a maximum resolution of 512 x 256. However, as this RAM is also addressed during line and frame blanking the display is actually 392 x 256 pixels. The RAM areas being page mode can be addressed at the same address and as supplied are set to 8000H although these addresses may be changed to suit at any 16K boundary. As addressing is by page mode, no space is normally used in the processor RAM plane, but unlike the Pluto and Climax cards, this method allows direct access to the video RAM. Three ports are also used for control of AVC via the 6845 video controller allowing non-standard video formats to be set up (not advised unless you know exactly how the 6845 works), current cursor positions, etc.

1 volt video outputs for Red, Green and Blue are provided with both separate and mixed syncs. TTL output levels are also available on the card. An external connector allows the Nascom PIO to be connected to an area of logic which can deselect the colour outputs completely or select different colours to the different outputs, for instance, the red output may be turned off or may be directed to either the blue or green outputs, and likewise for the other outputs. This allows very high speed colour switching and use of this feature allows for limited animation graphics. It's a pity that horizontal and vertical syncs are not provided as separate ouputs from the card as we had to build a primitive RC sync separator to feed frame pulses back into the Nascom

to allow this feature to be used to the best advantage. A further possible ommission is that room was not found for a binary adder on the board as this could then allow very high speed colour 'rotates' to take place a la BBC computer. Perhaps it could be argued that the provision of this feature would be more of a gimmick than useful.

Another useful feature is the provision for placing two RAM planes 'side by side', allowing the use of two colours in a very high res mode of 784 x 256. Use is made of this feature when the AVC is used as a terminal with Nascom's CP/M to allow the card to be used as an 80 x 25 text display with high res. graphics thrown in. The only snag with this mode is that the 80 x 25 display is effectively bit mapped. This means that the characters have to be looked up in a table and displayed as graphics. Fine so far, but when it comes to scrolling, it means that 32K of video RAM has to be scrolled to remap the characters in their new positions. This makes scrolling rather slow.

Port controlled provision is provided for the routing of the normal Nascom video through the AVC outputs, allowing the use of one common display monitor. The 48 x 16 normal Nascom display is suited to a colour monitor. But this approach gets a little complicated when debugging programs as it is often useful to have the normal Nascom display displaying say, counters, etc, within a program whilst the colour display is constructing a picture.

The Climax Card.

The Climax card is available in two versions, Version A provides modulated RF output using a high quality wide bandwidth modulator and PAL composite video with mixed syncs at £199.00. Version B provides modulated RF and composite PAL as before, and also analogue RGB outputs at £220.00. In neither case are separate syncs provided and we had to extract a separate composite sync signal from elsewhere on the card to make the Kaga montiors lock. Note that the outputs are analogue as the Climax card is capable of full and half saturation colours. Be careful when selecting a colour monitor for use with the Climax, as most cheaper monitors are logic inputs only, and the full benefit of the half saturation colours will be lost, displaying either black or full saturation colour. The only moderately priced monitor found to be suitable for the Climax RGB drive is the KAGA Vision-I monitor which, whilst not quoted as being analogue, worked extremely well.

The Climax uses 64K by four bits of RAM and is under command of the Thompson colour controller chip which gives 256 x 256 resolution in eight fully saturated colours and eight half saturation colours. The screen display is totally square, leaving black margins at either side of the screen.

The card is port addressed using seventeen ports from COH to DOH inclusive. This makes the card greedy of port allocations, but exceptionally easy to understand. Single and multibyte instructions are passed to the various controller registers to perform a number of different functions directly, but software generation of some of the more complex functions is required.

SUITABILITY.

The Pluto Card.

The Pluto card is equally suited to either Nascom or Gemini machines. The card is provided with the NASIO decode signal, although this is only partially decoded. No system specific software is included, so there is no problem from the software point of view. Being totally port addressed no provision has to be made within the memory map.

The Nascom Card.

The Nascom card is most suited to the Nascom as the software is written with use under NAS-DOS in mind. Nascom tape versions are available, and I understand some form of software is to be made (or is) available for use under CP/M. The software provided (on NAS-DOS disk) is exchangeable to tape on request is in the form of an extension to the Nascom Basic, and provides a number of useful commands. Being page mode memory mapped no provision need be made within the memory map. The NASIO signal is provided.

The Climax Card.

The Climax card is equally suited to the Nascom or Gemini but the software is available only on disk in Gemini formats and aimed at use with the Microsoft MBASIC and hence, implies the use of CP/M. Full listings of the colour driver primitives are supplied and it wouldn't be too difficult (more tedious) to convert them for use with Nascom Basic or 'stand alone' machine code subroutines. Being port addressed no provision need be made within the memory map. The NASIO decode signal is provided.

DOCUMENTATION.

The Pluto Card.

A slim, ring bound volume is provided with the Pluto, and introduces the features of the Pluto card, fitting to the system, an explanation of the preprogrammed board functions and a couple of demo programs. Although thorough and well written, I found something indefinably wrong with this manual, I found it extremely difficult to understand and yet it was written clearly enough. It took several goes through the book to grasp an understanding of the Pluto card from the manual. References to command types were in alphabetical order which didn't help as I wasted considerable time searching for control codes by the obscure alphabetical labels given to them instead of being able to spot them instantly. In the end I rewrote the command lists in function order to make them understandable.

No circuit diagrams were supplied and the technical description was brief. I found it helpful to draw a block diagram of the card from what description there was and what I discovered in use.

On a couple of occasions I resorted to phoneing IO to clarify points, on each occasion I was answered by a telephone answering machine. Now as a rule I hate telephone answering machines, but they usually give you the option of leaving a message to ring back or saying something rude about answering machines. This telephone answering machine was exactly as described, it answered the phone, told me no-one was around and then rang off. Not very helpful at all!!

In the end I had so much trouble thumbing through the manual to find things that I decided to write a set of straight forward driver routines linkable from MBASIC in the same manner as the Climax routines. I got some way with this, but as yet have not finished them.

The Nascom Card.

In contrast to the IO manual, the Nascom manual is a fat tome, bound in a ring folder of Lucas green. Everything was explained in exhaustive detail and was very clearly written. All the software functions were well explained with numerous examples. Circuit diagrams were provided and a fairly thorough circuit description was in the manual. As the Nascom card has little computing power of its own, being confined to the line and frame generation and memory mapping of the 6845 video controller chip, all functions such as line drawing are handled in software. These functions are normally called from the Nascom Basic, using the SET function to allow the Basic functions to be extended. All functions were treated in detail. One oddity was noticed, there was no circle drawing function, but a function for drawing regular polygons with n sides was provided, a 255 sided polygon was found to produce an entirely satisfactory circle.

Unlike the Pluto card, the Nascom card requires extensive software to drive it and a disk or tape is supplied with a number of colour driver primitives, the manual explains these are well, function for function. Sadly Nascom do not see fit to supply the source listings of these colour driver primitives. A pity as some of them are extremely fast and efficient. Without the source it would be difficult to rewrite these to allow the card to be used with systems not based around the the Nascom 8K Basic.

The Climax Card.

The manual supplied with the Climax falls midway between the Pluto and Nascom manuals. Certainly fatter than the Pluto manual, it was single sided photostated sheets with no binding. The manual started with the get you going bit of how to connect the card, followed shortly by a simple Basic program to test the correct working of the card. Just the sort of order I like things, read a bit of the manual, plug it in and see if it goes!! This was followed by a fairly good and certainly adequate description of the workings of the card, but no circuit diagrams. The description was followed by a very well written section on the uses of the registers of the Thompson chip. This is much more understandable than the manufacturers descriptions I have read in their data sheets. To finish the description of the registers a short extract of the Thompson data sheet was supplied to clarify timing and register addressing. So far so good.

As explained previously, the Thompson chip is capable of point to point line drawing, plotting points, colour changing, and a lot of other simple things (the chip has an internal character set for instance), however, to do the cleverer things like drawing circles requires external software. The manual follows the chip description with a description of the software drivers and a complete source listing. Now when I comment source files for publication in this rag, I often think to myself that the amount of comment included is overkill. Climax seem to work on the same principle, the comment is certainly extensive, and make the routines fully understandable. The colour driver primitives are intended to link automatically with MBASIC, but because they are written entirely as subroutines, they can easily be modified into stand alone machine code routines, or linked with some other high level language.

Whilst on the subject of the Climax driver routines, I must say they had their drawbacks. The source listing is printed in the manual, but is also available on disk with some demo material. The snag was that I didn't have the disk so I had to type in about 40K of source and then hope I had got it right. That was 40K without much of the comment - when I finally did receive the disk, I found the source listing was the best part of 100K. Still it was good typing practice and even more incredible, I only made two mistakes.

USING THE CARDS.

The Pluto Card.

The Pluto card was certainly fast and powerful. My main complaint with the card was not with the card itself but with the documentation which as mentioned previously, I found awkward. Fairly simple programs could be written in Basic to provide the necessary primitives to draw lines etc, and line plotting was very fast indeed. The resolution was all it was supposed to be, the 640 x 288 format over filling the screen of the monitors used, and resolution patterns were just resolvable on the best monitors. I found the lack of circle drawing and fill routines a nuisance, but I suppose I could have fitted the extended command ROM (£50.00) had I wished, instead I pinched some of the machine code primitives from the Climax card and used those instead. The page memory system of the Pluto was very useful, it allowed preset shapes to be set up and used quickly, this is most useful for character sets which may be created in several fonts for use. The use of text in the 80 column mode was well displayed, but seemed something of a waste of a very expensive monitor if 80 column display were the only use of the card.

The Nascom Card.

The Nascom card plugged straight into a Nascom system and refused to work. This was traced to not having read the manual as a small mod. is required to the page modeing of the RAM card to allow the AVC to work without corrupting the existing system RAM. Having read the manual and fixed the RAM card the AVC was well behaved and proved easy to use. The software supplied linked into the Nascom Basic transparently and provided all the expected functions. Overall the speed was good, not as good as the Pluto by a long shot, but adequately fast for most purposes. The main drawback as far as speeding up the card was not being able to make immediate use of the Nascom machine code primitives. Access to the primitives is via a jump table at the top of the workspace and although entirely possible by careful reading of the manual, the absence of the source listing made life difficult as a lot of preset registers required setting up before any primitive could be called.

Lots of fun was had using the port addressed colour changing facilities and a small binary adder was built to provide colour 'rotation', marvellous for making high speed animated wallpaper effects and baffling people who could not understand how the Nascom card did it. All round I don't have any moans about the way the Nascom AVC worked.

The Climax Card.

The Climax plugged in and worked in a Gemini system straight away. The problems over the disk primitives was solved by spending an evening typing them in, and taking advantage of having to retype them by arranging them for easy assembly into stand alone machine code primitives if required. It seems that the disk of primitives are supplied as an optional extra at £15.00 + VAT, so if you value about four hours typing at more than £15.00, you'd better buy the primitives. The primitives automatically linked with both versions 5.12 and 5.2 of MBASIC with no trouble and provided most of the functions I required. Two notable ommissions I have supplied as source listings elsewhere in this rag, and Climax users may tag these straight on the end of the primitives as supplied, but note, the whole lot must be reassembled otherwise the relocator tables won't be present.

The Climax was notable for its speed. Some of the functions were achieved (visually) instantly, the box drawing and filling routines were very impressive indeed. The method of sending instructions from Basic to the primitives used the Basic CALL function and was very efficient with one nasty snag. If the label called was misspelt or otherwise missing, then MBASIC would return to the operating system (being effectively a call to 0000H) thus losing the whole program. I soon learnt to save ALL programs to disk before running them!!!

Overall I found the Climax card very fast indeed and very effective with no vices. Nothing I did locked it up, either by design or unintentionally. The documentation, whilst not as thorough as the Nascom was adequate and the provision of the primitive source listings was most helpful.

THE SUMMARY.

I have tried to examine the salient points of all three cards, and in the process have glossed over a lot. I suggest that any potential purchaser try to get hold of the manuals for all three before making a choice as there is a lot I have missed here and there may be things which are of some importance to the potential user that I haven't mentioned at all. As to how I rated the cards:

Pluto.

The worst buy, being expensive and poorly documented. Most suited to specialist uses and finding its way into broadcast standard equipment. Very definitely not a cheap way of playing colour space invaders.

Nascom.

The best buy, the cheapest, the best documented and with very adequate resolution. A little slow in some respects and pity about the 10" x 8" card format, but in all other respects fine.

Climax.

The one in the middle nearly equal with the Nascom but let down by the higher price for slightly lower resolution. The square display leaves margins on the screen but well worth considering from the speed and provision of the primitive source listings alone.

```
MV256 Relocating Graphics Drivers     M-80     20 Jun 1983     23:54     PAGE   1
Disk read/write routines for MV256

                         subttl Disk read/write routines for MV256

                         ; Using a compressed format

                         ; CP/M Disk equates
0005                     BDOS    EQU     0005H
0080                     TBUF    EQU     80H
0009                     PRS     EQU     9
000F                     OPNFL   EQU     15
0010                     CLSFL   EQU     16
0013                     DELFL   EQU     19
0014                     RDFL    EQU     20
0015                     WRFL    EQU     21
0016                     MKFL    EQU     22
001A                     SETDMA  EQU     26

                         ; ROUTINE GSAVE
                         ; Save the graphics screen to disk
                         ; GSAVE(N$)
0A59     CD 0B8C         GS:     CALL    FCB1          ; Put the name
                                 R                     ;      in the FCB
0A5C     11 0C68                 LD      DE,FCB        ; Close file in
                                 R                     ;      case it's open
0A5F     0E 10                   LD      C,CLSFL
0A61     CD 0005                 CALL    BDOS
0A64     11 0C68                 LD      DE,FCB        ; Delete file in case
                                 R                     ;      it already exists
0A67     0E 13                   LD      C,DELFL
0A69     CD 0005                 CALL    BDOS
0A6C     11 0C68                 LD      DE,FCB        ; Open a file with
                                 R                     ;      the new name
0A6F     0E 16                   LD      C,MKFL
0A71     CD 0005                 CALL    BDOS
0A74     3C                      INC     A             ; Test for successful
0A75     20 09                   JR      NZ,GS1        ;      open
0A77     11 0BF8                 LD      DE,GMESS1     ; Failed to open,
                                 R                     ;      so say so
0A7A     0E 09                   LD      C,PRS
0A7C     CD 0005                 CALL    BDOS
0A7F     C9                      RET

                         ; Main load down loop, get screen and write it out.
0A80     21 0000         GS1:    LD      HL,0          ; H=Y L=X, point to 0,0
0A83     CD 0ABD         GS2:    CALL    LGS1          ; Get a buffer full
                                 R                     ;      (128 bytes)
0A86     E5                      PUSH    HL            ; Save pointer to X,Y
0A87     F5                      PUSH    AF            ; Save the flags
0A88     11 0080                 LD      DE,TBUF       ; Write it out
0A8B     0E 1A                   LD      C,SETDMA
0A8D     CD 0005                 CALL    BDOS
0A90     11 0C68                 LD      DE,FCB
                                 R
```

```
0A93    0E 15               LD      C,WRFL
0A95    CD 0005             CALL    BDOS
0A98    B7                  OR      A               ; Test for successful
0A99    28 0B               JR      Z,GS3           ;      write
0A9B    11 0C1D             LD      DE,GMESS2       ; Write failed,
                            R                       ;      so say so
0A9E    0E 09               LD      C,PRS
0AA0    CD 0005             CALL    BDOS
0AA3    E1                  POP     HL              ; Lose AF and HL
0AA4    F1                  POP     AF              ;      off the stack
0AA5    C9                  RET
0AA6    F1          GS3:    POP     AF
0AA7    E1                  POP     HL
0AA8    20 D9               JR      NZ,GS2          ; All done? No, go
                                                    ;      round again
                    ; Now close the file and go home
0AAA    11 0C68             LD      DE,FCB
                            R
0AAD    0E 10               LD      C,CLSFL
0AAF    CD 0005             CALL    BDOS
0AB2    3C                  INC     A               ; Test for successful
0AB3    C0                  RET     NZ              ;      close
0AB4    11 0C39             LD      DE,GMESS3       ; Close failed, so say so
                            R
0AB7    0E 09               LD      C,PRS
0AB9    CD 0005             CALL    BDOS
0ABC    C9                  RET

                    ; Get a buffer full into TBUF
                    ; The four MSBs of each byte represent 1 to 16 pixels
                    ;           of the same colour.
                    ; The four LSBs represent the colour.
0ABD    11 0080     LGS1:   LD      DE,TBUF         ; Point to temporary
0AC0    06 80               LD      B,128           ;      output buffer
0AC2    AF                  XOR     A               ; Clear out MSBs
0AC3    D3 CA               OUT     (YCM),A         ;      of X and Y registers
0AC5    D3 C8               OUT     (XCM),A

0AC7    D5          LGS2:   PUSH    DE              ; Save the TBUF pointer
0AC8    1E 00               LD      E,0             ; Set the pixel counter
0ACA    CD 0AF3             CALL    GCOL            ; Get a pixel
                            R
0ACD    57                  LD      D,A             ; Save it
0ACE    23          LGS3:   INC     HL              ; Point to next pixel
0ACF    7C                  LD      A,H             ; Test for end of screen
0AD0    B5                  OR      L
0AD1    28 0D               JR      Z,LGS4          ; Is end, so write it out
0AD3    CD 0AF3             CALL    GCOL            ; Get the next pixel
                            R
0AD6    BA                  CP      D               ; Is it the same
0AD7    20 07               JR      NZ,LGS4         ; No, write out last colour
0AD9    1C                  INC     E               ; Yes, so inc pixel counter
```



```
0ADA    3E 10               LD      A,16            ; 16 pixels yet?
0ADC    BB                  CP      E
0ADD    20 EF               JR      NZ,LGS3         ; No, so go and get the next
0ADF    1D                  DEC     E               ; Fudge the pixel count
0AE0    7B          LGS4:   LD      A,E             ; Put the pixel count
0AE1    CB 27               SLA     A               ;      in MSBs of A
0AE3    CB 27               SLA     A
0AE5    CB 27               SLA     A
0AE7    CB 27               SLA     A
0AE9    82                  ADD     A,D             ; Add the colour to it
0AEA    D1                  POP     DE              ; Get the TBUF pointer back
0AEB    12                  LD      (DE),A          ; Save the byte
0AEC    13                  INC     DE              ; Point to next in TBUF
0AED    7C                  LD      A,H             ; Test for end of screen
0AEE    B5                  OR      L
0AEF    C8                  RET     Z               ; Go home write out anyway
0AF0    10 D5               DJNZ    LGS2            ; More required in TBUF
0AF2    C9                  RET                     ; TBUF full so write it out

                    ; Get a pixel
0AF3    CD 08E0     GCOL:   CALL    READY
                            R
0AF6    7C                  LD      A,H             ; Send H & L to Y & X
0AF7    D3 CB               OUT     (YCL),A
0AF9    7D                  LD      A,L
0AFA    D3 C9               OUT     (XCL),A
0AFC    3E 0F               LD      A,0FH           ; Send GET PIXEL command
0AFE    D3 C0               OUT     (CMD),A
0B00    CD 08E0             CALL    READY           ; Wait till done
                            R
0B03    DB D0               IN      A,(GETCOL)      ; Get the pixel
0B05    E6 0F               AND     0FH             ; Get rid of MSBs
0B07    C9                  RET

                    ; ROUTINE GLOAD
                    ; Send graphics file to screen
                    ; GLOAD(M$)
0B08    CD 0B8C     GL:     CALL    FCB1            ; Get name into the FCB
                            R
0B0B    11 0C68             LD      DE,FCB          ; Open the file
                            R
0B0E    0E 0F               LD      C,OPNFL
0B10    CD 0005             CALL    BDOS
0B13    3C                  INC     A               ; Test successful open
0B14    20 09               JR      NZ,GL1
0B16    11 0C51             LD      DE,GMESS4       ; Open failed, so say so
                            R
0B19    0E 09               LD      C,PRS
0B1B    CD 0005             CALL    BDOS
0B1E    C9                  RET
```

```
                        ; Main load down loop
0B1F    DB C2           GL1:    IN      A,(CNTL2)       ; Get the current line
0B21    F5                      PUSH    AF              ;      style and save it
0B22    AF                      XOR     A               ; Set continuous style
0B23    D3 C2                   OUT     (CNTL2),A
0B25    21 0000                 LD      HL,0            ; Set HL to (X,Y) = 0,0
0B28    E5              GL2:    PUSH    HL              ; Save the screen pointer
0B29    11 0080                 LD      DE,TBUF         ; Set DMA to TBUF start
0B2C    0E 1A                   LD      C,SETDMA
0B2E    CD 0005                 CALL    BDOS
0B31    11 0C68                 LD      DE,FCB  ; Get a record into TBUF
                                R
0B34    0E 14                   LD      C,RDFL
0B36    CD 0005                 CALL    BDOS
0B39    E1                      POP     HL              ; Get screen pointer back
0B3A    B7                      OR      A               ; Test for end of file
0B3B    20 05                   JR      NZ,GL3          ; Yes, skip screen write
0B3D    CD 0B49                 CALL    LGL1            ; Write TBUF to screen
                                R
0B40    20 E6                   JR      NZ,GL2          ; Go and get some more
0B42    F1              GL3:    POP     AF              ; Get the line style back
0B43    CD 08E0                 CALL    READY
                                R
0B46    D3 C2                   OUT     (CNTL2),A
0B48    C9                      RET                     ; Go home

                        ; Load a buffer full to the screen
                        ; The four MSBs of each byte represent 1 to 16 points
                        ;  of the same colour. The four LSBs rep. the colour.
0B49    06 80           LGL1:   LD      B,128           ; Set count to 128 bytes
0B4B    11 0080                 LD      DE,TBUF         ; Point to buffer
0B4E    AF                      XOR     A               ; Set X & Y MSBs to 0
0B4F    D3 CA                   OUT     (YCM),A
0B51    D3 C8                   OUT     (XCM),A
0B53    1A              LGL2:   LD      A,(DE)          ; Get a byte
0B54    13                      INC     DE              ; Point to next
0B55    D5                      PUSH    DE              ; Save the pointer
0B56    5F                      LD      E,A             ; Get pixel count in E
0B57    CB 3B                   SRL     E
0B59    CB 3B                   SRL     E
0B5B    CB 3B                   SRL     E
0B5D    CB 3B                   SRL     E
0B5F    E6 0F                   AND     0FH             ; Get the colour in A
0B61    CD 08E0                 CALL    READY           ; Send the colour
                                R
0B64    D3 D0                   OUT     (PENCOL),A
0B66    1C                      INC     E               ; Fudge the pixel count
0B67    CD 0B7B         LGL3:   CALL    PCOL            ; Put a colour at X,Y
                                R
0B6A    23                      INC     HL              ; Point to next pixel
0B6B    7C                      LD      A,H             ; Test for end of screen
0B6C    B5                      OR      L
0B6D    20 02                   JR      NZ,LGL4         ; Go home regardless if end
0B6F    D1                      POP     DE              ; Get DE off the stack
0B70    C9                      RET
```



```
0B71    1D              LGL4:   DEC     E               ; Count a pixel
0B72    20 F3                   JR      NZ,LGL3         ; Same colour, round again
0B74    D1                      POP     DE              ; Get TBUF pointer back
0B75    10 DC                   DJNZ    LGL2            ; More in TBUF? Round again
0B77    37                      SCF                     ; Fudge the Z flag off
0B78    CB 1F                   RR      A
0B7A    C9                      RET                     ; TBUF empty, so go home

                        ; Plot a pixel at HL (X,Y)
0B7B    CD 08E0         PCOL:   CALL    READY
                                R
0B7E    7C                      LD      A,H             ; Send H and L to Y and X
0B7F    D3 CB                   OUT     (YCL),A
0B81    7D                      LD      A,L
0B82    D3 C9                   OUT     (XCL),A
0B84    CD 08E0                 CALL    READY
                                R
0B87    3E 80                   LD      A,80H           ; Plot point command
0B89    D3 C0                   OUT     (CMD),A
0B8B    C9                      RET

                        ; Unscramble an FCB name from the string pointer in HL
                        ;  and place in FCB for use.
0B8C    06 24           FCB1:   LD      B,36            ; Clear out FCB
0B8E    11 0C68                 LD      DE,FCB
                                R
0B91    AF                      XOR     A
0B92    12              FCB1A:  LD      (DE),A
0B93    13                      INC     DE
0B94    10 FC                   DJNZ    FCB1A
                        ; Get the pointers
0B96    4E                      LD      C,(HL)          ; Get the string length
0B97    23                      INC     HL
0B98    5E                      LD      E,(HL)          ; Get pointer to
0B99    23                      INC     HL              ;      string into IX
0B9A    56                      LD      D,(HL)
0B9B    D5                      PUSH    DE
0B9C    DD E1                   POP     IX
                        ; Test if colon present in second byte
0B9E    DD 7E 01                LD      A,(IX+1)
0BA1    FE 3A                   CP      ":"
0BA3    20 10                   JR      NZ,FCB2
                        ; If colon, put drive number in FCB
0BA5    DD 7E 00                LD      A,(IX)          ; Get drive letter
0BA8    D6 40                   SUB     40H             ; Convert to number
0BAA    32 0C68                 LD      (FCB),A         ; Put in FCB
                                R
0BAD    DD 23                   INC     IX              ; Point past the drive
0BAF    DD 23                   INC     IX              ;      letter and colon
0BB1    0D                      DEC     C               ; Dec the string count
0BB2    0D                      DEC     C               ;      appropriately
0BB3    18 04                   JR      FCB3
```

```
                        ; If no colon, put default in FCB
OBB5    AF              FCB2:   XOR     A
OBB6    32 OC68                 LD      (FCB),A
                                R
                        ; Copy up filename to FCB up to . (if present).
OBB9    06 08           FCB3:   LD      B,8
OBBB    11 OC69                 LD      DE,FCB+1        ; Point to destination
                                R
OBBE    DD E5                   PUSH    IX              ; Point to source
OBCO    E1                      POP     HL
OBC1    79              FCB4:   LD      A,C             ; Get the string length
OBC2    B1                      OR      C               ; Is zero?
OBC3    28 OF                   JR      Z,FCB5          ; Yes so pad with spaces
OBC5    3E 2E                   LD      A,"."           ; Look for
OBC7    BE                      CP      (HL)
OBC8    28 OA                   JR      Z,FCB5          ; Yes so pad with spaces
OBCA    7E                      LD      A,(HL)          ; No so put it in FCB
OBCB    12                      LD      (DE),A
OBCC    23                      INC     HL              ; Point to next
OBCD    13                      INC     DE
OBCE    OD                      DEC     C               ; Dec string count
OBCF    05                      DEC     B               ; Dec character count
OBDO    20 EF                   JR      NZ,FCB4         ; More? If so round again
OBD2    28 06                   JR      Z,FCB7          ; Otherwise filename done
OBD4    3E 20           FCB5:   LD      A," "           ; Pad with spaces
OBD6    12              FCB6:   LD      (DE),A
OBD7    13                      INC     DE
OBD8    10 FC                   DJNZ    FCB6
OBDA    3E 2E           FCB7:   LD      A,"."           ; Test if HL now at .
OBDC    BE                      CP      (HL)
OBDD    20 02                   JR      NZ,FCB7A
OBDF    23                      INC     HL              ; if so, then skip it.
OBEO    OD                      DEC     C

                        ; Copy up the filetype
OBE1    06 03           FCB7A:  LD      B,3             ; Set count to three
OBE3    79              FCB8:   LD      A,C             ; Test string length
OBE4    B1                      OR      C
OBE5    28 OA                   JR      Z,FCB9          ; All done
OBE7    7E                      LD      A,(HL)          ; Get the character
OBE8    12                      LD      (DE),A          ; Put it in the FCB
OBE9    23                      INC     HL              ; Point to next
OBEA    13                      INC     DE
OBEB    OD                      DEC     C               ; Dec string count
OBEC    05                      DEC     B               ; Dec character count
OBED    20 F4                   JR      NZ,FCB8         ; More? So round again
OBEF    18 06                   JR      FCB11           ; Otherwise all done
OBF1    3E 20           FCB9:   LD      A," "           ; Pad with spaces
OBF3    12              FCB10:  LD      (DE),A
OBF4    13                      INC     DE
OBF5    10 FC                   DJNZ    FCB10
OBF7    C9              FCB11:  RET                     ; All done
```



```
                        ; Error messages
OBF8    43 61 6E 20     GMESS1: DEFM    "Can not open file, directory full.",CR,LF,"$"
OBFC    6E 6F 74 20
OCOO    6F 70 65 6E
OCO4    20 66 69 6C
OCO8    65 2C 20 64
OCOC    69 72 65 63
OC10    74 6F 72 79
OC14    20 66 75 6C
OC18    6C 2E OD OA
OC1C    24
OC1D    57 72 69 74     GMESS2: DEFM    "Write error or full disk.",CR,LF,"$"
OC21    65 20 65 72
OC25    72 6F 72 20
OC29    6F 72 20 66
OC2D    75 6C 6C 20
OC31    64 69 73 6B
OC35    2E OD OA 24
OC39    46 61 69 6C     GMESS3: DEFM    "Failed to close file.",CR,LF,"$"
OC3D    65 64 20 74
OC41    6F 20 63 6C
OC45    6F 73 65 20
OC49    66 69 6C 65
OC4D    2E OD OA 24
OC51    46 61 69 6C     GMESS4: DEFM    "Failed to open file.",CR,LF,"$"
OC55    65 64 20 74
OC59    6F 20 6F 70
OC5D    65 6E 20 66
OC61    69 6C 65 2E
OC65    OD OA 24

OC68                    FCB:    DEFS    36              ; File control block
```

Doctor Dark's Diary – Episode 17.

When issues 2 and 3 of 80-BUS News arrived together, a friend of mine commented, "Just like busses! You wait months, and then they all come along at once..." I bet he wasn't the only one to think that either! I only mention this because I think it is about to happen again. And all to save money on postage! [Ed. - Believe what you want!]

Whatever happened to... Part 94.

Readers of another magazine called Micropower may be wondering what happened to their "Win a CMOS RAM board competition". So am I. I entered the competition, and sent suggestions for five boards that I thought would be good ideas. The results of the competition have yet to appear, although they have been due for a long time, as indeed their magazine has been, but I thought you might like to have a look at what I said to them. It was more or less as follows, except that I have edited in a few extra remarks, not just to protect myself from accusations of breach of their copyright on my material, but also because I have thought of them since the competition, and they make the boards even more desirable...

The 80-BUS is particularly well suited to the use of intelligent peripheral boards, a prime example of the species being the Pluto graphics board, which has a specification not very different from that suggested for a graphics board in one of the early episodes of "Doctor Dark's Diary". One of the main features of such a board is that it is accessed through the Z80 port addressing system, so that it does not take up any of the system's memory space. I have five proposals for new 80-BUS boards, all of which would be of this type. As far as the main processor in the system is concerned, each of these boards would be controlled by the use of two or more ports, in much the same way as the Pluto and Gemini IVC boards are. One port is called the "status" port, and is used to find out whether the board is ready to receive fresh instructions or not; the other port is the "data" port, and the instructions and data are sent to this port. The five boards I propose are as follows:-

Intelligent speech generator.

This board would contain the following items:-

(a) Port decoding for two ports.
(b) A Z80 CPU.
(c) Random access memory.
(d) Read only memory.
(e) A Votrax SC-01 speech synthesis chip.
(f) Audio output circuitry.

The functioning of the board would be as follows. The main system processor would send codes representing the words to be spoken to the data port of the board. The on-board Z80 would store the instructions sent to it in the random access memory; the control program telling it how to do this would be stored in the ROM. During the periods when it was not reading the data port, which would be most of the time, the Z80 would control the SC-01 chip. The software in the ROM could be quite simple, providing only the necessary data collecting and speech control functions, or could include software to translate from normal text to the phonetic code required by the SC-01. If there was enough ROM, it would be possible to provide a dictionary of commonly used words, with their phonetic equivalents.

Intelligent sound effects board.

This board would be very similar to the board proposed above. However, the speech synthesiser chip would be replaced by two AY-3-8910 programmable sound generators. The outputs of the two chips would be fed to a DIN socket, to allow connection to a high fidelity amplifier. Since the AY-3-8910 chip provides further output lines, it could control digital to analogue converters, also connected to the board output, enabling the synthesis of far more sophisticated wave forms than those provided by the AY-3-8910 itself. An advantage of this board over the more normal sound board with no processing power of its own (such as the late, lamented Winchester Technology sound board) would be the ability to "sweep" the frequency and other control registers of the sound chip without taking up time needed by the main system processor.

Intelligent sound effects and speech board.

This board would consist simply of the above two boards combined into one. The output ports of the two AY-3-8910's would control the SC-01 chip, as well as two digital to analogue converters. There should be ample space for the port decoding logic, a Z80, ROM, RAM, two AY-3-8910's, two DAC's and an SC-01 on an eight inch square board. After all, Gemini can fit 512K of RAM in that space!

Parallel processor board.

In many applications, such as chess programs, the speed of the main processor is not sufficient to calculate as far ahead as is desirable. If it were possible for the main processor to generate moves, and pass the new positions created to other processors for evaluation, speed improvements would result. This would mean that a comparatively simple program could achieve better results by looking further ahead than is usually possible. Sargon, for example, does not usually look more than six plies deep, and can take hours to do so. A suitable board for this kind of application could be constructed using the following:

(a) Port decoding logic for eight ports.
(b) Four Z80 processors, each operated via two ports.
(c) ROM and RAM for each processor.

Thus the board would contain, in effect, four subsidiary computer systems, each able to run programs independantly of the others, and of the main processor. Since there would not be a lot of room for RAM on this board, it would be fitted with suitable connectors to enable it to be expanded on separate boards. There are many applications besides chess which require rapid results from a large number of calculations. For example, the calculation of the new coordinates of a complex moving shape, for animated 3-D displays. A board of this type would be of considerable assistance in producing fast displays with the Pluto graphics board, where it frequently happens that the Pluto board has to wait for the main processor to generate the next picture in an animated sequence. The matrix multiplication calculations used in creating moving displays are just crying out for this sort of parallel processing. Anyone feel like writing an Occam compiler for the board? Oh all right, I will do it in a spare hour, when one comes along! [Ed. - for the idiots (like me), what is Occam?]

Add-on 16 (or 32?) bit processor board.

This would be similar in concept to the board described above, but would carry only one processor, of a considerably more powerful type. As well as the new processor, there would be operating software in ROM, large amounts of RAM, and the ability to extend the board further. The board could use the Z8000 processor, along with its memory management unit, and 64K words of RAM. An alternative would be to use the Intel 8086 processor, in conjunction with the Intel 8087 floating point processor, which gives truly amazing speed of calculation. In either case, the result of using such a board would be a system that could not be described as old fashioned, as it would be able to keep up with the latest in hardware, while not wasting the investment in the existing system. In the "ultimate" system, the original main processor would still be in use, running programs that had as their main function the transfer of data and programs from one processor board to another. What a good idea the 80-BUS is! [Ed. - makes you wonder what is so 'original' about the Tycom MicroFrame claims?]

Here endeth the quotation from the competition entry, and we are back into the present. Notice that last idea? I am sure that there is no connection between Micropower and Belectra Ltd, who announced their HSA-88B recently, and am not for one moment suggesting that they have not had the same idea as I did, completely independantly. It is an extremely good idea, no matter who had it first, and I will be ordering a board from them as soon as I get the readies together. In the meantime, anyone who wants to use any of the other ideas in the list above is welcome to help themselves, as long as they send me a free board to review. I am surprised that Belectra have not sent me a board to try out, and review...

The Ring is in Orbit!

Actually, it is more like a square at the moment, but I expect we can improve on that! For those of you who are new readers, the old, printed program library has met its end - but I thought it would be a good idea to circulate discs or tapes with programs on, as a substitute for the library. Not many other people found the time to write and say that they thought so too, and it nearly didn't get off the ground. Frank Everest WAS convinced, and set a disc going for the CP/M users with Pertec DSDD drives, which earns him the Real Enthusiast's Medal (or REM, which gets ignored!), so now there are four of us swapping our programs. There would have been more, but I lost my list of names and addresses of people with similar systems. So, if you are wondering what you said that annoyed me, chaps, it wasn't anything at all! Those who want to join in the loop for systems compatible with mine had better let me know, pronto. Users of other systems, tapes, or whatever - if you want to run some sort of exchange, it won't happen if you sit on your hands and wait for it! So far, I have had free copies of some quite clever CP/M utilities, and have given away a couple of games I wrote or adapted, and the source code for boring old MONITOR.COM. It may not sound like a lot, but it is more than nothing at all, which is what you get if you do nothing...

On behalf of IO Research!

Since they seem to be too busy to let the editor know what ports the Pluto uses, or send me details of the price of the palette board, I will copy the relevant information out of the Pluto manual. (I wonder why nobody who sells Pluto boards has thought of looking in there? I thought I was the only one who only looked in the manual as a last resort!) I quote from the manual, with slight changes to avoid being accused of ripping it off... [Ed. - see article on IO mapping elsewhere.] This seems to cover all the questions asked, although it does not say whether the recently announced pallette board, for which I am willing to sell my soul, is to use the spare two ports. I expect it does, as I can not think of any other use for them...

SYS is dead — long live?

by D.R. Hunt

Obituary

SYS came into being some two and a half years ago when the Henelec - Gemini G805 disk system came on the market. The G805 CP/M disk system was fitted with a spectacularly primitive BIOS, not so much by design, but all that could be got into a 1K EPROM. SYS got round this problem by quite an ingenious method which has been dealt with at length in past issues. Of course, the whole idea of SYS was to allow the competent machine code programmer access to the BIOS of the system without having to tear half of MOVCPM apart each time a minor change was contemplated.

Over a period of time, SYS grew, and additional features were added. At the tender age of about one year SYS was rewritten incorporating parts of Gemini's 48 t.p.i. disk drivers to cater for the Gemini GM809 card. Some months later, virtual disk operation was added using Gemini or Nascom cards. At the beginning of 1983 new features were added to SYS to make it Gemini Galaxy compatible and further extensions to the virtual disk were added. Parts of Gemini's 96 t.p.i./Winchester BIOS were incorporated for completeness although, at Gemini's request, issue of this version with SYSB6A was restricted. SYS however, without SYSB6A remained popular with many, as an ideal way of adding features to the existing Gemini and Nascom 48 t.p.i/Pertec MultiBoard BIOSes.

And so SYS's future looked bright for some time to come, however, it seems that some commercial organisation has allegedly used large chunks of SYS for their own purposes and incorporated it into their own BIOS for a competeing system instead of writing their own!! Now this is not only in contravention of the copyright on SYS as a whole, and the individual copyrights on the Pertec drivers and the Gemini 96 t.p.i. disk drivers, but is against the whole concept of 'for own use only' which is what SYS is about. Gemini aren't amused, in fact it is nearer the truth to say that Gemini very much upset. Gemini have not yet withdrawn their permission to use their disk drivers in SYS, but I suspect that it won't take them long to get round to it. So reluctantly it has been decided to withdraw SYS from sale. Also, as result of this alleged piracy, the source of Gemini's BIOS is no longer distributed on their system disks F.O.C., but is available upon request for the nominal sum of £500.00.

So what is the result of this? Well one of the most flexible tools available to the machine code programmer for Nascom/Gemini and Gemini multiboard machines has now ceased to exist, and if you happen to dislike the Gemini BIOS and want to do something about it, you are left with the difficult and tedious job of writing your own BIOS or coughing up a large fortune for the Gemini sources to modify.

All very negative, it is my opinion that having purchased a CP/M at what is after all a quite high price, you should be at least entitled to the source of the parts specific to your machine. But if people are going to lift that source for their own commercial gain, then what can a manufacturer do? I do not agree with Gemini's policy of grossly overcharging for the source of their BIOS (although I can not think of a simple, safe, alternative), but it will certainly keep its circulation under control on the grounds that very few if any are going to buy it at that price, except that is, the enterprising pirate, to whom the investment of £500.00 is small beer compared with the possible return. But then if Gemini wish to pitch their prices such as to restrict the sale of their products only to the pirates ......

[Ed. - buying a copy of the listing doesn't entitle anyone to reproduce and resell - the copyright is still retained by Gemini]

80—BUS IO MAP – PART 2.

In the last issue a brief summary of the I/0 ports currently occupied by 80-BUS/Nasbus compatible boards was given, along with a request for any corrections to this and for any information on products not included. The following paragraphs are extracts from various letters received in response to the above request. Thanks to all concerned.

Pluto

Doctor Dark writes: "Only two I/0 ports are required for communication with Pluto. The ports have consecutive addresses that may be selected to be on any 20H byte boundary. Pluto decodes 4 addresses, two of which are not used, but are reserved for future use. Pluto is pre-configured with a base address of AO hex. This can be changed to any of 00, 20, 40, 60, 80, CO, or EO, all of which are in hex, of course. For compatibility with Nascom systems a NASIO signal is optionally provided by Pluto. Only one board in the entire system should provide this signal which is asserted when an IO address for the Nascom main board is decoded. If this signal is to be provided by Pluto then the points marked NASIO should be linked. Pluto asserts this signal for all addresses from 00 to 7F hex inclusive which means that all peripheral boards (including Pluto) should use I/0 addresses above 80 hex. The Nascom Internal/External I/0 addressing switch must be set to enable external addressing. For compatibility with Nascom 1 systems, a DBDR option is provided by linking the points marked."

Graphics Board

Mr R. E. Moyle writes: "You may not be aware of S. Holmes' Graphics Board. This board uses ports 8 - 31 to control a Texas Colour Graphics chip, two sound generators, a RTC, CMOS scratchpad memory and eight ADCs. Unfortunately the board is not fully 80-BUS compatible as it omits the "obsolete" signals and daisy-chain protocols. These are easily added, however, and moving the I/0 addresses to 32 - 63 is also simple."

CMOS RAM and RTC

CHS Data Sciences write: "We have produced a board which:
a) has 16K of CMOS RAM and a Real Time Clock which also detects power up/down and reset conditions.
b) standard I/0 address is DO-D3, no alternative is suggested until such time as an I/0 map is produced, the link selectable header plug may be changed for any contiguous block of 4 on a boundary of 4, paged memory is also on port FFH.
c) NASIO and DBDR are provided from open-collector gates with NASIO also being link selected
d) the board is fully Nasbus 4 compatible
e) the memory is page selected by port FFH, additionally it will always be selected on page 0 (selected by reset) regardless of page switch setting, this ensures that the board controls power up/down and 'manual' resets
This board does not use the National Semiconductor Real Time Clock and so it does produce the Year Date, the clock is supplied via a on-board battery so maintaining clocking integrity."

IO Research A/D Board

J. Da Silva Alvoeiro writes: "I have one IO Research A/D Convertor Board and it uses ports 20-23, NOT 30-33 as described in your magazine."

EV IEEE Board

EV Computing writes: "Our IEEE card uses port FF hex for page mode switching as well as the ports 34-3F as correctly shown. We apologise for this error on our part."

Lucas Nascom

Mike Hessey of Lucas Logic writes: "Please find attached a list of the I/O ports currently being used by Lucas Nascom. Other future products may use port numbers mentioned on this list. All these boards can be used with all other Nascom boards.
Nascom 1 - ports 0-2 & 4-7. No alternative ports. Uses NASIO. Requires buffering for connection to Nasbus.
Nascom 2/3 - ports 0-2 & 4-7. No alternative ports. NASIO is not used. No restrictions.
Input/Output Board - 8-B & 10-12 & 14-1F. Alternative ports: All addresses can be selected via on-board links. Potentially could access any I/O address via A7-A0. Second board would normally use addresses up to 2F. Requires interrupt daisy chain.
FDC Board - E0-E3. Selectable to 20-23, 40-43 etc.
AVC - B0-B2. Screen memory is paged in automatically by graphics support software, normally at 8000 (link selectable). Requires use of RAM disable.
RAM B - Port FF output used for page selection."

Well, Mike, I hate to contradict you (honest) but:
a) Surely NASIO is used on N2 & N3 when there is external I/O? After all, there's a switch on the PCB to select between NASIO internal/external.
b) The I/O board circuit diagram I saw implies 8-B & 10-1F are decoded.
c) The FDC circuit diagram I saw implies E0-E4 R/W and E5 R/O, and so does the customer below.

Sound and FDC

Mr. A. Brown writes: "I wish to supply the following information-
Easicomp Sound Board - Port 2, write only, port 3 read/write, or
" 10 " " " 11 " I/O provided
Lucas Nascom FDC Board - E0-E5. I/O provided."

Animation Graphics Board

Mr N. Crook writes: "The R & EW Animation Graphics board for the Nascom (Nasbus) (R&EW Jan. '83) uses ports 08-1DH inclusive as far as I can tell from the article. Although I do not have one I know several people at NASTUG who are building them.

"Your ports map is coming in useful for me since I have finally started the design of my own I/O board (blow the dust off the prototyping board!)."

Isn't this the same board mentioned by Mr Moyle above? If so then there is a contradiction in the port requirements for this board between their letters. Oh well, if anyone else has any more corrections to make, or new boards to add then please write in. I hope that I will have sufficient accurate information by the time the Nov-Dec '83 issue goes to print to provide a "this is a state-of-the-art port map of the 80-BUS at the end of 1983." For some reason I had imagined that this would be a simple job, but with some people being unwilling to supply any information and others supplying contradictory information I am beginning to think that I let myself in for somewhat more than I expected! Never mind, I won't need to bother after 31/12/83 as it will be **1984** and Big Brother will be watching over everyone for me.

NASCOM ROM BASIC DIS—ASSEMBLED

PART 2

BY CARL LLOYD—PARKER

```
Dis-assembly of NASCOM ROM BASIC Ver 4.7                    PAGE    1

                ; GENERAL EQUATES

0001            UARTD   EQU     01H             ; UART data port
0002            UARTS   EQU     02H             ; UART status port

0003            CTRLC   EQU     03H             ; Control "C"
0007            CTRLG   EQU     07H             ; Control "G"
0008            BKSP    EQU     08H             ; Back space
000A            LF      EQU     0AH             ; Line feed
000C            CS      EQU     0CH             ; Clear screen
000D            CR      EQU     0DH             ; Carriage return
000F            CTRLO   EQU     0FH             ; Control "O"
0012            CTRLR   EQU     12H             ; Control "R"
0013            CTRLS   EQU     13H             ; Control "S"
0015            CTRLU   EQU     15H             ; Control "U"
001A            CTRLZ   EQU     1AH             ; Control "Z"
001B            ESC     EQU     1BH             ; Escape
001C            TBRK    EQU     1CH             ; "T" monitor break
001D            TBS     EQU     1DH             ; "T" monitor back space
001E            TCS     EQU     1EH             ; "T" monitor clear screen
001F            TCR     EQU     1FH             ; "T" monitor carriage return
007F            DEL     EQU     7FH             ; Delete

                ; MONITOR LOCATIONS

0000            MONSTT  EQU     0000H           ; Start of monitor
000D            STMON   EQU     000DH           ; NAS-SYS initialisation
0051            MFLP    EQU     0051H           ; Flip tape LED ("T")
008D            MONTYP  EQU     008DH           ; Type of "T" monitor
03D1            T2DUMP  EQU     03D1H           ; "T2" Dump routine
0400            T4WR    EQU     0400H           ; "T4" Write routine
070C            T4READ  EQU     070CH           ; "T4" Read routine

0800            VDU     EQU     0800H           ; NASCOM Video RAM base

                ; MONITOR WORK SPACE LOCATIONS

0C00            PORTO   EQU     0C00H           ; Copy of output port 0
0C0C            ARG1    EQU     0C0CH           ; Argument 1
0C0E            ARG2    EQU     0C0EH           ; Argument 2
0C18            TCUR    EQU     0C18H           ; "T" monitor cursor
0C29            CURSOR  EQU     0C29H           ; NAS-SYS Cursor
0C2B            ARGN    EQU     0C2BH           ; Number of ARGS
0C4A            TOUT    EQU     0C4AH           ; "T" Output reflection
0C4D            TIN     EQU     0C4DH           ; "T" Input reflection
0C75            CIN     EQU     0C75H           ; NAS-SYS Input table
0C7E            NMI     EQU     0C7EH           ; NAS-SYS NMI Jump
```

```
                ; BASIC WORK SPACE LOCATIONS

1000            WRKSPC  EQU     1000H           ; BASIC Work space
1003            USR     EQU     1003H           ; "USR (x)" jump
1006            OUTSUB  EQU     1006H           ; "OUT p,n"
1007            OTPORT  EQU     1007H           ; Port (p)
1009            DIVSUP  EQU     1009H           ; Division support routine
100A            DIV1    EQU     100AH           ; <- Values
100E            DIV2    EQU     100EH           ; <-   to
1012            DIV3    EQU     1012H           ; <-   be
1015            DIV4    EQU     1015H           ; <-inserted
1017            SEED    EQU     1017H           ; Random number seed
103A            LSTRND  EQU     103AH           ; Last random number
103E            INPSUB  EQU     103EH           ; "INP (x)" Routine
103F            INPORT  EQU     103FH           ; PORT (x)
1041            NULLS   EQU     1041H           ; Number of nulls
1042            LWIDTH  EQU     1042H           ; Terminal width
1043            COMMAN  EQU     1043H           ; Width for commas
1044            NULFLG  EQU     1044H           ; Null after input byte flag
1045            CTLOFG  EQU     1045H           ; Control "O" flag
1046            LINESC  EQU     1046H           ; Lines counter
1048            LINESN  EQU     1048H           ; Lines number
104A            CHKSUM  EQU     104AH           ; Array load/save check sum
104C            NMIFLG  EQU     104CH           ; Flag for NMI break routine
104D            BRKFLG  EQU     104DH           ; Break flag
104E            RINPUT  EQU     104EH           ; Input reflection
1051            POINT   EQU     1051H           ; "POINT" reflection (unused)
1054            PSET    EQU     1054H           ; "SET"   reflection
1057            RESET   EQU     1057H           ; "RESET" reflection
105A            STRSPC  EQU     105AH           ; Bottom of string space
105C            LINEAT  EQU     105CH           ; Current line number
105E            BASTXT  EQU     105EH           ; Pointer to start of program
1061            BUFFER  EQU     1061H           ; Input buffer
1066            STACK   EQU     1066H           ; Initial stack
10AB            CURPOS  EQU     10ABH           ; Character position on line
10AC            LCRFLG  EQU     10ACH           ; Locate/Create flag
10AD            TYPE    EQU     10ADH           ; Data type flag
10AE            DATFLG  EQU     10AEH           ; Literal statement flag
10AF            LSTRAM  EQU     10AFH           ; Last available RAM
10B1            TMSTPT  EQU     10B1H           ; Temporary string pointer
10B3            TMSTPL  EQU     10B3H           ; Temporary string pool
10BF            TMPSTR  EQU     10BFH           ; Temporary string
10C3            STRBOT  EQU     10C3H           ; Bottom of string space
10C5            CUROPR  EQU     10C5H           ; Current operator in EVAL
```



```
10C7            LOOPST  EQU     10C7H           ; First statement of loop
10C9            DATLIN  EQU     10C9H           ; Line of current DATA item
10CB            FORFLG  EQU     10CBH           ; "FOR" loop flag
10CC            LSTBIN  EQU     10CCH           ; Last byte entered
10CD            READFG  EQU     10CDH           ; Read/Input flag
10CE            BRKLIN  EQU     10CEH           ; Line of break
10D0            NXTOPR  EQU     10D0H           ; Next operator in EVAL
10D2            ERRLIN  EQU     10D2H           ; Line of error
10D4            CONTAD  EQU     10D4H           ; Where to CONTinue
10D6            PROGND  EQU     10D6H           ; End of program
10D8            VAREND  EQU     10D8H           ; End of variables
10DA            ARREND  EQU     10DAH           ; End of arrays
10DC            NXTDAT  EQU     10DCH           ; Next data item
10DE            FNRGNM  EQU     10DEH           ; Name of FN argument
10E0            FNARG   EQU     10E0H           ; FN argument value
10E4            FPREG   EQU     10E4H           ; Floating point register
10E7            FPEXP   EQU     FPREG+3         ; Floating point exponent
10E8            SGNRES  EQU     10E8H           ; Sign of result
10E9            PBUFF   EQU     10E9H           ; Number print buffer
10F6            MULVAL  EQU     10F6H           ; Multiplier
10F9            PROGST  EQU     10F9H           ; Start of program text area
115D            STLOOK  EQU     115DH           ; Start of memory test

                ; BASIC ERROR CODE VALUES

0000            NF      EQU     00H             ; NEXT without FOR
0002            SN      EQU     02H             ; Syntax error
0004            RG      EQU     04H             ; RETURN without GOSUB
0006            OD      EQU     06H             ; Out of DATA
0008            FC      EQU     08H             ; Function call error
000A            OV      EQU     0AH             ; Overflow
000C            OM      EQU     0CH             ; Out of memory
000E            UL      EQU     0EH             ; Undefined line number
0010            BS      EQU     10H             ; Bad subscript
0012            DD      EQU     12H             ; Re-DIMensioned array
0014            DZ      EQU     14H             ; Division by zero (/0)
0016            ID      EQU     16H             ; Illegal direct
0018            TM      EQU     18H             ; Type mis-match
001A            OS      EQU     1AH             ; Out of string space
001C            LS      EQU     1CH             ; String too long
001E            ST      EQU     1EH             ; String formula too complex
0020            CN      EQU     20H             ; Can't CONTinue
0022            UF      EQU     22H             ; UnDEFined FN function
0024            MO      EQU     24H             ; Missing operand
```

```
E000 C303E0     START:  JP      STARTB          ; Jump for restart jump
E003 F3         STARTB: DI                      ; No interrupts
E004 DD210000           LD      IX,0            ; Flag cold start
E008 C312E0             JP      CSTART          ; Jump to initialise

E00B 8BE9               DEFW    DEINT           ; Get integer -32768 to 32767
E00D F2F0               DEFW    ABPASS          ; Return integer in AB

E00F C33CE7             JP      LDNMI1          ; << NO REFERENCE TO HERE >>

E012 210010     CSTART: LD      HL,WRKSPC       ; Start of workspace RAM
E015 F9                 LD      SP,HL           ; Set up a temporary stack
E016 C3BBFE             JP      INITST          ; Go to initialise

E019 11DFE2     INIT:   LD      DE,INITAB       ; Initialise work space
E01C 0663               LD      B,INITBE-INITAB+3;Bytes to copy
E01E 210010             LD      HL,WRKSPC       ; Into workspace RAM
E021 1A         COPY:   LD      A,(DE)          ; Get source
E022 77                 LD      (HL),A          ; To destination
E023 23                 INC     HL              ; Next destination
E024 13                 INC     DE              ; Next source
E025 05                 DEC     B               ; Count bytes
E026 C221E0             JP      NZ,COPY         ; More to move
E029 F9                 LD      SP,HL           ; Temporary stack
E02A CDDFE4             CALL    CLREG           ; Clear registers and stack
E02D CD81EB             CALL    PRNTCR          ; Output CRLF
E030 32AA10             LD      (BUFFER+72+1),A ; Mark end of buffer
E033 32F910             LD      (PROGST),A      ; Initialise program area
E036 2103E1     MSIZE:  LD      HL,MEMMSG       ; Point to message
E039 CD10F2             CALL    PRS             ; Output "Memory size"
E03C CDFCE4             CALL    PROMPT          ; Get input with "? "
E03F CD36E8             CALL    GETCHR          ; Get next character
E042 B7                 OR      A               ; Set flags
E043 C25BE0             JP      NZ,TSTMEM       ; If number - Test if RAM there
E046 215D11             LD      HL,STLOOK       ; Point to start of RAM
E049 23         MLOOP:  INC     HL              ; Next byte
E04A 7C                 LD      A,H             ; Above address FFFF ?
E04B B5                 OR      L
E04C CA6DE0             JP      Z,SETTOP        ; Yes - 64K RAM
E04F 7E                 LD      A,(HL)          ; Get contents
E050 47                 LD      B,A             ; Save it
E051 2F                 CPL                     ; Flip all bits
E052 77                 LD      (HL),A          ; Put it back
E053 BE                 CP      (HL)            ; RAM there if same
E054 70                 LD      (HL),B          ; Restore old contents
E055 CA49E0             JP      Z,MLOOP         ; If RAM - test next byte
E058 C36DE0             JP      SETTOP          ; Top of RAM found
```



```
E05B CDA5E9     TSTMEM: CALL    ATOH            ; Get high memory into DE
E05E B7                 OR      A               ; Set flags on last byte
E05F C2ADE3             JP      NZ,SNERR        ; ?SN Error if bad character
E062 EB                 EX      DE,HL           ; Address into HL
E063 2B                 DEC     HL              ; Back one byte
E064 3ED9               LD      A,11011001B     ; Test byte
E066 46                 LD      B,(HL)          ; Get old contents
E067 77                 LD      (HL),A          ; Load test byte
E068 BE                 CP      (HL)            ; RAM there if same
E069 70                 LD      (HL),B          ; Restore old contents
E06A C236E0             JP      NZ,MSIZE        ; Ask again if no RAM

E06D 2B         SETTOP: DEC     HL              ; Back one byte
E06E 115C11             LD      DE,STLOOK-1     ; See if enough RAM
E071 CD8AE6             CALL    CPDEHL          ; Compare DE with HL
E074 DA36E0             JP      C,MSIZE         ; Ask again if not enough RAM
E077 00                 NOP
E078 00                 NOP
E079 00                 NOP
E07A 00                 NOP
E07B 00                 NOP
E07C 00                 NOP
E07D 00                 NOP
E07E 00                 NOP
E07F 00                 NOP
E080 11CEFF             LD      DE,-50          ; 50 Bytes string space
E083 22AF10             LD      (LSTRAM),HL     ; Save last available RAM
E086 19                 ADD     HL,DE           ; Allocate string space
E087 225A10             LD      (STRSPC),HL     ; Save string space
E08A CDBAE4             CALL    CLRPTR          ; Clear program area
E08D 2A5A10             LD      HL,(STRSPC)     ; Get end of memory
E090 11EFFF             LD      DE,-17          ; Offset for free bytes
E093 19                 ADD     HL,DE           ; Adjust HL
E094 11F910             LD      DE,PROGST       ; Start of program text
E097 7D                 LD      A,L             ; Get LSB
E098 93                 SUB     E               ; Adjust it
E099 6F                 LD      L,A             ; Re-save
E09A 7C                 LD      A,H             ; Get MSB
E09B 9A                 SBC     A,D             ; Adjust it
E09C 67                 LD      H,A             ; Re-save
E09D E5                 PUSH    HL              ; Save bytes free
E09E 21C5E0             LD      HL,SIGNON       ; Sign-on message
E0A1 CD10F2             CALL    PRS             ; Output string
E0A4 E1                 POP     HL              ; Get bytes free back
E0A5 CDADF9             CALL    PRNTHL          ; Output amount of free memory
E0A8 21B7E0             LD      HL,BFREE        ; " Bytes free" message
E0AB CD10F2             CALL    PRS             ; Output string

E0AE 316610     WARMST: LD      SP,STACK        ; Temporary stack
E0B1 CDDFE4     BRKRET: CALL    CLREG           ; Clear registers and stack
E0B4 C3F8E3             JP      PRNTOK          ; Go to get command line
```

```
E0B7 20427974   BFREE:  DEFB    " Bytes free",CR,0,0

E0C5 4E415343   SIGNON: DEFB    "NASCOM ROM BASIC Ver 4.7   ",CR
E0E1 436F7079           DEFB    "Copyright (C) 1978 by Microsoft",CR,0,0

E103 4D656D6F   MEMMSG: DEFB    "Memory size",0

                ; FUNCTION ADDRESS TABLE

E10F 22F8       FNCTAB: DEFW    SGN
E111 E6F8               DEFW    INT
E113 38F8               DEFW    ABS
E115 0310               DEFW    USR
E117 D0F0               DEFW    FRE
E119 41F4               DEFW    INP
E11B FEF0               DEFW    POS
E11D ACFA               DEFW    SQR
E11F 8BFB               DEFW    RND
E121 C7F6               DEFW    LOG
E123 FAFA               DEFW    EXP
E125 00FC               DEFW    COS
E127 06FC               DEFW    SIN
E129 67FC               DEFW    TAN
E12B 7CFC               DEFW    ATN
E12D A3F5               DEFW    PEEK
E12F BCFD               DEFW    DEEK
E131 5110               DEFW    POINT
E133 82F3               DEFW    LEN
E135 9AF1               DEFW    STR
E137 1CF4               DEFW    VAL
E139 91F3               DEFW    ASC
E13B A2F3               DEFW    CHR
E13D B2F3               DEFW    LEFT
E13F E2F3               DEFW    RIGHT
E141 ECF3               DEFW    MID
```



```
                ; RESERVED WORD LIST

E143 C54E44     WORDS:  DEFB    80H+"END"
E146 C64F52             DEFB    80H+"FOR"
E149 CE455854           DEFB    80H+"NEXT"
E14D C4415441           DEFB    80H+"DATA"
E151 C94E5055           DEFB    80H+"INPUT"
E156 C4494D             DEFB    80H+"DIM"
E159 D2454144           DEFB    80H+"READ"
E15D CC4554             DEFB    80H+"LET"
E160 C74F544F           DEFB    80H+"GOTO"
E164 D2554E             DEFB    80H+"RUN"
E167 C946               DEFB    80H+"IF"
E169 D2455354           DEFB    80H+"RESTORE"
E170 C74F5355           DEFB    80H+"GOSUB"
E175 D2455455           DEFB    80H+"RETURN"
E17B D2454D             DEFB    80H+"REM"
E17E D3544F50           DEFB    80H+"STOP"
E182 CF5554             DEFB    80H+"OUT"
E185 CF4E               DEFB    80H+"ON"
E187 CE554C4C           DEFB    80H+"NULL"
E18B D7414954           DEFB    80H+"WAIT"
E18F C44546             DEFB    80H+"DEF"
E192 D04F4B45           DEFB    80H+"POKE"
E196 C44F4B45           DEFB    80H+"DOKE"
E19A D3435245           DEFB    80H+"SCREEN"
E1A0 CC494E45           DEFB    80H+"LINES"
E1A5 C34C53             DEFB    80H+"CLS"
E1A8 D7494454           DEFB    80H+"WIDTH"
E1AD CD4F4E49           DEFB    80H+"MONITOR"
E1B4 D34554             DEFB    80H+"SET"
E1B7 D2455345           DEFB    80H+"RESET"
E1BC D052494E           DEFB    80H+"PRINT"
E1C1 C34F4E54           DEFB    80H+"CONT"
E1C5 CC495354           DEFB    80H+"LIST"
E1C9 C34C4541           DEFB    80H+"CLEAR"
E1CE C34C4F41           DEFB    80H+"CLOAD"
E1D3 C3534156           DEFB    80H+"CSAVE"
E1D8 CE4557             DEFB    80H+"NEW"
```

```
E1DB D4414228          DEFB    80H+"TAB("
E1DF D44F              DEFB    80H+"TO"
E1E1 C64E              DEFB    80H+"FN"
E1E3 D3504328          DEFB    80H+"SPC("
E1E7 D448454E          DEFB    80H+"THEN"
E1EB CE4F54            DEFB    80H+"NOT"
E1EE D3544550          DEFB    80H+"STEP"

E1F2 AB                DEFB    80H+"+"
E1F3 AD                DEFB    80H+"-"
E1F4 AA                DEFB    80H+"*"
E1F5 AF                DEFB    80H+"/"
E1F6 DE                DEFB    80H+"^"
E1F7 C14E44            DEFB    80H+"AND"
E1FA CF52              DEFB    80H+"OR"
E1FC BE                DEFB    80H+">"
E1FD BD                DEFB    80H+"="
E1FE BC                DEFB    80H+"<"

E1FF D3474E            DEFB    80H+"SGN"
E202 C94E54            DEFB    80H+"INT"
E205 C14253            DEFB    80H+"ABS"
E208 D55352            DEFB    80H+"USR"
E20B C65245            DEFB    80H+"FRE"
E20E C94E50            DEFB    80H+"INP"
E211 D04F53            DEFB    80H+"POS"
E214 D35152            DEFB    80H+"SQR"
E217 D24E44            DEFB    80H+"RND"
E21A CC4F47            DEFB    80H+"LOG"
E21D C55850            DEFB    80H+"EXP"
E220 C34F53            DEFB    80H+"COS"
E223 D3494E            DEFB    80H+"SIN"
E226 D4414E            DEFB    80H+"TAN"
E229 C1544E            DEFB    80H+"ATN"
E22C D045454B          DEFB    80H+"PEEK"
E230 C445454B          DEFB    80H+"DEEK"
E234 D04F494E          DEFB    80H+"POINT"
E239 CC454E            DEFB    80H+"LEN"
E23C D3545224          DEFB    80H+"STR$"
E240 D6414C            DEFB    80H+"VAL"
E243 C15343            DEFB    80H+"ASC"
E246 C3485224          DEFB    80H+"CHR$"
E24A CC454654          DEFB    80H+"LEFT$"
E24F D2494748          DEFB    80H+"RIGHT$"
E255 CD494424          DEFB    80H+"MID$"
E259 80                DEFB    80H             ; End of list marker
```



```
                       ; KEYWORD ADDRESS TABLE

E25A 72E8     WORDTB:  DEFW    PEND
E25C 79E7              DEFW    FOR
E25E F6EC              DEFW    NEXT
E260 70EA              DEFW    DATA
E262 FDEB              DEFW    INPUT
E264 28EF              DEFW    DIM
E266 2CEC              DEFW    READ
E268 87EA              DEFW    LET
E26A 2DEA              DEFW    GOTO
E26C 10EA              DEFW    RUN
E26E FFEA              DEFW    IF
E270 46E8              DEFW    RESTOR
E272 1CEA              DEFW    GOSUB
E274 4BEA              DEFW    RETURN
E276 72EA              DEFW    REM
E278 70E8              DEFW    STOP
E27A 4DF4              DEFW    POUT
E27C E1EA              DEFW    ON
E27E B1E8              DEFW    NULL
E280 53F4              DEFW    WAIT
E282 06F1              DEFW    DEF
E284 AAF5              DEFW    POKE
E286 C7FD              DEFW    DOKE
E288 E6FD              DEFW    SCREEN
E28A ADFD              DEFW    LINES
E28C 8BFD              DEFW    CLS
E28E A5FD              DEFW    WIDTH
E290 A2FE              DEFW    MONITR
E292 5410              DEFW    PSET
E294 5710              DEFW    RESET
E296 23EB              DEFW    PRINT
E298 9EE8              DEFW    CONT
E29A DDE6              DEFW    LIST
E29C CAE9              DEFW    CLEAR
E29E F9F4              DEFW    CLOAD
E2A0 C3F4              DEFW    CSAVE
E2A2 B9E4              DEFW    NEW
```

```
                ; RESERVED WORD TOKEN VALUES

0080            ZEND    EQU     080H            ; END
0081            ZFOR    EQU     081H            ; FOR
0083            ZDATA   EQU     083H            ; DATA
0088            ZGOTO   EQU     088H            ; GOTO
008C            ZGOSUB  EQU     08CH            ; GOSUB
008E            ZREM    EQU     08EH            ; REM
009E            ZPRINT  EQU     09EH            ; PRINT
00A4            ZNEW    EQU     0A4H            ; NEW

00A5            ZTAB    EQU     0A5H            ; TAB
00A6            ZTO     EQU     0A6H            ; TO
00A7            ZFN     EQU     0A7H            ; FN
00A8            ZSPC    EQU     0A8H            ; SPC
00A9            ZTHEN   EQU     0A9H            ; THEN
00AA            ZNOT    EQU     0AAH            ; NOT
00AB            ZSTEP   EQU     0ABH            ; STEP

00AC            ZPLUS   EQU     0ACH            ; +
00AD            ZMINUS  EQU     0ADH            ; -
00AE            ZTIMES  EQU     0AEH            ; *
00AF            ZDIV    EQU     0AFH            ; /
00B2            ZOR     EQU     0B2H            ; OR
00B3            ZGTR    EQU     0B3H            ; >
00B4            ZEQUAL  EQU     0B4H            ; =
00B5            ZLTH    EQU     0B5H            ; <
00B6            ZSGN    EQU     0B6H            ; SGN
00C7            ZPOINT  EQU     0C7H            ; POINT
00CD            ZLEFT   EQU     0CDH            ; LEFT$
```



```
                ; ARITHMETIC PRECEDENCE TABLE

E2A4 79         PRITAB: DEFB    79H             ; Precedence value
E2A5 94F9               DEFW    PADD            ; FPREG = <last> + FPREG

E2A7 79                 DEFB    79H             ; Precedence value
E2A8 C8F5               DEFW    PSUB            ; FPREG = <last> - FPREG

E2AA 7C                 DEFB    7CH             ; Precedence value
E2AB 06F7               DEFW    MULT            ; FPREG = <last> * FPREG

E2AD 7C                 DEFB    7CH             ; Precedence value
E2AE 67F7               DEFW    DIV             ; FPREG = <last> / FPREG

E2B0 7F                 DEFB    7FH             ; Precedence value
E2B1 B5FA               DEFW    POWER           ; FPREG = <last> ^ FPREG

E2B3 50                 DEFB    50H             ; Precedence value
E2B4 81EE               DEFW    PAND            ; FPREG = <last> AND FPREG

E2B6 46                 DEFB    46H             ; Precedence value
E2B7 80EE               DEFW    POR             ; FPREG = <last> OR FPREG

                ; BASIC ERROR CODE LIST

E2B9 4E46       ERRORS: DEFB    "NF"            ; NEXT without FOR
E2BB 534E               DEFB    "SN"            ; Syntax error
E2BD 5247               DEFB    "RG"            ; RETURN without GOSUB
E2BF 4F44               DEFB    "OD"            ; Out of DATA
E2C1 4643               DEFB    "FC"            ; Illegal function call
E2C3 4F56               DEFB    "OV"            ; Overflow error
E2C5 4F4D               DEFB    "OM"            ; Out of memory
E2C7 554C               DEFB    "UL"            ; Undefined line
E2C9 4253               DEFB    "BS"            ; Bad subscript
E2CB 4444               DEFB    "DD"            ; Re-DIMensioned array
E2CD 2F30               DEFB    "/0"            ; Division by zero
E2CF 4944               DEFB    "ID"            ; Illegal direct
E2D1 544D               DEFB    "TM"            ; Type mis-match
E2D3 4F53               DEFB    "OS"            ; Out of string space
E2D5 4C53               DEFB    "LS"            ; String too long
E2D7 5354               DEFB    "ST"            ; String formula too complex
E2D9 434E               DEFB    "CN"            ; Can't CONTinue
E2DB 5546               DEFB    "UF"            ; Undefined FN function
E2DD 4D4F               DEFB    "MO"            ; Missing operand
```

```
                    ; INITIALISATION TABLE

E2DF C3AEE0     INITAB: JP      WARMST          ; Warm start jump
E2E2 C3A0E9             JP      FCERR           ; "USR (X)" jump (Set to Error)

E2E5 D300               OUT     (0),A           ; "OUT p,n" skeleton
E2E7 C9                 RET

E2E8 D600               SUB     0               ; Division support routine
E2EA 6F                 LD      L,A
E2EB 7C                 LD      A,H
E2EC DE00               SBC     A,0
E2EE 67                 LD      H,A
E2EF 78                 LD      A,B
E2F0 DE00               SBC     A,0
E2F2 47                 LD      B,A
E2F3 3E00               LD      A,0
E2F5 C9                 RET

E2F6 000000             DEFB    0,0,0                   ; Random number seed
                                                        ; Table used by RND
E2F9 354ACA99           DEFB    035H,04AH,0CAH,099H     ;-2.65145E+07
E2FD 391C7698           DEFB    039H,01CH,076H,098H     ; 1.61291E+07
E301 2295B398           DEFB    022H,095H,0B3H,098H     ;-1.17691E+07
E305 0ADD4798           DEFB    00AH,0DDH,047H,098H     ; 1.30983E+07
E309 53D19999           DEFB    053H,0D1H,099H,099H     ;-2.01612E+07
E30D 0A1A9F98           DEFB    00AH,01AH,09FH,098H     ;-1.04269E+07
E311 65BCCD98           DEFB    065H,0BCH,0CDH,098H     ;-1.34831E+07
E315 D6773E98           DEFB    0D6H,077H,03EH,098H     ; 1.24825E+07
E319 52C74F80           DEFB    052H,0C7H,04FH,080H     ; Last random number

E31D DB00               IN      A,(0)           ; INP (x) skeleton
E31F C9                 RET

E320 01                 DEFB    1               ; POS (x) number (1)
E321 2F                 DEFB    47              ; Terminal width (47)
E322 1C                 DEFB    28              ; Width for commas (3 columns)
E323 00                 DEFB    0               ; No nulls after input bytes
E324 00                 DEFB    0               ; Output enabled (^0 off)

E325 0500               DEFW    5               ; Initial lines counter
E327 0500               DEFW    5               ; Initial lines number
E329 0000               DEFW    0               ; Array load/save check sum

E32B 00                 DEFB    0               ; Break not by NMI
E32C 00                 DEFB    0               ; Break flag

E32D C307E6             JP      TTYLIN          ; Input reflection (set to TTY)
E330 C379FF             JP      POINTB          ; POINT reflection unused
E333 C340FF             JP      SETB            ; SET   reflection
E336 C355FF             JP      RESETB          ; RESET reflection

E339 5D11               DEFW    STLOOK          ; Temp string space
E33B FEFF               DEFW    -2              ; Current line number (cold)
E33D FA10               DEFW    PROGST+1        ; Start of program text
E33F            INITBE:                         ; END OF INITIALISATION TABLE
```



```
E33F 20457272   ERRMSG: DEFB    " Error",0
E346 20696E20   INMSG:  DEFB    " in ",0
E34A            ZERBYT  EQU     $-1             ; A zero byte
E34B 4F6B0D00   OKMSG:  DEFB    "Ok",CR,0,0
E350 42726561   BRKMSG: DEFB    "Break",0

E356 210400     BAKSTK: LD      HL,4            ; Look for "FOR" block with
E359 39                 ADD     HL,SP           ; same index as specified
E35A 7E         LOKFOR: LD      A,(HL)          ; Get block ID
E35B 23                 INC     HL              ; Point to index address
E35C FE81               CP      ZFOR            ; Is it a "FOR" token
E35E C0                 RET     NZ              ; No - exit
E35F 4E                 LD      C,(HL)          ; BC = Address of "FOR" index
E360 23                 INC     HL
E361 46                 LD      B,(HL)
E362 23                 INC     HL              ; Point to sign of STEP
E363 E5                 PUSH    HL              ; Save pointer to sign
E364 69                 LD      L,C             ; HL = address of "FOR" index
E365 60                 LD      H,B
E366 7A                 LD      A,D             ; See if an index was specified
E367 B3                 OR      E               ; DE = 0 if no index specified
E368 EB                 EX      DE,HL           ; Specified index into HL
E369 CA70E3             JP      Z,INDFND        ; Skip if no index given
E36C EB                 EX      DE,HL           ; Index back into DE
E36D CD8AE6             CALL    CPDEHL          ; Compare index with one given
E370 010D00     INDFND: LD      BC,16-3         ; Offset to next block
E373 E1                 POP     HL              ; Restore pointer to sign
E374 C8                 RET     Z               ; Return if block found
E375 09                 ADD     HL,BC           ; Point to next block
E376 C35AE3             JP      LOKFOR          ; Keep on looking

E379 CD93E3     MOVUP:  CALL    ENFMEM          ; See if enough memory
E37C C5         MOVSTR: PUSH    BC              ; Save end of source
E37D E3                 EX      (SP),HL         ; Swap source and dest" end
E37E C1                 POP     BC              ; Get end of destination
E37F CD8AE6     MOVLP:  CALL    CPDEHL          ; See if list moved
E382 7E                 LD      A,(HL)          ; Get byte
E383 02                 LD      (BC),A          ; Move it
E384 C8                 RET     Z               ; Exit if all done
E385 0B                 DEC     BC              ; Next byte to move to
E386 2B                 DEC     HL              ; Next byte to move
E387 C37FE3             JP      MOVLP           ; Loop until all bytes moved
```

```
E38A E5          CHKSTK: PUSH   HL              ; Save code string address
E38B 2ADA10              LD     HL,(ARREND)     ; Lowest free memory
E38E 0600                LD     B,0             ; BC = Number of levels to test
E390 09                  ADD    HL,BC           ; 2 Bytes for each level
E391 09                  ADD    HL,BC
E392 3E                  DEFB   (LD A,n)        ; Skip "PUSH HL"
E393 E5          ENFMEM: PUSH   HL              ; Save code string address
E394 3ED0                LD     A,LOW -48       ; 48 Bytes minimum RAM
E396 95                  SUB    L
E397 6F                  LD     L,A
E398 3EFF                LD     A,HIGH -48      ; 48 Bytes minimum RAM
E39A 9C                  SBC    A,H
E39B DAA2E3              JP     C,OMERR         ; Not enough - ?OM Error
E39E 67                  LD     H,A
E39F 39                  ADD    HL,SP           ; Test if stack is overflowed
E3A0 E1                  POP    HL              ; Restore code string address
E3A1 D8                  RET    C               ; Return if enough mmory
E3A2 1E0C        OMERR:  LD     E,OM            ; ?OM Error
E3A4 C3C1E3              JP     ERROR

E3A7 2AC910      DATSNR: LD     HL,(DATLIN)     ; Get line of current DATA item
E3AA 225C10              LD     (LINEAT),HL     ; Save as current line
E3AD 1E02        SNERR:  LD     E,SN            ; ?SN Error
E3AF 01                  DEFB   (LD  BC,nn)     ; Skip "LD E,DZ"
E3B0 1E14        DZERR:  LD     E,DZ            ; ?/0 Error
E3B2 01                  DEFB   (LD  BC,nn)     ; Skip "LD E,NF"
E3B3 1E00        NFERR:  LD     E,NF            ; ?NF Error
E3B5 01                  DEFB   (LD  BC,nn)     ; Skip "LD E,DD"
E3B6 1E12        DDERR:  LD     E,DD            ; ?DD Error
E3B8 01                  DEFB   (LD  BC,nn)     ; Skip "LD E,UF"
E3B9 1E22        UFERR:  LD     E,UF            ; ?UF Error
E3BB 01                  DEFB   (LD  BC,nn)     ; Skip "LD E,OV"
E3BC 1E0A        OVERR:  LD     E,OV            ; ?OV Error
E3BE 01                  DEFB   (LD  BC,nn)     ; Skip "LD E,TM"
E3BF 1E18        TMERR:  LD     E,TM            ; ?TM Error
```



```
E3C1 CDDFE4      ERROR:  CALL   CLREG           ; Clear registers and stack
E3C4 324510              LD     (CTLOFG),A      ; Enable output (A is 0)
E3C7 CD74EB              CALL   STTLIN          ; Start new line
E3CA 21B9E2              LD     HL,ERRORS       ; Point to error codes
E3CD 57                  LD     D,A             ; D = 0  (A is 0)
E3CE 3E3F                LD     A,"?"
E3D0 CD9BE6              CALL   OUTC            ; Output "?"
E3D3 19                  ADD    HL,DE           ; Offset to correct error code
E3D4 7E                  LD     A,(HL)          ; First character
E3D5 CD9BE6              CALL   OUTC            ; Output it
E3D8 CD36E8              CALL   GETCHR          ; Get next character
E3DB CD9BE6              CALL   OUTC            ; Output it
E3DE 213FE3              LD     HL,ERRMSG       ; "Error" message
E3E1 CD10F2      ERRIN:  CALL   PRS             ; Output message
E3E4 2A5C10              LD     HL,(LINEAT)     ; Get line of error
E3E7 11FEFF              LD     DE,-2           ; Cold start error if -2
E3EA CD8AE6              CALL   CPDEHL          ; See if cold start error
E3ED CA12E0              JP     Z,CSTART        ; Cold start error - Restart
E3F0 7C                  LD     A,H             ; Was it a direct error?
E3F1 A5                  AND    L               ; Line = -1 if direct error
E3F2 3C                  INC    A
E3F3 C4A5F9              CALL   NZ,LINEIN       ; No - output line of error
E3F6 3E                  DEFB   (LD A,n)        ; Skip "POP BC"
E3F7 C1          POPNOK: POP    BC              ; Drop address in input buffer
```

```
E3F8 AF         PRNTOK: XOR   A               ; Output "Ok" and get command
E3F9 324510             LD    (CTLOFG),A      ; Enable output
E3FC CD74EB             CALL  STTLIN          ; Start new line
E3FF 214BE3             LD    HL,OKMSG        ; "Ok" message
E402 CD10F2             CALL  PRS             ; Output "Ok"
E405 21FFFF     GETCMD: LD    HL,-1           ; Flag direct mode
E408 225C10             LD    (LINEAT),HL     ; Save as current line
E40B CDF2E5             CALL  GETLIN          ; Get an input line
E40E DA05E4             JP    C,GETCMD        ; Get line again if break
E411 CD36E8             CALL  GETCHR          ; Get first character
E414 3C                 INC   A               ; Test if end of line
E415 3D                 DEC   A               ; Without affecting Carry
E416 CA05E4             JP    Z,GETCMD        ; Nothing entered - Get another
E419 F5                 PUSH  AF              ; Save Carry status
E41A CDA5E9             CALL  ATOH            ; Get line number into DE
E41D D5                 PUSH  DE              ; Save line number
E41E CD09E5             CALL  CRUNCH          ; Tokenise rest of line
E421 47                 LD    B,A             ; Length of tokenised line
E422 D1                 POP   DE              ; Restore line number
E423 F1                 POP   AF              ; Restore Carry
E424 D216E8             JP    NC,EXCUTE       ; No line number - Direct mode
E427 D5                 PUSH  DE              ; Save line number
E428 C5                 PUSH  BC              ; Save length of tokenised line
E429 AF                 XOR   A
E42A 32CC10             LD    (LSTBIN),A      ; Clear last byte input
E42D CD36E8             CALL  GETCHR          ; Get next character
E430 B7                 OR    A               ; Set flags
E431 F5                 PUSH  AF              ; And save them
E432 CD99E4             CALL  SRCHLN          ; Search for line number in DE
E435 DA3EE4             JP    C,LINFND        ; Jump if line found
E438 F1                 POP   AF              ; Get status
E439 F5                 PUSH  AF              ; And re-save
E43A CA46EA             JP    Z,ULERR         ; Nothing after number - Error
E43D B7                 OR    A               ; Clear Carry
E43E C5         LINFND: PUSH  BC              ; Save address of line in prog
E43F D255E4             JP    NC,INEWLN       ; Line not found - Insert new
E442 EB                 EX    DE,HL           ; Next line address in DE
E443 2AD610             LD    HL,(PROGND)     ; End of program
E446 1A         SFTPRG: LD    A,(DE)          ; Shift rest of program down
E447 02                 LD    (BC),A
E448 03                 INC   BC              ; Next destination
E449 13                 INC   DE              ; Next source
E44A CD8AE6             CALL  CPDEHL          ; All done?
E44D C246E4             JP    NZ,SFTPRG       ; More to do
E450 60                 LD    H,B             ; HL = New end of program
E451 69                 LD    L,C
E452 22D610             LD    (PROGND),HL     ; Update end of program
```



```
E455 D1         INEWLN: POP   DE              ; Get address of line
E456 F1                 POP   AF              ; Get status
E457 CA7CE4             JP    Z,SETPTR        ; No text - Set up pointers
E45A 2AD610             LD    HL,(PROGND)     ; Get end of program
E45D E3                 EX    (SP),HL         ; Get length of input line
E45E C1                 POP   BC              ; End of program to BC
E45F 09                 ADD   HL,BC           ; Find new end
E460 E5                 PUSH  HL              ; Save new end
E461 CD79E3             CALL  MOVUP           ; Make space for line
E464 E1                 POP   HL              ; Restore new end
E465 22D610             LD    (PROGND),HL     ; Update end of program pointer
E468 EB                 EX    DE,HL           ; Get line to move up in HL
E469 74                 LD    (HL),H          ; Save MSB
E46A D1                 POP   DE              ; Get new line number
E46B 23                 INC   HL              ; Skip pointer
E46C 23                 INC   HL
E46D 73                 LD    (HL),E          ; Save LSB of line number
E46E 23                 INC   HL
E46F 72                 LD    (HL),D          ; Save MSB of line number
E470 23                 INC   HL              ; To first byte in line
E471 116110             LD    DE,BUFFER       ; Copy buffer to program
E474 1A         MOVBUF: LD    A,(DE)          ; Get source
E475 77                 LD    (HL),A          ; Save destinations
E476 23                 INC   HL              ; Next source
E477 13                 INC   DE              ; Next destination
E478 B7                 OR    A               ; Done?
E479 C274E4             JP    NZ,MOVBUF       ; No - Repeat
E47C CDC5E4     SETPTR: CALL  RUNFST          ; Set line pointers
E47F 23                 INC   HL              ; To LSB of pointer
E480 EB                 EX    DE,HL           ; Address to DE
E481 62         PTRLP:  LD    H,D             ; Address to HL
E482 6B                 LD    L,E
E483 7E                 LD    A,(HL)          ; Get LSB of pointer
E484 23                 INC   HL              ; To MSB of pointer
E485 B6                 OR    (HL)            ; Compare with MSB pointer
E486 CA05E4             JP    Z,GETCMD        ; Get command line if end
E489 23                 INC   HL              ; To LSB of line number
E48A 23                 INC   HL              ; Skip line number
E48B 23                 INC   HL              ; Point to first byte in line
E48C AF                 XOR   A               ; Looking for 00 byte
E48D BE         FNDEND: CP    (HL)            ; Found end of line?
E48E 23                 INC   HL              ; Move to next byte
E48F C28DE4             JP    NZ,FNDEND       ; No - Keep looking
E492 EB                 EX    DE,HL           ; Next line address to HL
E493 73                 LD    (HL),E          ; Save LSB of pointer
E494 23                 INC   HL
E495 72                 LD    (HL),D          ; Save MSB of pointer
E496 C381E4             JP    PTRLP           ; Do next line
```

```
E499 2A5E10     SRCHLN: LD      HL,(BASTXT)     ; Start of program text
E49C 44         SRCHLP: LD      B,H             ; BC = Address to look at
E49D 4D                 LD      C,L
E49E 7E                 LD      A,(HL)          ; Get address of next line
E49F 23                 INC     HL
E4A0 B6                 OR      (HL)            ; End of program found?
E4A1 2B                 DEC     HL
E4A2 C8                 RET     Z               ; Yes - Line not found
E4A3 23                 INC     HL
E4A4 23                 INC     HL
E4A5 7E                 LD      A,(HL)          ; Get LSB of line number
E4A6 23                 INC     HL
E4A7 66                 LD      H,(HL)          ; Get MSB of line number
E4A8 6F                 LD      L,A
E4A9 CD8AE6             CALL    CPDEHL          ; Compare with line in DE
E4AC 60                 LD      H,B             ; HL = Start of this line
E4AD 69                 LD      L,C
E4AE 7E                 LD      A,(HL)          ; Get LSB of next line address
E4AF 23                 INC     HL
E4B0 66                 LD      H,(HL)          ; Get MSB of next line address
E4B1 6F                 LD      L,A             ; Next line to HL
E4B2 3F                 CCF
E4B3 C8                 RET     Z               ; Lines found - Exit
E4B4 3F                 CCF
E4B5 D0                 RET     NC              ; Line not found,at line after
E4B6 C39CE4             JP      SRCHLP          ; Keep looking
```



```
E4B9 C0         NEW:    RET     NZ              ; Return if any more on line
E4BA 2A5E10     CLRPTR: LD      HL,(BASTXT)     ; Point to start of program
E4BD AF                 XOR     A               ; Set program area to empty
E4BE 77                 LD      (HL),A          ; Save LSB = 00
E4BF 23                 INC     HL
E4C0 77                 LD      (HL),A          ; Save MSB = 00
E4C1 23                 INC     HL
E4C2 22D610             LD      (PROGND),HL     ; Set program end

E4C5 2A5E10     RUNFST: LD      HL,(BASTXT)     ; Clear all variables
E4C8 2B                 DEC     HL

E4C9 22CE10     INTVAR: LD      (BRKLIN),HL     ; Initialise RUN variables
E4CC 2AAF10             LD      HL,(LSTRAM)     ; Get end of RAM
E4CF 22C310             LD      (STRBOT),HL     ; Clear string space
E4D2 AF                 XOR     A
E4D3 CD46E8             CALL    RESTOR          ; Reset DATA pointers
E4D6 2AD610             LD      HL,(PROGND)     ; Get end of program
E4D9 22D810             LD      (VAREND),HL     ; Clear variables
E4DC 22DA10             LD      (ARREND),HL     ; Clear arrays

E4DF C1         CLREG:  POP     BC              ; Save return address
E4E0 2A5A10             LD      HL,(STRSPC)     ; Get end of working RAM
E4E3 F9                 LD      SP,HL           ; Set stack
E4E4 21B310             LD      HL,TMSTPL       ; Temporary string pool
E4E7 22B110             LD      (TMSTPT),HL     ; Reset temporary string ptr
E4EA AF                 XOR     A               ; A = 00
E4EB 6F                 LD      L,A             ; HL = 0000
E4EC 67                 LD      H,A
E4ED 22D410             LD      (CONTAD),HL     ; No CONTinue
E4F0 32CB10             LD      (FORFLG),A      ; Clear FOR flag
E4F3 22DE10             LD      (FNRGNM),HL     ; Clear FN argument
E4F6 E5                 PUSH    HL              ; HL = 0000
E4F7 C5                 PUSH    BC              ; Put back return
E4F8 2ACE10     DOAGN:  LD      HL,(BRKLIN)     ; Get address of code to RUN
E4FB C9                 RET                     ; Return to execution driver

E4FC 3E3F       PROMPT: LD      A,"?"           ; "?"
E4FE CD9BE6             CALL    OUTC            ; Output character
E501 3E20               LD      A," "           ; Space
E503 CD9BE6             CALL    OUTC            ; Output character
E506 C34E10             JP      RINPUT          ; Get input line
```

```
E509 AF        CRUNCH: XOR   A              ; Tokenise line % HL to BUFFER
E50A 32AE10            LD    (DATFLG),A     ; Reset literal flag
E50D 0E05              LD    C,2+3          ; 2 byte number and 3 nulls
E50F 116110            LD    DE,BUFFER      ; Start of input buffer
E512 7E        CRNCLP: LD    A,(HL)         ; Get byte
E513 FE20              CP    " "            ; Is it a space?
E515 CA91E5            JP    Z,MOVDIR       ; Yes - Copy direct
E518 47                LD    B,A            ; Save character
E519 FE22              CP    '"'            ; Is it a quote?
E51B CAB1E5            JP    Z,CPYLIT       ; Yes - Copy literal string
E51E B7                OR    A              ; Is it end of buffer?
E51F CAB8E5            JP    Z,ENDBUF       ; Yes - End buffer
E522 3AAE10            LD    A,(DATFLG)     ; Get data type
E525 B7                OR    A              ; Literal?
E526 7E                LD    A,(HL)         ; Get byte to copy
E527 C291E5            JP    NZ,MOVDIR      ; Literal - Copy direct
E52A FE3F              CP    "?"            ; Is it "?" short for PRINT
E52C 3E9E              LD    A,ZPRINT       ; "PRINT" token
E52E CA91E5            JP    Z,MOVDIR       ; Yes - replace it
E531 7E                LD    A,(HL)         ; Get byte again
E532 FE30              CP    "0"            ; Is it less than "0"
E534 DA3CE5            JP    C,FNDWRD       ; Yes - Look for reserved words
E537 FE3C              CP    ";"+1          ; Is it "0123456789:;" ?
E539 DA91E5            JP    C,MOVDIR       ; Yes - copy it direct
E53C D5        FNDWRD: PUSH  DE             ; Look for reserved words
E53D 1142E1            LD    DE,WORDS-1     ; Point to table
E540 C5                PUSH  BC             ; Save count
E541 018DE5            LD    BC,RETNAD      ; Where to return to
E544 C5                PUSH  BC             ; Save return address
E545 067F              LD    B,ZEND-1       ; First token value -1
E547 7E                LD    A,(HL)         ; Get byte
E548 FE61              CP    "a"            ; Less than "a" ?
E54A DA55E5            JP    C,SEARCH       ; Yes - search for words
E54D FE7B              CP    "z"+1          ; Greater than "z" ?
E54F D255E5            JP    NC,SEARCH      ; Yes - search for words
E552 E65F              AND   01011111B      ; Force upper case
E554 77                LD    (HL),A         ; Replace byte
E555 4E        SEARCH: LD    C,(HL)         ; Search for a word
E556 EB                EX    DE,HL
E557 23        GETNXT: INC   HL             ; Get next reserved word
E558 B6                OR    (HL)           ; Start of word?
E559 F257E5            JP    P,GETNXT       ; No - move on
E55C 04                INC   B              ; Increment token value
E55D 7E                LD    A,(HL)         ; Get byte from table
E55E E67F              AND   01111111B      ; Strip bit 7
E560 C8                RET   Z              ; Return if end of list
E561 B9                CP    C              ; Same character as in buffer?
E562 C257E5            JP    NZ,GETNXT      ; No - get next word
E565 EB                EX    DE,HL
E566 E5                PUSH  HL             ; Save start of word
```



```
E567 13        NXTBYT: INC   DE             ; Look through rest of word
E568 1A                LD    A,(DE)         ; Get byte from table
E569 B7                OR    A              ; End of word ?
E56A FA89E5            JP    M,MATCH        ; Yes - Match found
E56D 4F                LD    C,A            ; Save it
E56E 78                LD    A,B            ; Get token value
E56F FE88              CP    ZGOTO          ; Is it "GOTO" token ?
E571 C278E5            JP    NZ,NOSPC       ; No - Don't allow spaces
E574 CD36E8            CALL  GETCHR         ; Get next character
E577 2B                DEC   HL             ; Cancel increment from GETCHR
E578 23        NOSPC:  INC   HL             ; Next byte
E579 7E                LD    A,(HL)         ; Get byte
E57A FE61              CP    "a"            ; Less than "a" ?
E57C DA81E5            JP    C,NOCHNG       ; Yes - don't change
E57F E65F              AND   01011111B      ; Make upper case
E581 B9        NOCHNG: CP    C              ; Same as in buffer ?
E582 CA67E5            JP    Z,NXTBYT       ; Yes - keep testing
E585 E1                POP   HL             ; Get back start of word
E586 C355E5            JP    SEARCH         ; Look at next word

E589 48        MATCH:  LD    C,B            ; Word found - Save token value
E58A F1                POP   AF             ; Throw away return
E58B EB                EX    DE,HL
E58C C9                RET                  ; Return to "RETNAD"
E58D EB        RETNAD: EX    DE,HL          ; Get address in string
E58E 79                LD    A,C            ; Get token value
E58F C1                POP   BC             ; Restore buffer length
E590 D1                POP   DE             ; Get destination address
E591 23        MOVDIR: INC   HL             ; Next source in buffer
E592 12                LD    (DE),A         ; Put byte in buffer
E593 13                INC   DE             ; Move up buffer
E594 0C                INC   C              ; Increment length of buffer
E595 D63A              SUB   ":"            ; End of statement?
E597 CA9FE5            JP    Z,SETLIT       ; Jump if multi-statement line
E59A FE49              CP    ZDATA-":"      ; Is it DATA statement ?
E59C C2A2E5            JP    NZ,TSTREM      ; No - see if REM
E59F 32AE10    SETLIT: LD    (DATFLG),A     ; Set literal flag
E5A2 D654      TSTREM: SUB   ZREM-":"       ; Is it REM?
E5A4 C212E5            JP    NZ,CRNCLP      ; No - Leave flag
E5A7 47                LD    B,A            ; Copy rest of buffer
E5A8 7E        NXTCHR: LD    A,(HL)         ; Get byte
E5A9 B7                OR    A              ; End of line ?
E5AA CAB8E5            JP    Z,ENDBUF       ; Yes - Terminate buffer
E5AD B8                CP    B              ; End of statement ?
E5AE CA91E5            JP    Z,MOVDIR       ; Yes - Get next one
E5B1 23        CPYLIT: INC   HL             ; Move up source string
E5B2 12                LD    (DE),A         ; Save in destination
E5B3 0C                INC   C              ; Increment length
E5B4 13                INC   DE             ; Move up destination
E5B5 C3A8E5            JP    NXTCHR         ; Repeat
```

```
E5B8 216010   ENDBUF: LD    HL,BUFFER-1    ; Point to start of buffer
E5BB 12               LD    (DE),A         ; Mark end of buffer (A = 00)
E5BC 13               INC   DE
E5BD 12               LD    (DE),A         ; A = 00
E5BE 13               INC   DE
E5BF 12               LD    (DE),A         ; A = 00
E5C0 C9               RET

E5C1 3A4410   DODEL:  LD    A,(NULFLG)     ; Get null flag status
E5C4 B7               OR    A              ; Is it zero?
E5C5 3E00             LD    A,0            ; Zero A - Leave flags
E5C7 324410           LD    (NULFLG),A     ; Zero null flag
E5CA C2D5E5           JP    NZ,ECHDEL      ; Set - Echo it
E5CD 05               DEC   B              ; Decrement length
E5CE CAF2E5           JP    Z,GETLIN       ; Get line again if empty
E5D1 CD9BE6           CALL  OUTC           ; Output null character
E5D4 3E               DEFB  (LD A,n)       ; Skip "DEC B"
E5D5 05       ECHDEL: DEC   B              ; Count bytes in buffer
E5D6 2B               DEC   HL             ; Back space buffer
E5D7 CAE9E5           JP    Z,OTKLN        ; No buffer - Try again
E5DA 7E               LD    A,(HL)         ; Get deleted byte
E5DB CD9BE6           CALL  OUTC           ; Echo it
E5DE C310E6           JP    MORINP         ; Get more input

E5E1 05       DELCHR: DEC   B              ; Count bytes in buffer
E5E2 2B               DEC   HL             ; Back space buffer
E5E3 CD9BE6           CALL  OUTC           ; Output character in A
E5E6 C210E6           JP    NZ,MORINP      ; Not end - Get more
E5E9 CD9BE6   OTKLN:  CALL  OUTC           ; Output character in A
E5EC CD81EB   KILIN:  CALL  PRNTCR         ; Output CRLF
E5EF C307E6           JP    TTYLIN         ; Get line again

E5F2 CD6DFE   GETLIN: CALL  MONTST         ; Is it NAS-SYS?
E5F5 CA07E6           JP    Z,TTYLIN       ; No - Character input
E5F8 2A750C           LD    HL,(CIN)       ; Point to NAS-SYS input table
E5FB 7E               LD    A,(HL)         ; Get input mode
E5FC FE74             CP    74H            ; Is it "X" mode?
E5FE CA07E6           JP    Z,TTYLIN       ; Yes - Teletype line input
E601 CDE8FE           CALL  INLINE         ; Get a line from NAS-SYS
E604 C386EB           JP    DONULL         ; POS(X)=0 and do nulls
```



```
E607 216110   TTYLIN: LD    HL,BUFFER      ; Get a line by character
E60A 0601             LD    B,1            ; Set buffer as empty
E60C AF               XOR   A
E60D 324410           LD    (NULFLG),A     ; Clear null flag
E610 CDCCE6   MORINP: CALL  CLOTST         ; Get character and test ^O
E613 4F               LD    C,A            ; Save character in C
E614 FE7F             CP    DEL            ; Delete character?
E616 CAC1E5           JP    Z,DODEL        ; Yes - Process it
E619 3A4410           LD    A,(NULFLG)     ; Get null flag
E61C B7               OR    A              ; Test null flag status
E61D CA29E6           JP    Z,PROCES       ; Reset - Process character
E620 3E00             LD    A,0            ; Set a null
E622 CD9BE6           CALL  OUTC           ; Output null
E625 AF               XOR   A              ; Clear A
E626 324410           LD    (NULFLG),A     ; Reset null flag
E629 79       PROCES: LD    A,C            ; Get character
E62A FE07             CP    CTRLG          ; Bell?
E62C CA6DE6           JP    Z,PUTCTL       ; Yes - Save it
E62F FE03             CP    CTRLC          ; Is it control "C"?
E631 CC81EB           CALL  Z,PRNTCR       ; Yes - Output CRLF
E634 37               SCF                  ; Flag break
E635 C8               RET   Z              ; Return if control "C"
E636 FE0D             CP    CR             ; Is it enter?
E638 CA7CEB           JP    Z,ENDINP       ; Yes - Terminate input
E63B FE15             CP    CTRLU          ; Is it control "U"?
E63D CAECE5           JP    Z,KILIN        ; Yes - Get another line
E640 FE40             CP    "%"            ; Is it "kill line"?
E642 CAE9E5           JP    Z,OTKLN        ; Yes - Kill line
E645 FE5F             CP    "_"            ; Is it delete?
E647 CAE1E5           JP    Z,DELCHR       ; Yes - delete character
E64A FE08             CP    BKSP           ; Is it back space?
E64C CAE1E5           JP    Z,DELCHR       ; Yes - Delete character
E64F FE12             CP    CTRLR          ; Is it control "R"?
E651 C268E6           JP    NZ,PUTBUF      ; No - put in buffer
E654 C5               PUSH  BC             ; Save buffer length
E655 D5               PUSH  DE             ; Save DE
E656 E5               PUSH  HL             ; Save buffer address
E657 3600             LD    (HL),0         ; Mark end of buffer
E659 CDF4FF           CALL  OUTNCR         ; Output and do CRLF
E65C 216110           LD    HL,BUFFER      ; Point to buffer start
E65F CD10F2           CALL  PRS            ; Output buffer
E662 E1               POP   HL             ; Restore buffer address
E663 D1               POP   DE             ; Restore DE
E664 C1               POP   BC             ; Restore buffer length
E665 C310E6           JP    MORINP         ; Get another character
```

RAM–DISKS

by D. Parkinson

80BUS WARNING
READING ABOUT RAM-DISKS MAY ADVERSELY AFFECT YOUR WALLET

ON-GOING SITUATIONS

Starting off back in the Nascom days, the main expansion memory was the domestic tape recorder. Using this programs and data were saved for reloading later. The data transfer rate varied from about 110 Baud upto 2400 Baud. (A few adventurous souls reached the dizzy heights of 4800 baud). In fact there must be megabytes of data held on the humble audio cassette, which was recorded on ordinary domestic tape recorders. The next step on from this was the arrival of the floppy disk in the form of GM805, followed in a little while by double and quad density disk drives and the various 80-BUS disk controller boards. The disk systems offered far higher performance than the cassette, and after a while those who were using disks began to wonder how on earth they used to manage with cassette tapes. The trouble is that users' habits change to meet the capabilities of the equipment they use. With the faster access time of disks they started using multipass compilers, linking loaders, and a lot of other software that uses disk for the intermediate storage of data. So after a while the cry goes out again for higher performance. The next step on is the Winchester disk. This offers high capacity, (you don't have to keep changing disks), together with faster access times. But unfortunately it is expensive, requires a specialised controller (which isn't exactly cheap either!), and is still a mechanical device with the attendent constraints on access times. By now some programs are manipulating large amounts of data, and as the system memory is finite (64k), most of this manipulation is done via intermediate storage on disk. Life would be far better if more memory was available.

MEMORY EXPANSION

Various approaches have been used to expand the address space of the Z80. The two that 80-BUS readers will be familiar with are the original Nascom "page-mode", where individual memory boards are paged into and out of the system, and the Gemini memory mapping scheme implemented on the GM813 CPU/Memory board.

Any method of expanding the address space of the Z80 suffers from the problem that it tends to be unique to a particular system, and no standard software will support it. Only software written specifically for the system would be any use, and it is likely that software like that would be extremely thin on the ground. So the approach being taken now, by a large number of manufacturers, is to use the extra memory in conjunction with disk operating systems (DOSs), and to make the memory appear as a disk. This solves virtually all the problems. The DOS handles the management problem of organising data in the extra memory, and all that has to be added to the BIOS of the disk system, is a simple driver to convert a disk track/sector read/write request into a read/write request to a specific area of this extra memory. All the standard disk software still runs perfectly under the DOS, but any read/write request to the "Memory disk" will result in an immediate response. This is because it only takes a few microseconds to locate the wanted "track" and "sector" on the memory drive, as opposed to tens or hundreds of milliseconds on a mechanical drive.

HOW CAN WE ADD THE MEMORY?

Let's now consider the various ways of implementing the memory drive.

Page-Mode

The disadvantage of the RAMB/GM802 page mode is that it switches an entire memory board in and out of the system. Considering the case of GM802 (a full 64k board) this means that when a board is paged out, the running program (that is doing the switching!) vanishes with it. Therefore the control program has to be copied across to the same addresses in all boards in the page mode system so that it can continue to run. (An identical copy then appears in the place of the version that has just been paged out). If one of the paged memory boards is removed for any reason, the system will crash if it attempts to switch to that board as no memory will appear. Unfortunately there is no way to dynamically determine the presence or absence of a memory board in a particular page without crashing.

(Users with page-mode Memory drives may have discovered by now that they cannot Boot a CP/M system set up for a memory drive if one or more of the expected boards is absent).

The problems are not so severe if a common area of memory exists that is not paged, (e.g. the workspace RAM on a Nascom 2), as checks can be included for the presence or absence of paged (or banked) RAM.

The standard page mode supports up to four boards, giving the standard 64k of memory together with a 192k Memory disk.

Memory Mapping

This approach is neater than page mode, as the memory can be moved around (or remapped) in 4k sections, rather than the full 64k amounts. The 19 address lines defined on the 80-BUS restrict the total memory size to 512k byte, though it could always be combined with page mode to give a total of 2Mbytes.

The control software needs a little thought when moving data to ensure that there is no clash between source, destination, and the driving program. Also it has to cater for the case where a block transfer may straddle a 4k boundary (either source, destination, or both).

Ram-Disk

This finally leads me on to the concept of the Gemini RAM-DISK. Here the memory is not a system memory board, but is arranged as a block of memory separate from the 80-BUS address lines, which the CPU can only access via a few IO ports. To communicate with the memory, the CPU has to write an address to two IO ports, and then read/write data from/to another IO port.

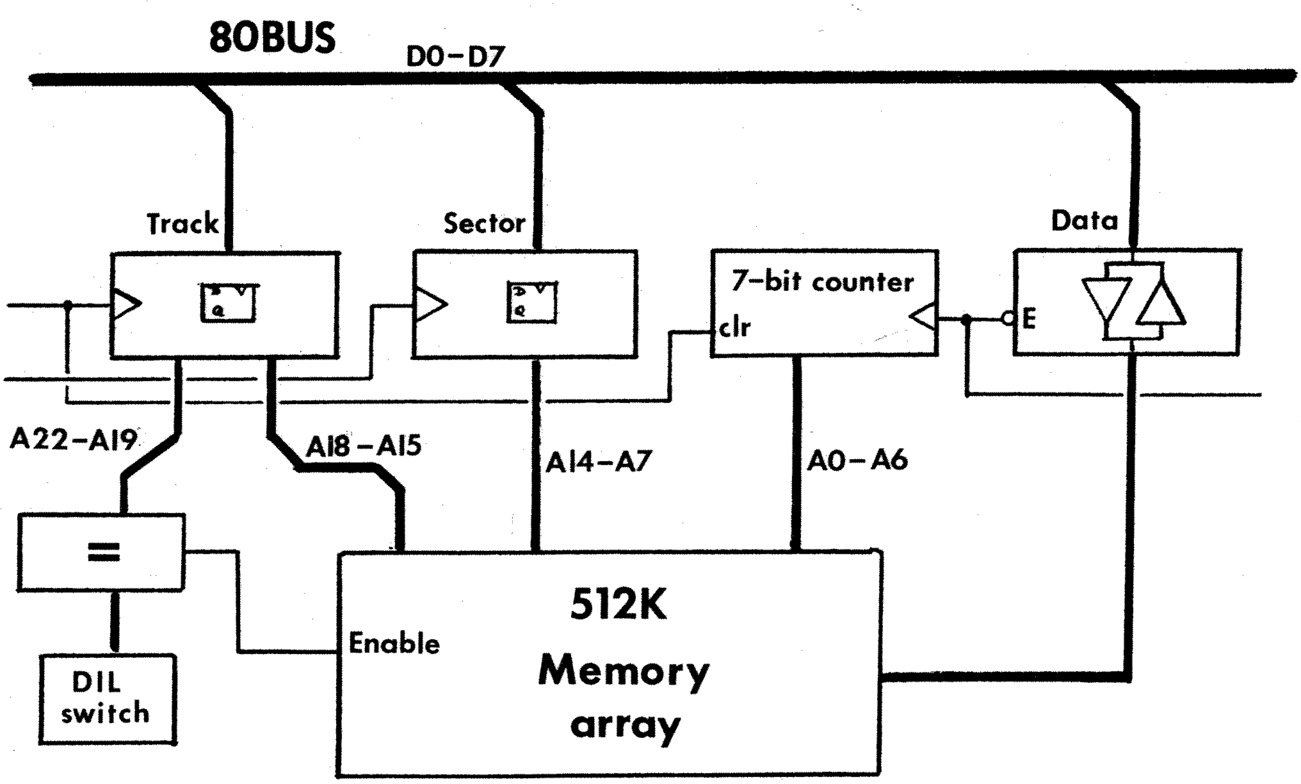


FIG 1

If the RAM-DISK memory was addressed on a byte-by-byte basis, the data transfer rate would be slowed, and the board would be rather clumsy to use. However Gemini have optimised the board for use with CP/M, and have made the interface disk-like. (See Fig 1).

One can regard the three IO ports as 'track', 'sector', and 'data'. The address applied to the memory array is made up of three components: A0-A6 coming from the seven-bit counter, A7-A14 from the 'sector' latch, and A15-A18 from the 'track' latch. In addition, the four high 'address' lines from the 'track' latch are compared with an on-board DIL switch to provide an enable signal to the memory array.

The seven-bit counter is controlled in two ways: Whenever data is written into the 'track' register, the counter is cleared. Whenever data is read/written to the data port the counter is incremented. Thus the memory array can be regarded as a disk of sixteen tracks, (the low four bits of the 'track' register), with 256 sectors per track, (eight bits of the 'sector' register), and with 128 bytes per sector, (the seven bits of address from the counter). So transfers to and from the RAM-DISK occur in blocks (or 'sectors') of 128 bytes each. A typical CP/M driver for the board would look like:

```
ld   a,(track)         ; Get track number
out  (track_port),a    ; Set it
ld   a,(sector)        ; Get sector number
out  (sector_port),a   ; Set it
ld   b,128             ; Sector length
ld   c,data_port       ; Point at data port
ld   hl,(dmaadr)       ; Set transfer address
inir                   ; Move data (OTIR for out)
```

- very simple and fast!

This way of adding extra memory, as well as being more economical in its requirements for support software, is also economical in hardware. In the case of a paged or mapped memory board, the memory has to be designed to work at the full speed of the shortest Z80 memory cycle - the M1 and refresh cycle. With the IO approach the memory array only has to meet the more relaxed specification of an IO cycle, thus allowing slower and cheaper RAMs to be used, and leading to a more reliable board with larger margins on critical timing paths.

GM833 - What you get

Now we've covered the principles - what about the product?

The RAM-DISK comes in the familiar Gemini packaging. On unpacking you find a ready assembled 8x8 board. The board is to the usual Gemini standard, with plated-through holes, silk screen component identification, and solder resist. The first thing you notice is that almost exactly half the board holds a dense array of ICs, the 64 64k dynamic RAMS of the RAM-DISK. The remaining ICs, (15 in all), handle the BUS interface, and the control of the array.

Accesses to the memory array, and refresh of the memory array, are controlled by the one large (40-pin) IC on the board. This is the Texas Instruments TMS4500A dynamic RAM controller.

Also included on the leading edge of the board is an LED, an activity indicator which illuminates every time the board is accessed.

A concise manual is included with the board, together with a circuit diagram. The manual follows the usual Gemini format, and gives a description of how the circuit works, together with a small section on software to drive the board. The software section is not extensive, and assumes that the board will be used in conjunction with a CP/M BIOS. (See below). It also gives an indication of how a BASIC program could drive the board directly.

PLUGGING IT IN

That is almost all you have to do! The board includes one mini-DIP switch, (4 pole), and two links.

The mini-DIP switch is used to select the board number in a multi-RAM-DISK-board system and, in most cases, will not need changing from its setting of 0. (Unless Gemini change their test procedures and they come set to some other value!).

LK1 will provide you with a NASIO signal if your system requires it, but most people will not need it, or will have another board in the system supplying it already.

LK2 allows the clock input to the board to come from either the CLK line, or the AUX CLK line. If you do not have a 4MHz system clock, you will have to cut the trace between 1 and 2 on LK2, and connect 1 to 3 in order to pick up 4MHz from the AUX CLK line. (Assuming it has been implemented on your system).

SOFTWARE SUPPORT

Gemini BIOSs Versions 2.3 and higher support GM833. Also the associated program CONFIG has been extended to include the board as an option when setting up the parameters for a "Memory drive". If you have an earlier BIOS, a BIOS update program (together with CONFIG) should be available through your usual dealer.

The BIOS supports the drive as drive "M", and is so arranged that it does not re-initialise the memory drive if you are forced to press Reset.

SIZE

The standard board provides a memory drive of 512k (or 0.5Mbyte) capacity - quite a respectable size. However if this is not enough for you further boards may be added up to a maximum of 16, so providing the full 8Mbyte capacity that CP/M2.2 will support. (N.B. 16 boards are beyond the capacity of the currently available commercial 80-BUS backplanes, and may also require added power supply capacity!).

I find the 512k single-board drive more than adequate. It lets me keep a reasonable amount a system software on the drive (such as Wordstar Overlay files) along with all the data files. You may like more if you are handling particularly large data bases.

GM833 in use - BENCHMARKS

To give an example of the benefits of GM833 a few Benchmarks are shown below. So that a reasonable assesment can be made of relative performance I have also included figures for 5.25" and 8" floppy-disk drives, and the Gemini Winchester disk subsystem. As always the figures should be taken as a guide only, as the figures for the floppy disk performance can be made to vary widely, depending upon where the files are located on the disk. I have attempted to position the files in such a way as to produce 'average' figures.

The first benchmark is an example of how a program can be transformed by the use of RAM-DISK. The program in question is TRANSLAT, a program that translates 8080 mnemonics to Z80 mnemonics. Who ever wrote the program made no attempt to optimise the IO - the internal input and output buffers appear to be one sector in size. The result is that the majority of the run time of the program is taken up by the disk drive moving the head back and forth between the tracks holding the input and output files. The RAM-DISK has no such physical problem!

(N.B. another such program nearly gave me a heart attack when I used it to compare files on two different drives. It briefly turned my system into an imitation of a machine gun as the heads on the two 8" drives alternately loaded and unloaded several times a second.)

Anyway here are the figures for TRANSLAT working on a large 8080 source file. The number in brackets is the approximate number of tracks that separated the source and destination files.

5" Pertec GEMDDDS format	8 mins	6 secs	(15 tracks)
5" Microp. GEMQDSS format	8 mins	40 secs	(40 tracks)
8" Standard Single density	6 mins	20 secs	(43 tracks)
R0201 Winchester	1 min	42 secs	(85 tracks)
GM833 RAM-DISK		17 secs	

The RAM-DISK figure is not a mis-print, it is just 17 seconds. This test is perhaps a little artificial, so the next benchmark is the time taken to PIP a source file from the Winchester to the destination drive, load M80 (from the drive under test), assemble it, then link and load it using L80.

5" Pertec GEMDDDS format	2 mins	34 secs
5" Microp. GEMQDSS format	2 mins	35 secs
8" Standard Single density	2 mins	39 secs
R0201 Winchester	2 min	00 secs
GM833 RAM-DISK	1 min	30 secs

Here the performance difference is not so marked, but it is still significant. (This shows that in this instance the majority of the total time taken was in actual processing time and not disk access time.) The floppy disks had an edge on the Winchester as they only started off with M80 and L80 on them. By comparison the Winchester was already holding 3.9Mbytes of data and programs, and the files created during the test filled in odd holes here and there on the disk, and weren't stored in successive blocks. Also the head was on average about 200 tracks away from the directory track for most of the time!

CAVEAT

Using the RAM-DISK requires a careful approach to work to ensure that at the end of the day all changed and new programs end up back on permanent storage (disk). Copying a file to disk as soon as it is changed rather defeats one of the benefits of the RAM-DISK, but it has to be remembered that the RAM-DISK is volatile. Though the Gemini BIOS does not obliterate files if you are forced to press the reset switch, it cannot protect you against accidental (or deliberate!), powering down.

This is the time when the programmable function keys of the Gemini keyboard come into their own. At the start of a session one (or more) can be programmed up to provide a backup command, (e.g. PIP A:=M:*.MAC^M), and then this key can be pressed at idle moments when you pause for thought (or answer the phone), and, finally, at the end of the session.

[Ed. - In one application that I know a RAM-DISK is being used, its volatility does not matter. The application requires as rapid as possible access to any record in a very large database. When it is started the program copies a number of index files to the RAM-DISK, and uses these to control access to the database. If the power fails nothing of importance is lost.]

ERROR PROTECTION?

One area open to debate is whether, with so much memory on the board, GM833 should incorporate some form of error detection/protection? The soft error rate of RAMS is very low, but here we have 64 of them in an array (plus an additional 8 in the system memory).

Error detection and correction would be an overkill in this environment, but a possibility lies in a simple parity check on each byte. (I note a number of the large memory expansion boards for the IBM PC now offer parity protected memory). This would add eight more dynamic RAMs to the board, along with the parity generation/check logic (assuming it could all be fitted on). It also raises the question of what to do in the event of a parity error being detected. Light a LED? Generate an interrupt? Halt?

I have used a board for some months now without being aware of any errors, and test programs I have written to exercise the RAM-DISK have run for over 48 hours continuously without finding any.

I think the parity check comes under the heading of - nice to have for peace of mind, but not essential. Statistically, the more Gemini sell, the more likely an error is to occur on one. Will you be the lucky one? (Anyone out there won the major prize on ERNIE yet?)

ALTERNATIVES

The alternative way of providing a memory drive is via the normal expansion RAM boards, either the GM802 or the MAP256.

Here is a small comparison table:

	GM802	MAP256	GM833
Max size	192k bytes	960k bytes	8M bytes
Operation	Page-mode	MAP extended page mode	RAM-DISK
Size/slot	64k	256k	512k
Expansion	64k units	64k units	512k units
Straps	Solder straps to enable page mode. DIL switch to select page.	Flexible, so strapping needs some thought.*	DIL switch
Remarks	Configured system won't run if board removed.	Configured system may run if board removed.	Configured system will run if board removed.
Cost/64k inc VAT.	£144	£82	£65

* According to RB "Don't think, you'll only get confused. Just follow the manual".

SUMMARY

The GM833 RAM-DISK is up to the usual high standard we expect from GEMINI, with the usual level of documentation, and if you have applications that are disk intensive then this is the board for you. It is also close to the ideal of a 'plug-in-and-go' board.

It is unfortunate that the board is only available with the full 512k of RAM fitted, as the resultant £450 price tag puts it more in the court of the business user, to whom time is money. Perhaps Gemini will offer a partly populated board sometime to allow a lower cost entry to the benefits of the RAM-DISK?

BOOK NOTES

by R. O'Farrell

S. Monger's report in 80-BUS News Vol.2 Iss.2 that I had been sighted in Amersham etc. for some reason reminds me of Moby Dick (cries of 'Thar she blows' and suitable whale music in the background).

Wonder was expressed that I did not appear to be carrying any books. Obviously Mr. Monger did not look inside the boot of my Volvo. Therein he would have found much that might surprise him, as this trip my bookbuying was conducted for the most part in the bookshops of the V&A and the BM, with digressions through bookshops in Canterbury and Winchester. There were few books of computer interest that took my attention.

In these notes, it is not my intention to produce the detailed, scholarly and exhaustive review proper to a learned Journal - nor could I do so anyway! Equally, I do not wish to adopt the cheap journalistic approach and seize on some small error or trivial slip, which is blown up out of all proportion, for the agrandisement of the reviewers reputation. My notes - and I do not claim them to be more than that - are personal and subjective comments, which I hope will be of use to others in drawing their attention to a particular publication. In the last analysis, you are the person who puts your hand in your pocket to buy the book, so the final decision must be yours. It is regretable that books are now so expensive. At a certain stage in any field, one reaches the situation where most books are going over and over the same ground. Sometimes a book will take a new and interesting path through this ground, reflecting keen insights on the part of author. More usually, a book will only contribute one new idea. If that idea or piece of information debugs a program or gives one the clue to how to approach a problem, is it not worth it?

Cogniscant of the feeling among some 80-BUS readers that there is too much written in the 80-BUS News about a certain operating system, I will start with some other books. Recently in a bookshop in Dublin, I espied a book on the "These books are slightly shopsoiled and need a good home" table. This was:

Computer methods for Science and Engineering, by Robert LaFara, published 1973 by Hayden, (U.S.A.) distrib. John Wiley.

This book is priced about £10, I think, but as I was giving it a good home, I didn't pay that for it!. It is a work using FORTRAN and flow charts to discuss the problems of numerical methods in computing. It deals with Interpolation, Taylor's Series, finding roots of an equation by a number of methods, solution of simultaneous equations by matrix methods, curve fitting, differentiation and integration, and smoothing methods. He suggests that readers would need mathematics through calculus. I agree - after a lapse of nearly twenty years I had forgotten how calculus makes one's head ACHE! In small quantities, it is a useful book to dip into for reference, the FORTRAN examples translating very readily into BASIC or Pascal. It is dated in that one would nowadays expect such a work to deal with the Fast Fourier Transform, but this doesn't get a look in, as it probably hadn't come into fashion at the time this book was written.

In Archaeology, the use of computers seems largely to hinge around a technique called 'Cluster Analysis'. This is a method whereby collections of disparate elements, as it might be bronze Axeheads, can be analysed into groups. Other applications for this method of analysis include pattern matching - as for example matching sections of tree-rings to the master Dendrochronological database for dating timber samples. The technique is described in:

Cluster Analysis Algorithms by Helmuth Spath, published by Ellis Horwood (John Wiley distrib.) at approx £18 hardback and £12 softback.

All examples given in this work are in FORTRAN, which translates easily to BASIC, but the author's background as a Professor of Mathematics assumes a certain mathematical background on the part of the reader, and a familiarity with Group and Set manipulation that has long since eluded me.

On the same subject, I draw the attention of interested readers to:

"Computer Applications in Archaeology 1974 - 1982" (continuing) mostly available from Dept. of Archaeolgy, University of Birmingham, at a price of approximately £2 per volume.

These 60/100 page Journals are the Proceedings of the Annual Conference on Quantitative methods in Archaelogy (or Computer methods or similar - the name is not always constant) and consist of the typescripts of most of the papers delivered. The fields are widely ranging - use of the computer as an excavation recording teminal, graphic recreation of pot shapes, cluster analysis of collections of (guess what?) Axeheads. They seem to use the computer for nearly everything but Word processing! If you are an archaeologist, try and get your hands on these - your University Library may have them. They will certainly give you ideas about the use of computers in Archaeology - perhaps you will be able to bring some of the flexibility of the 80-BUS systems into that field in return. Digression: consider how many professions depend on literary output. E.g. Barristers, Orthopaedic Consultants, Archaelogists, Historians. How many of these even consider using a 'Word processor'? Fools! To avoid argument, let me state here that it doesn't matter what an archaeologist digs up if he doesn't write it up. I know one orthpaedic consultant who spends more than half his time dictating reports and opinions on patients for legal proceedings arising from the accidents that brought them to his attention. The other half of his time he spends doctoring - he can even recognise patients from their X-rays - but that is another story. End digression.

One of the fields opening up is the use of the microprocessor in control of machinery. With our detailed knowledge of the intricacies of the Z80, and the powerful and reasonably priced CPU cards available to us, it is practical for us to consider their use in control applications, and perhaps even to advise on it. Consider what can be done with a CPU, a serial I/O, and a PIO. A simple control program can be blown in EPROM and then the CPU card can control almost anything. A marvellous read on this subject is:

Industrial Design with Microprocessors by S.K. Roberts, publ. Prentice-Hall Inc., costing approx £22.

This is a most enjoyable book on the philosophy and practicalities of using purpose built controllers for industrial applications. In addition to dealing with the hardware interfacing necessary, the author deals with the debugging and user friendlying necessary if such a machine is to work successfully. He deals with a number of projects based on his own experience, giving copious examples of what happened, and what went wrong, with the object of guiding you away from these sticky areas. In spite of the expense, I feel that this book should be on the bookshelves of every inplant engineer. The author takes a light-hearted approach to the problems, making the book enjoyable and easy to read, but never lightweight or trivial.

Now for three Z80 books. These are:

Z80 Assembly Language Programming for Students by Roger Hutty, published by Macmillan, cost approx £5.

This is a "slim volume" (127p) which deals quite adequately with the use of Z80 assembly language and an assembler. It would make a reasonable starting point for a beginner at machine code. In his treatment of the instruction set, the author deals only with the simplest of the Input/Output instructions, and mentions the interrupts, so this is not the book to buy if you intend to get into the interrupt setup very quickly, but for the beginner, a fairly clear introduction to assembly language.

Introduction to the Z80 Microcomputer by Adi J. Khambata, publ. John Wiley costing about £12. (330+pages)

This author has written a textbook on microprocessors and an associated series of processor specific manuals, of which this is the Z80 version. As I have not seen the major textbook, I cannot comment on it, but without any doubt, this book contains the best discussion of all the Z80 family peripheral chips I have seen, dealing with their programming and timing requirements. It should not be necessary to purchase the main manual if you had any experience in using the Z80 (or had read, marked and inwardly digested the 80-BUS News!). This book takes up the subject a little bit further along from where Hutty leaves it down.

A Z80 Workshop Manual by E.A. Parr, published Babani, £2.75, (184 p).

This is a 'paperback' sized book that gives as good a survey of the Z80 and peripheral chips as one could reasonably expect. It deals with the types of instructions, the architecture of the CPU, the addressing modes, instruction set, assembly language programming, and use of some of the Z80 family peripheral chips. It also must endear itself to us as it's examples of hardware configuration and monitor facilities are based on the Nascom - albeit with NASBUG monitor. I think this would form a good and very reasonably priced introduction to the intricacies of Assembly Language and the Z80 for the beginner.

A Practical Introduction to Pascal - with BS 6192 by I.R.Wilson and A.M.Addyman published Macmillan (approx £6)

This is the latest edition of these authors' book on Pascal programming. It includes the text of the British Standard for Pascal, which is interesting if only to read exactly how a language is defined. I note one surprising omission from the standard - during the discussion over the last few years leading to the adoption of this standard, it was generally agreed that the 'case of' structure should have an 'otherwise' extension to allow for the exceptional situation where the operator did not match a case-constant. This seems to have been deleted before adoption of the standard. The textbook is succinct and to the point, being based on the introductory lectures in programming in Manchester. It is liberally illustrated with example programs, and would make a good starting point to find out about the language.

So far, so good. I haven't mentioned CP/M even once. Now comes the denouement, as the Bishop said to the Actress! One of the problems with CP/M is that its manual - Digital Research's CP/M Operating Manual - was, in its earlier incarnations, absolutely and utterly incomprehensible. Its latest version (July 1982) is slightly better, but suffers still from 8080 mnemonics and 'clever' use of macros. This has given rise to a plethora of books on CP/M, all written with the intention of explaining what shouldn't need to be explained. It is with some of these that I propose now to deal.

To put before you I have six books on CP/M. These are:
Osborne CP/M Users Guide by Hogan, (286p) publ. Osborne/McGraw Hill
CP/M Revealed by Dennon, (180p) publ. Hayden (dist. Wiley)
Mastering CP/M by Miller, (c300p) publ. Sybex
A Programmers Notebook:Utilities for CP/M by Cortesi, (368p) publ. Reston (USA), dist. Prentice Hall
Inside CP/M - A guide for users and programmers by Cortesi, (571p) publ. Holt Rinehart Winston (USA) dist. Holt Saunders
System Programming under CP/M-80 by Hughes, (197p) publ. Reston (USA), dist. Prentice Hall

If you cannot (or will not) read 8080 mnemonics, then stop here. All these books admit to the existance of the Z80, but are written in 8080 mnemonics. These are nearly impenetrable - I find I can visually disassemble hex listings easier than understand these. Due to an accident of history, the 8080 type mnemonic has dominated the USA - very much to the detriment of the code produced. As the first reasonably priced and popular processor in the UK was the Nascom, it set a firm base for Z80 mnemonics. Oh for a book on CP/M using Z80 mnemonics!

As in all of the Osborne manuals, the **Osborne CP/M User Guide** gives a clear, competent discussion of its subject. It surveys all of the standard utilities supplied with CP/M, effectively being a rewrite of the supplied D.R. manual. It includes a full index.

CP/M Revealed is a 'hands on' exploration guide to this operating system, using the standard utilities. By means of demonstration programs, the author shows how to explore the visible and invisible portions of CP/M. He develops an interesting utility named COMMON to allow read-only access to files across USER partitions, and another to RESTORE an erased file.

In 80-BUS News V2 No2, Dr. Dark reviewed Miller's **Mastering CP/M**. Firstly let me ask "How dare Dr. Dark attempt to review a book?" I was incensed when I noticed his review in 80-BUS News V2 No2. Then I forgave him. After all, the poor chap must need to resort to almost any method to bolster his ego - and a discriminating, educated, discerning audience such as yourselves would not easily be fooled by the disjointed scribblings from his pen! For the experienced programmer, this is probably the best purchase, introducing as it does the concept of the Macro Library, and the use of the Macro Assembler. It is worth remarking at this point that the Microsoft Macro 80/Link 80 package differs in many small ways from D.R.'s MAC. The major difference is that the Macro 80 can understand both 8080 and Z80 mnemonics if one sets the right switches, whereas DR's assembler only handles the Z80 instruction mnemonics by means of macros.

A similar book is **A Programmers Notebook** by Cortesi. This sets out to introduce the experienced programmer to the use of the Macro Assembler, again using DR's MAC, to explore the facilities offered by CP/M. He constructs a series of programs to extend the standard DUMP utility, to PACK and UNPACK ASCII files, which can save a lot of space on a disc of text, INCLUDE to allow almost any program to include other files on disc as if they were typed into the file in full, and a MACREF, a cross refernce generator. I am unable to compare this with Miller as my copy of Miller has been on loan for the last few months. Try and see both of these books before making up your mind. My impression is that Cortesi's programs are more substantial than Miller's, but that Miller is more generally useful.

Cortesi is author of another book on CP/M, **Inside CP/M**. This is divided into a tutorial manual, giving quite a detailed description of CP/M, and an exploration of many of its features, though not to such a great extent as Dennon. The second part of this sizeable book is a reference manual, giving a detailed page by page description of all of CP/M's facilities. The miscreant who has borrowed Miller from me suggests that Cortesi is too verbose in his tutorial section, but to give Cortesi his due, when we were having a problem with a SUBMIT file, we eventually found the answer here - and nowhere else! For your information, I note that a SUBMIT file does not like blank lines. It simply won't run. Cortesi remarks (p132) that this bug has been reported several times, but no fix has yet appeared. In consequence of finding this piece of information, and sorting out the SUBMIT problem we were having, I feel very kindly towards Cortesi.

For our last CP/M book, we have Hughes's **System Programming under CP/M-80**. This book surveys, briefly and succinctly, the standard facilities of CP/M, and proceeds to deal with the problems of interfacing Assembly Language routines to it. In the course of the book, he develops LIST, a file printing utility, XDIR, an extended directory facility, SYSGEN, which is similar to the SYSGEN supplied with CP/M, but now at least you know what is happening, and he then proceeds to introduce some of the problems of writing and implementing a BIOS. I'm quite fond of this book, as the description of the standard facilities is quite succinct.

The availability of three Colour Graphics boards for the 80-BUS at reasonable prices has turned my thoughts to books on Graphics. Using my usual rule of finding out what has been written on a subject before setting out to reinvent the wheel, I've found two good books on Computer Graphics. These are:

Fundamentals of Interactive Computer Graphics by Foley and van Dam, (664) published Addison Wesley
Principles of Interactive Computer Graphics by Newman and Sproull, (541) published in paperback by McGraw Hill.

Both of these books cover substantialy the same ground. They are concerned with the methods used in Graphics display terminals, either in BW or Colour, low or hi res, and the solution of problems such as representation in three dimensions, movement of shapes in real time, perspective control etc. Foley and van Dam is more lavishly illustrated in colour than Newman and Sproul, but they are both well illustrated by line drawings. All of their demo programs are given in Pascal, which will allow them to be readily translated to almost any language.

Threaded Interpretive Languages by Loeliger, (250p) published by BYTE/McGraw Hill

Recent interest in Forth is reflected in this book, which contains substantially the entire code, in Z80 mnemonics, to allow implemention of a Forth Complier. I include it here as I recently discovered that the Graphics routines of the EPSON QX10 computer and some others have all been written in Threaded Interpretive Languages (Forth-type languages). Those who have seen the EPSON QX10 in action will realise that it's graphics are very powerful. Perhaps it might be worth looking into the use of such a language for similar purposes on the 80-BUS. The ideas contained in it, of the threaded type language, are quite different to the more conventional method of language design.

DATABASES

by D.R. Hunt

Some more about databases, and what to do with them. In the last part, we got as far as looking at the way a database file can be split up, into records, and each record into fields of a given length. By splitting up database records into fields of fixed length, this naturally means that the records must also be of fixed length (as a record is composed of fixed length fields). This is extremely convenient, as it is relatively easy for the programmer of a disk system to enable rapid access to any byte(s) into a disk file from a given starting point. Simply, this means that if a record is 50 bytes long, and we wish to gain access to the 75th record, then the starting point of the record must be: 75 x 50 = the 3750th byte from the start of the file. Admittedly the arithmetic which takes place inside the DOS is not quite as simple, as the disk is itself split up into sectors of fixed length which are in turn spaced around a number of disk tracks, but given a map of where the DOS has originally placed the file, it is not difficult for the DOS to calculate a track/sector address for any given record. This technique is very fast and called Random Access because it can pick up any random point within a disk. It is commonly used in database controlling programs.

A second method of data access, perhaps simpler to understand is the Sequential Access method, where a disk file is read in a sequential manner from the first byte, counting the bytes read until the correct place is reached. In a large file this can take a very long time if the required record is towards the end of the file. Random Access is therefore the prefered method of gaining access to any record when speed is important.

There are of course other ways of organising a database file, one of which is the 'free field' method. This may be prefered where the data to be contained in a record is likely to be of considerably different length. With the fixed field record, the record length must always be of the length of the maximum data it is to contain. This is usually fine for financial programs where money fields may be perhaps 10 bytes long, and detail fields perhaps no more than, say, 30 bytes. The utilisation of space within the records will most likely be greater than 70% and the wasted space is more than made up for in speed of access. The free field database on the other hand, may contain a record of one byte on the one hand, followed immediately by a record of a couple of K or more. The utilisation will be 100% in this instance as the length of the data determines the length of the field allocated to it. If such a file were constructed within fixed fields, the utilisation of space would easily fall below 50%, and on the basis that space must be allocated for the maximum length field, then the utilisation could end up as a few fractions of a percent. This would lead to vast acres of unused disk space. Note that 'free field' methods usually treat fields and records as one and the same, one record usually being one field long, although field delimiters can often be added as a further refinement.

The snag with 'free field' methods is, of course, finding the data. Sequential access is the only immediately possible method (I'm leaving record and field indexing till later). In this instance, not even the starting byte is known, so a sequential search has to be made for some key which will uniquely identify the record concerned. This may be a symbol not used elsewhere in any record, followed by a record number of known length (i.e. a fixed record number field within the 'free field' structure) or it can be a specific keyword put in by the user. In any event, a sequential search must be made of the file until the key is found.

Now it just so happens that DISKPEN/GEMPEN (they are one and the same except for the name) has recently had a major revamp. It now has the ability to execute overlay programs, and one such, called MAXiFILE, is a 'free field' data controller of the type described above. With a bit of lateral thinking, it is easy to see how a text processor can become a free field database. What are the major requirements of a database controller apart from its ability to find a given record? They are the ability to display and edit the record. What does a text processor do? It displays text and allows you to edit it!! So what does MAXiFILE do? It does the searching bit.

Lets make one thing quite clear, MAXiFILE will only work with the new PEN, that is type VG:3 and release 1.3 or better; and the new PEN will only work on computers fitted with the Gemini GM812 IVC card, that means Nascoms so fitted, Gemini Multiboard and Galaxy computers, Quantum 2000s and the Gemini based version of the Kenilworth Portable computer. Versions will soon be available for SuperBrain and Mimi computers. New PENs can be purchased from Gemini and Microvalue dealers at £50.00 + VAT, or upgrades to earlier PENs only from Henry's Radio at £15.00 + VAT on return of the original distribution disk to Henry's. MAXiFILE is one of several overlay programs available and is an optional extra at £20.00 + VAT.

Having got the commercial out of the way. What are the uses that MAXiFILE can be put? Well I've been using it for my letters, amongst other things, as it treats separate disk files (all my letters are saved as single files) as records on the disk. Having invoked MAXiFILE, it saves the existing work in hand and asks for the file names to be searched. The reply may be an unambiguous file name or may be ambiguous using the standard CP/M conventions. A list of files to be searched is displayed and MAXiFILE prompts for one of two ways to carry out the search (it also allows you to escape from MAXiFILE at this point, or to reenter new file names). With MAXiFILE there are two distinct and different way of carrying out the search, there is the straight forward 'find' and the rather more complicated and extremely powerful 'find by logical expression'.

The straight forward 'find' is simple, supply the key and away it goes. Now in my letter files the key would usually be the surname of the person I'm looking for, as this would be most likely to be found either in the name and address block or the salutation of the letter concerned, i.e. Dear Mr. Bloggs, etc. Of course, if I can't remember the name, then part of the address, or something in the letter will do. If I can't remember the name or address or what the letter was about, then MAXiFILE can't help either, as I might as well have forgotten that the letter ever existed, and certainly have no right to go looking for it. Anyway, MAXiFILE in the 'find' mode treats each file as a single record, and searches through for a match with the supplied 'find' line. As it searches each file it displays the name of the current file so you know how the search is progressing. If it finds a match, the file is loaded with the cursor pointing at the first occurence of the match within the file.

Several options are then open to me, to find the next occurence of the match within the file and if not found continue with the next file. To forget about this file and skip straight on to the next, to use this file, and to edit and resave it. To merge it with the file in use before MAXiFILE was invoked, or to forget the whole idea and continue with what I was doing before MAXiFILE was invoked. As I said, this mode of searching is very useful for finding letters, but could equally well be a cross reference of books or articles, recipes, or a sort of diary of things to do. In fact anything that can be found from a key. All useful stuff, but not half as clever as the next bit!!

MAXiFILE can evaluate logical string expressions. In other words, I can say, find the reference which contains the words "Jim" and "Fred" but not "Sid", or some such. It is equipped with logical AND, logical OR and logical NOT. The expressions may be (but need not be) enclosed either within quotes or within brackets, if the expression contains quotes then these should be enclosed in brackets, if the expression contains brackets then these should be enclosed within quotes. If the expression contains both brackets and quotes, then you're on your own!!. Proper precedence is given to the evaluation. Unlike the simple 'find' mode, the 'expression' mode will work on records within a file. Records are separated by ^L within a file, and the evaluation is carried out on a record by record, then file by file basis. When a match is found the start of the record is displayed, and the options open under the simple 'find' mode are again available. This mode is particularly useful in preparing text as it means that several things may be found and brought together by the scanning of existing files for chunks of text which do or do not contain certain keys.

A few other things about MAXiFILE, it is intended for use with text. It can't cope with files like .COM files, it doesn't crash or anything, it simply wastes time searching for something which patently will not be there. There are two characters it can't find, the '_' and ^Z, as the former is used as a string delimiter in Nascom versions of PEN, and the latter is used as a file delimiter by PEN. Although it uses sequential file searches its speed is impressive as it is written in machine code (none of your interpretive Basic here). I've timed it at about 90 seconds for about 320K of text, which is equivalent to about 80 A4 pages of text, there aren't many people who can read that fast.

So MAXiFILE embodies all the principles of a 'free field' database. The ability to create and enter data, to find it again, and to allow re-editing and re-saveing of the file. Really there is not much more that can be done with a 'free field' database, except perhaps to give it some arithmetic capability for summing results in fields, but this is difficult as the fields are not necessarily in the same place in each record. This sort of thing is much simpler with the 'fixed field' database. The other improvement is to provide some sort of indexing to speed the search, but this is really outside the scope of MAXiFILE. There is only 1K of it and what it does in that space is little short of a miracle. Apart from that, the whole philosophy of PEN and its overlays is that it be cheap, powerful and simple to use, and indexing records in a 'free field' database is neither cheap nor easy to use.

So having finished the discussion of the structure of a database file, where now? In the next episode we will look at indexing the data for faster retrieval and some of the more important features that may be incorporated into the database controller. All good clean fun, and as I only have experience with my own 'home brew' database controllers and dBASE II, I'll have to do a bit of reading up on some of the others around if I intend to be objective next time round.

Aunt Agatha's Agony Column

by D. Parkinson

First a "Thank you" to those of you who wrote in with messages of sympathy. It was obviously very late at night when the Editor put the last magazine together, and "Aunt Alice" was the only title that surfaced through the alcohol fumes. [Ed. - do you prefer this title?]

Denizen of Hell

It is strange how things resurrect themselves. Recently I've been rung about IMPs and IMPRINT, and I've also seen a letter on the same topic. I gather somebody somewhere has bought the remains of all the Nascom IMP printer and is selling the circuit boards and printer mechanisms as scrap. The IMP was best described as an early low-cost printer whose quality of output was poor compared to the current Japanese offerings. The mechanism used did not offer very good registration, and IMP printouts reproduced directly in magazines were easily recognisable by the drunken appearance of the columns. (The registration on the IMP I had was bad despite careful adjustment. The best I could do was to get it to line up at about every tenth column across the page - others may have fared better). However it did produce legible printout, which is far better than nothing! I trust current IMP owners will forgive the use of the past tense above.

The IMP was actually controlled by a Z80 microprocessor. The original IMP was sold with NASPRINT as the control program installed in the printer. Subsequently I wrote IMPRINT [1], a replacement control program for the IMP which offered enhanced features such as selectable unidirectional-bidirectional printing, and a graphics mode. IMPRINT was supplied in a 2716 EPROM as a plug-in replacement for NASPRINT. Installation was just a case of removing the cover of the IMP, (easier said than done!), carefully extracting the NASPRINT EPROM from its socket, and inserting IMPRINT in its place, taking care to maintain the same orientation of the EPROM.

Be rude & Interrupt

For those of you embarking on adventurous software/hardware projects (like trying to breathe life into the remains of an IMP), don't overlook the capabilities of your computer. For example when I developed IMPRINT I intially used the Nascom 2 to check the performance of the software in a non-destructive manner. In the IMP the print-head solenoids are driven via an output port and transistor buffers. The print-head has a maximum permitted duty cycle, and if this is exceeded the result tends to be a dead print-head and smoking drive transistors! An error in the software could have easily lead to the end of the project, leaving me with a totally useless printer. The answer was to disconnect the drive to the print-head during the development to prevent this happening. - "But then you couldn't see what was happening" I hear you cry - wrong, this is where the N2 comes in again. The output of the IMP's print-head driving port was connected to the PIO on the Nascom, together with a strobe signal. Three programs were then written for the N2.

a) An interrupt-driven routine which read a character from the PIO and stored it in a buffer in memory.
b) A program to initialise (a), enable interrupts,and then echo characters from the keyboard (or elsewhere) to the IMP.
c) A program to analyse and display the contents of the memory buffer.

Running (b) resulted in a buffer full of data representing the on/off states of the printer solenoids during a printing pass of the print head. (Every time the print-head solenoids were 'fired' by the IMP the N2 picked up the data via an interrupt from the PIO, and stored it in the buffer). Program (c) included an automatic check of the buffer to ensure that the maximum permissable duty cycle of the solenoids hadn't been exceded (by leaving one on for too long), and also displayed the printed text on the N2 screen using the block graphics characters to represent the 'dots' of the printhead. The cursor control keys -> & <- were used to scroll the 'IMP' line backwards and forwards across the screen.

So the moral of this tale is don't forget the interrupt system of your computer - you can do a lot with it, even use it to measure it's own performance [1]. Some more words on the Z80 interrupt system can be found in [2] and [3].

Printer Interfaces

Driving the IMP (or any printer) requires a suitable software and hardware interface. The IMP uses an RS232 interface, and optionally includes a TTL level handshake line. A handshake line, or printer handshake protocol, allows characters to be transfered to a printer at a rate sufficiently high enough to ensure that the printer is never idle. Various aspects of interfacing printers to Nascoms or Geminis are covered in [4] and [5]. (Don't be deceived by the title of the latter - it covers both RS232 and Centronics interfaces!).

Printers and Wordstar

Our esteemed book reviewer writes that he finds Wordstar + Epson FX80 slower than Naspen + IMP (there's that word again), and he's wondering about looking at Wordstar's printer drivers to see what's wrong. If he finds a solution I for one would like to know it. The trouble is Wordstar is a very powerful wordprocessing program. By powerful I mean that it does a great deal. It can handle a variety of printer types, and while printing a file it does some further processing on the line (printing alternate lines backwards for daisy wheel printers, doing incremental spacing, looking for superscript/subscript toggles, underline markers, etc, etc). It also prints in a spooling mode, the print being a background task, allowing the user to edit another file at the same time. The net result is that the file comes out slowly (I would guess at around 1000 baud equivalent rate). I have little enthusiasm for looking inside a program as large and as complex as Wordstar, and we can only hope that Rory can find a workable solution; mine, is to go off and make a cup of coffee.

By contrast PEN is a straight forward program, and, as it is not trying to do everything under the sun at once, it can zap the file straight out to the printer.

Blocking/Deblocking

I have received a request for an explanation of what the blocking/deblocking routines in the CP/M BIOS are up to.
First a few words from the BIOS manual: "All CP/M software transfers data to and from the disk in 128-byte 'chunks'. This is due to the fact that this was the sector size on the machine that CP/M was originally written for, (and is also a widely used IBM standard). It is only now with new technology and increasing packing densities that larger sector sizes become more

attractive. In order to achieve this and still be compatible with previous CP/M software, (and also to allow programs to maintain economical 128-byte buffers rather than larger ones), some software is interposed between CP/M and the disk drivers. This software maintains a physical sector buffer in memory (512 bytes in size in our case) through which all the CP/M data transfers are passed."

Associated with the buffer are some flags, and a record of which sector the buffer contains. (Drive number, track number, physical sector number). Let us start by considering what happens when the BDOS wants to read a logical sector (128 bytes of data). We will totally ignore writing for the moment. The BDOS starts by issuing drive, track and sector requests, followed by a Call to the BIOS Read routine. The Read routine starts by converting the CP/M logical sector number to a physical sector number. (It divides it by 4 as 512/128=4). Next it checks a flag to see if there is anything in the buffer. If there isn't, it jumps on to do the actual read. If there is, it checks to see if it is the same drive/track/sector as the current request. If it is the same, then the read can be skipped as there is no point in overwriting the buffer with identical data! Once the buffer is full of data, the flag is set to indicate valid data is present, and the buffer pointers are updated to the correct drive/track/sector combination. Finally the transfer of 128 bytes of data to the BDOS follows. To locate which 128 bytes, the logical sector sector number is reloaded, and the lower two bits are isolated, (the remainder when divided by 4). These are then used as an index into the appropriate quarter of the buffer, which is then copied to the requested destination.

Writing follows a similar pattern, but has various extra quirks. The main one is that before a logical sector can be written into the buffer, the buffer must contain the full physical sector. This is because the BDOS Write is only modifying one quarter of the sector, and the other three quarters must be maintained intact. Thus a Write might actually require a pre-read to load the physical sector into the buffer. However the efficiency of the system can be increased by defering the physical write, (following the transfer of the 128 bytes to the buffer), because the odds are that the BDOS is performing a sequential Write, and so will be writing to another quarter of the same physical sector on the next Call. If the assumption is wrong, then nothing will have been lost, but if it is right, then the time taken to do a physical Write will have been saved. However a 'Must Write' flag has to be set to say that the buffer contains unwritten data in case the next request is a read or a write of another physical sector. In fact the code of both the Read and Write commands does check this flag, and if necessary Writes the buffer to disc before reusing it.

The BDOS also passes some additional information in register C to the BIOS on every Write Call. If register C is zero, then it is a normal write. (i.e. the BIOS handles it in the manner described above). If register C is set to 1, then the write is to a sector of the directory. In this case the buffer should be immediately re-written to disc, the write must not be defered. (Note This is in keeping with the 'rugged' approach of CP/M, in making it difficult for you to accidently destroy directories by removing or changing discs at the wrong time). If register C is set to 2, then the write is to the first sector of a newly allocated block of sectors on the disc. This last one is another 'tweak' to improve system performance. When the BDOS is writing a new file out to disc, (or extending an existing one), it tells the BIOS (by setting C=2) everytime it starts on a new block of sectors. As the BDOS is writing to an area of the disc that contains no useful data, (it has only just been allocated to the file), the BIOS has no need to pre-read the sectors from the disc, and the system performance will increase as a result. However the BDOS only tells the BIOS on the very first sector of the area, and not on every sector. Thus the BIOS has to maintain a flag, (saying "I'm writing unallocated data"), and maintain a record of the next drive/track/sector expected. If the unallocated flag is set, it compares the next request against the stored values to check that the write is following in sequence. As long as this continues, it knows it can dispense with pre-reads.

As an example here are the results of doing a "SAVE 128 JUNK" on a Micropolis Drive-

As normal (No unnecessary pre-read): 5.6 seconds
C always forced to 0 on Write: 18.5 seconds
C always forced to 1 on Write: 57.0 seconds
Finally a Double density system, but with a physical sector size of 128 bytes. (i.e. no Blocking/deblocking) 11.5 seconds.

The latter is a bit artificial, as the timing figure can be varied widely by altering the sector skew, but I hope I picked a figure in line with the Gemini skew. Anyway it gives an indication of performance.

I trust that equiped with the above in one hand, and the relavent section of the BIOS in the other, you can make some sense of the blocking-deblocking code within a few iterations.

Preview Time

Coming in the next issue: Wait states on the Nascom 2. Using 2716/2732 EPROMs in the byte-wide sockets.

Reminder

This column is fueled by your letters, so write! [Ed. - fueled? Does this mean that you burn them to keep warm?]

References:

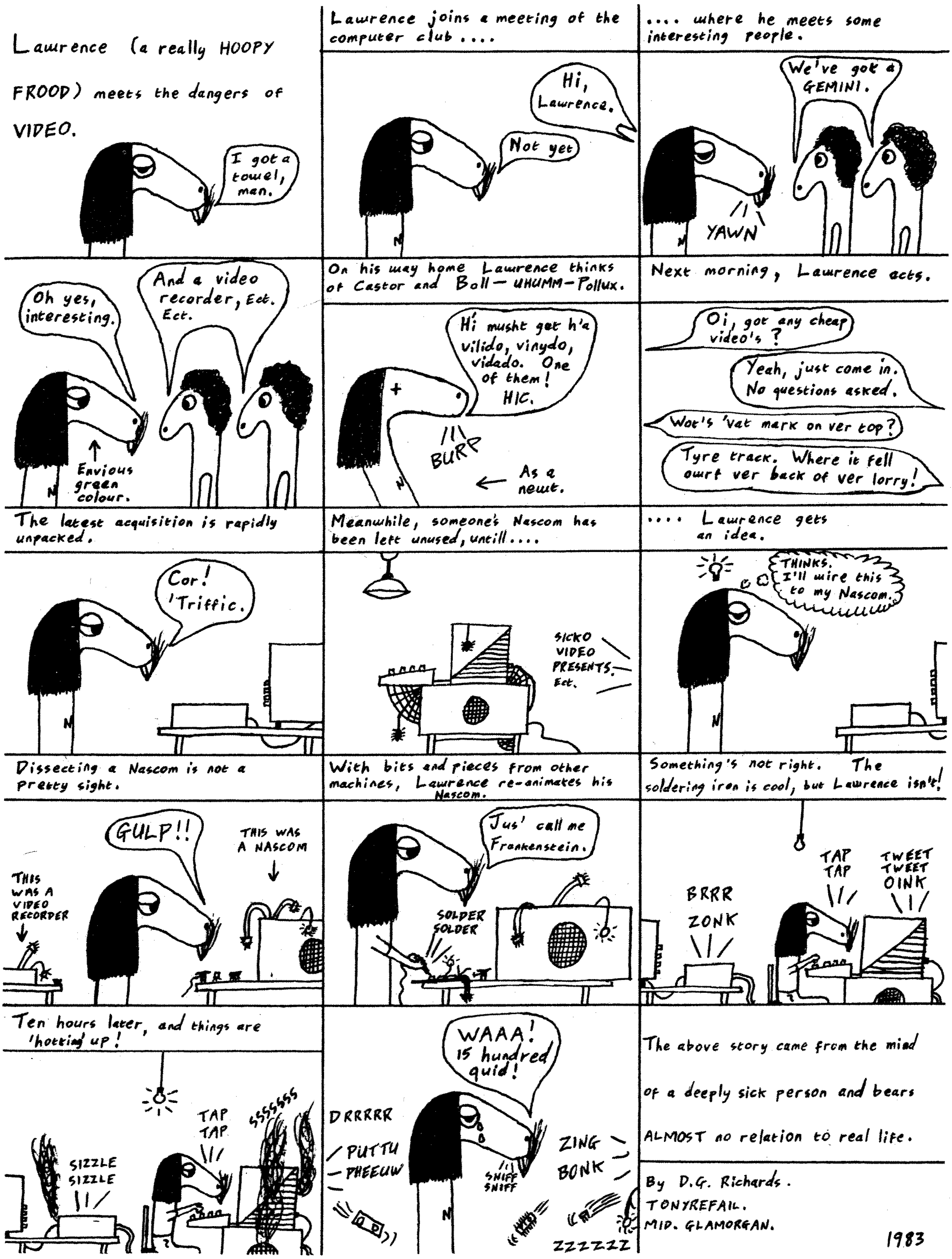
1. O'FARRELL R., "IMPRINT - a review", INMC80-4, May/Sept 1981, pp58-59
2. PARKINSON D.W., "Parkinson's Pep-up", PCW 3-10, Oct 1980, pp82-83,123
3. O'FARRELL R., "The Interrupt System of the Z80", 80-BUS News Vol 2 issue 1 Jan-Feb 1983, pp6-12
4. BEAL R., "Serial Interface problems made easy", 80-BUS News 2-3, May/Jun 1983
5. HUNT D.R., "The Kiddies Guide to Z80 Assembler Programming", 80-BUS News 1-3, Jul/Oct 1982

USER CLUB

The East Kent Computer Users' Club meets on the second Wednesday of each month in room 111/112 of the Computer Science section of the University of Kent at Canterbury. They are also affiliated to the Amateur Computer Club. The meetings take the form of a talk on subjects of common, followed by a somewhat less formal session in the university bar.

The membership contains only Nascom and BBC owners currently. For more information on EKCUC contact either:

Kevin Wood, 24 Hudson Close, Sturry, Canterbury, Kent. CT2 OHX. 0227-710720 or Laurence Fisher, 21 Manwood Ave, St. Stephen's, Canterbury, Kent. CT2 7AH. Phone 0227 65948.


Lawrence (a really HOOPY FROOD) meets the dangers of VIDEO.
I got a towel, man.
Lawrence joins a meeting of the computer club ....
Hi, Lawrence.
Not yet
.... where he meets some interesting people.
We've got a GEMINI.
YAWN
Oh yes, interesting.
And a video recorder, Ect. Ect.
Envious green colour.
On his way home Lawrence thinks of Castor and Boll—UHUMM—Pollux.
Hi musht get h'a vilido, vinydo, vidado. One of them! HIC.
BURP
As a newt.
Next morning, Lawrence acts.
Oi, got any cheap video's ?
Yeah, just come in. No questions asked.
Wot's 'vat mark on ver top?
Tyre track. Where it fell owrf ver back of ver lorry!
The latest acquisition is rapidly unpacked.
Cor! 'Triffic.
Meanwhile, someone's Nascom has been left unused, untill....
SICKO VIDEO PRESENTS. Ect.
.... Lawrence gets an idea.
THINKS. I'll wire this to my Nascom.
Dissecting a Nascom is not a pretty sight.
GULP!!
THIS WAS A NASCOM
THIS WAS A VIDEO RECORDER
With bits and pieces from other machines, Lawrence re-animates his Nascom.
Jus' call me Frankenstein.
SOLDER SOLDER
Something's not right. The soldering iron is cool, but Lawrence isn't!
BRRR
ZONK
TAP TAP
TWEET TWEET OINK
Ten hours later, and things are 'hotting' up!
SIZZLE SIZZLE
TAP TAP
WAAA! 15 hundred quid!
DRRRRR
PUTTU
PHEEUW
SNIFF SNIFF
ZING
BONK
ZZZZZZ
The above story came from the mind of a deeply sick person and bears ALMOST no relation to real life.
By D.G. Richards.
TONYREFAIL.
MID. GLAMORGAN.
1983

Private Ads.

Nascom 1 + 32K RAM, Teletype printer, 4800 baud cassette interface, floppy disc system. All PCBs and keyboard built into a Harris VDU terminal giving a professional appearance. Software BASIC and assembler on disc. All documentation. £450. Tel. Jim Taylor, Cramlington (0670) 731677 after 6pm.

Nascom 1, 32K, and Nascom 2, 64K, both cased, for sale either as working systems or in bits. Also 300 baud RS232 ICL Termiprinter with keyboard. All in good condition. Offers? Taylor, Congleton (02602) 2156.

Nascom 2, in a Vero rack with a 48K RAM B board, sound board, monitor, manuals, games. £300 ono. Also AVT monitor £60 ono. Kidlington (08675) 3750.

Nascom 2 cased with 32K, graphics, Nas-Sys 1 & 3, ZEAP, Nas-pen, port probe and software, spare cassettes, mags, books and manuals. £350 ono. 01-856-4287.

Nascom 2 cased, Gemini RAM 64K with 48K, BLS Pascal, Micro Power games and BASIC file handler, ZEAP, NAS-DIS, Debug, £400. Tel. 0642-597101 evenings.

IMP printer with 'IMPRINT'. Fully working. £140 or consider offers. Also 12" green screen professional quality video monitor, £50. Tel 0209-860-480. Clive Bowden.

Thomson-CSF 4-page VDU card, uses the SFF 96364 chip with 4K x 6 of dynamic RAM. Generates TTL video and sync. outputs, and has a UART with link-selectable baud rates, and RS232 line drivers and receivers on the card. All it needs is a parallel ASCII keyboard and a PSU to form a high quality 16 line x 64 character terminal. The card is new, boxed, complete with data, and is completely unused. £125 ono. Tel. Martin Davies, 0684 72178.