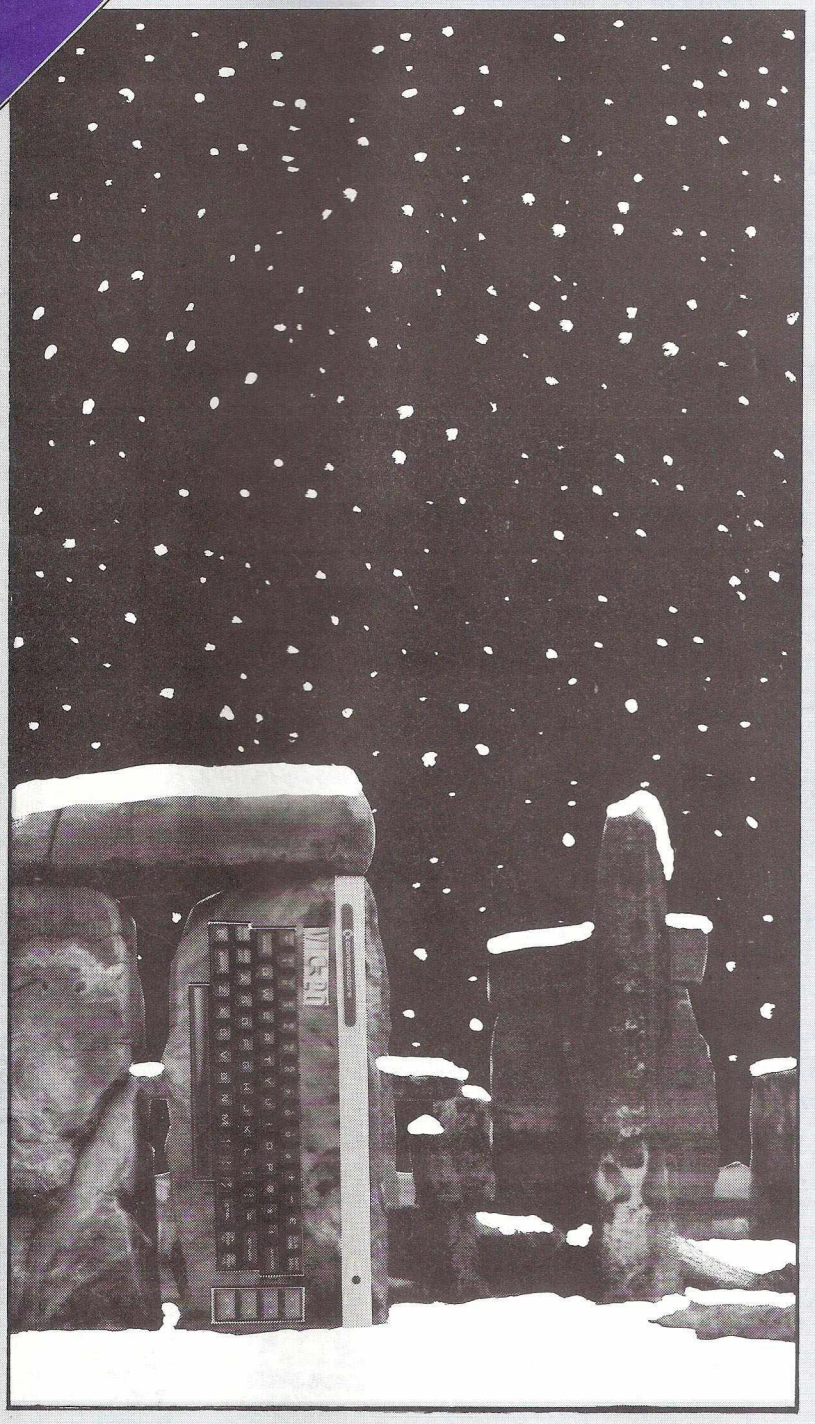
COMMUNICATIONS
COMPUTER NETWORK
AND
PROGRAM DISTRIBUTIONS

COMMODORE CLUB NEWS

December 1981



Volume 3 Issue 11

JIM BUTTERFIELD

A bumper bundle of articles

BULLETIN BOARD

Computer links by telephone network

DISK DRIVE USERS

Inspect sequential and relative files

CONCURRENT CLOCK

detailed programming guide

COMMUNICATIONS SPECIAL

Dr. Barker rides again!

commodore COMPUTER

"Every PET owner should read it"

Chuck Peddle, Inventor of the PET

"The PET Companion" is a fascinating collection of essential PET information from the pages of *Microcomputer Printout*. It contains **all** of the editorial from the 1979 & 1980 issues, including 105 PET programming hints and tips, 116 news reports, reviews of 54 peripherals ranging from light pens to printers and 27 major articles on PET programming. All of it written in straightforward English.

Some of the topics covered:

PROGRAMMING THE PET

Double Density Plotting
Modular Programming
Programming Style
Graphics
Subroutines
Sorting Out Sorts
Tokens
The Game of LIFE
Tommy's Tips
ROM Addresses

HARDWARE REPORTS

The New ROM Set
CBM 8032 SuperPET
CompuThink Disk Drives
Hardware Repeat Key
High Resolution Graphics
The Commodore Printer
How the Keyboard Works
AIM161 A to D Convertor
Commodore's 3040 Disk Drive
PET's Video Logic
Colour for PET: The Chromadaptor

THE SOFTWARE

Business Software Survey
Cosmic Invaders
Superchip
PETAID Do-It-Yourself Database
What's Wrong with WORDPRO?
Screen Display Aids
Keyboard Tutor
Photography Course
Who Do You Want To Be?: Fantasy Games
Commodore's Assembler Development System
Programming Aids & Utilities Survey
PET Games

THE SPECIAL REPORTS

PET in Education
PET Show Report
The Jim Butterfield Seminar
Hanover Fair Report
PET In Public Relations
Local User Groups
High Resolution Graphics
Commodore's New Technology
Future Shock: Forecasting The Future
Speech Synthesis
PET As Secret Agent
A Visit to the Commodore

plus news, letters, gossip and regular columns by leading PET experts.

To: Printout Publications, P.O. Box 48, Newbury RG16 0BD. Tel: 0635-201131

Please rush me a copy of the PET Companion

[ ] I enclose cheque/postal order/money order

[ ] Please charge my Access/Eurocard/Mastercharge or Barclaycard/Visa No:.....

[ ] UK £9.50 plus 45p postage [ ] USA $25 [ ] Overseas £12

(Credit card orders accepted by telephone on 0635-201131)

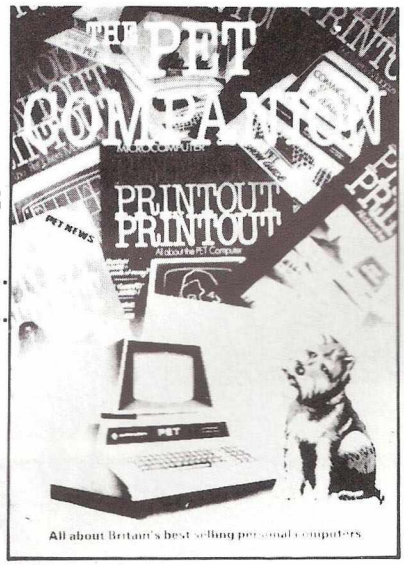
NAME: .....

ADDRESS: .....

.....

Postcode: .....

BARCLAYCARD VISA ACCESS



Contents

SPECIAL SUPPLEMENT

Next Month

Next month sees the return of Disk Use for Beginners (back by popular demand), as well as the next in our series of articles on how to write your own compiler. The story of how your editor bravely trod into darkest Birmingham to unearth a company with more than 60 PETs is also revealed.

Reviews next time concentrate on Commodore's new software product The Manager, and how one first time user coped with it.

As ever, Jim Butterfield will be regaling us with his own brand of computeristic wisdom, and we'll have the usual mixture of programs in Basic and Machine Code, for the beginner as well as the experienced user.

The special section returns it's attentions to the role of PETs in education, and features a report on how one school acted as a test site for the new SP9000. As well, we have the complete up-to-the-minute list of educational workshops both in this country and abroad, and also, for the first time, a list of regional, independent user groups, along with where and when they all have their local meetings.

See you next time!

Editorial

It's Christmas time again, so before moving onto this month's magazine, let me wish you all the best for Christmas, and the New Year. I hope you all have a good time.

First of all, let me apologise for the lack of the beginners guide to machine code in the last couple of magazines : it seems we're fated never to learn the intricacies of this, at first, daunting language! Our first series ended when its author, Paul Higginbottom, emigrated to Canada. When the American newsletter, Interface, started producing a guide, I started reproducing that guide here : however, over the last couple of issues of Interface the guide hasn't appeared, which left me in the position of having no article for you. My apologies for that, and, as last time, all I can say is that I'll endeavour to find another alternative source. We'll find out one day! I've tried to find out from the States what happened, but as yet have no news for you.

As ever, if you've any contribution to make to the magazine, big or small, I'd be delighted to take a look at it. In particular, an article on machine code programming would be most welcome : there's a considerable need for more understanding in this area, so for any program or article you've written, you can reach me at the address below.

The Editor
Commodore Club News
675 Ajax Avenue
Trading Estate
Slough
Berks

To subscribe, a cheque for ten pounds (or 15 pounds if you're overseas) will gurantee you a years (12 issues) supply of magazines, starting from the end of the month you send your cheque in : keep yourself in touch with the world of Commodore!

No more for this month, the magaine's big enough as it is. Don't drink too much on New Year's Eve!

Audiogenic Present New Software for the VIC

Enthusiastic Reading company Audiogenic (see back cover of this magazine for address and 'phone number) have just produced a new PETPack catalogue full of software for the PET, including new versions of many of the arcade games, which now work on the 80 column PET. They've completely re-designed the old Commodore PETPack Master Library, and now present an attractive range of inexpensive packages. Ask them for a catalogue!

However, the main reason for writing about Audiogenic is the large number of VIC programs that they're selling, the vast majority of which will run on the standard VIC without any memory add-ons etc. Hopefully the catalogue descriptions I saw will change : Americans are all right, but my goodness their catalogue descriptions are horrendous! I quote just one example, from a game called Satellites and Meterorites. "Your stalwart defense may take you near the greatest danger of all - THE BLACK HOLE. You realise the gravity of the situtation." Thankfully, the game is a lot better than the write-up.

Arcade Games

As was probably inevitable in the early days of the VIC, the programs tend to concentrate on VIC impersonations of many well-known arcade games. The ever-present Space Invaders of course, given a new twist (and a new title) by appearing this time in 3D. Other games with exotic titles include Spiders of Mars : no, nothing to do with David Bowie, you are a Space Fly trapped by a horde of Martian Spiders, and desperately trying to escape. Amok features a lot of robots, and "a legless, bouncing glob", whatever a legless, bouncing glob may be : a glob that's had one over the eight prehaps ?

Pac It In, a version of the currently popular Munchy Men (or whatever it's called in your area), Robot Blasters and Astro Transporter complete the game line up. All use full colour and sound of course, but the one thing I can't tell you at the moment is the price : all will be revealed in due course.

The othe programs tend to concentrate on the educational side of the VIC, and there are a couple of programs for teaching spelling, grammar etc.

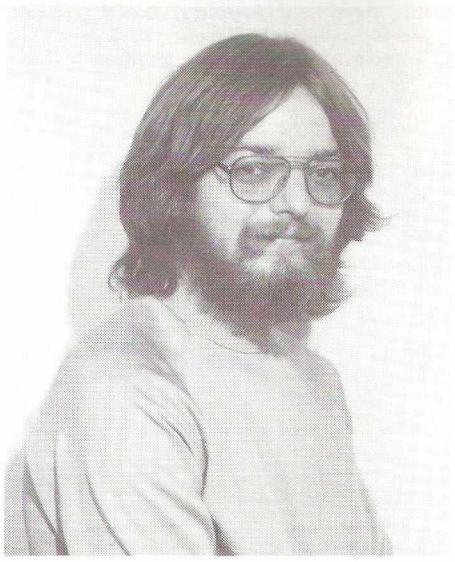
Finally we come to 2 programs by a gentleman who appears to be something of a VIC programming entrepeneur, namely one Len Sasso. ViCalc, the first in a series of calculator programs for the VIC, has ten memory registers and four stacked data registers all visible on the screen at all times. Basically, almost Visicalc but not quite! All calculations are performed instantly. His other program is VPM, or VIC Securities Portfolio Management. As might be gathered from the title, it looks after stocks and shares.

So there you have it : a whole new series of programs for the VIC.

Job Vacancies

To round off with Audiogenic for the month, there are currently two vacancies going there. The first is for a Salesman for VIC and PET software, and the other is for a Production Co-ordinator at their plant in Reading. For further details of both jobs contact Martin Maynard, director of Audiogenic, on Reading (STD Code 0734) 595647.

As a footnote to the Audiogenic story, their resident superstar Martin Maynard recently flew over to Los Angeles : just cruising down to L.A. for the weekend, as superstars are wont to do from time to time. The purpose of the visit was to find out about new VIC software, and bring back a whole range of goodies to look at.



Again, a mixture of arcade games, educational programs and home management are the name of the game. Probably the most exciting development is a product called VICTERM, which, as you might surmise from the name, turns your VIC into a terminal, and allows you to log onto Bulletin Board (amongst other things) via the telephone lines. Another aid, as opposed to a programme is a little gem called BUTI (Basic Utility ROM), which is essentially a VIC equivalent of the world famous Toolkit. As Martin himself said, the original chip BUTI. Sorry ...

Home Management

Onto home management, we find programs such as VICAT, short for Viable Catalogue. This is a tape based data base system, and can handle up to 99 files. Similar in name if not in nature is VICHECK, a cheque book management program. It seems that more and more of this type will be appearing as VICs begin to increase their user-base. At present the majority of the software is coming from the States and Japan, where VICs have been around for a while, but we expect Britain to catch up very shortly.

Education will again be an area where, at present, most of the software is coming from outside, but with an increasing amount coming from the U.K. as time goes by. Some of the American programs from Audiogenic include Super Additon and Super Hangman, where, according to the catalogue description anyway, the victim is hung against a setting sunset. Sky Math and Sky Division are two further programs, which are a mathematical equivalent of Space Invaders. Numbers appear across the top of the screen, and you've got to shoot the correct numbers down.

Games

Finally, onto the games. As usual, some of the names in their original American form leave a lot to be desired. Master Whip, for instance, conjures up somewhat bizarre images to say the least. In reality it is quite tame however, being a version of the famous Mastermind. Kiddy Checkers, whilst not being quite such a bad name, lacks that certain flair somehow. Still, you can at least guess what the program's all about.

Blastoids, as you might probably guess, is a version ot the popular Asteroids, where hordes of space debris fly around the screen and you've got to both avoid and destroy them, whilst doing the same to the little space ships that appear from time to time. Amazein again is fairly logical : a maze game.

Last, but certainly not least, the game of games, a little gem called Alien Blitz, which Martin assures me is "the greatest game of all time". I'll be bringing you a full review in due course, in which we'll put this claim to the test!

Additional news on Wordform

One of my complaints about Wordform, in the review in the October issue of the newsletter, was the fact that if you were a very fast typist you would occasionally find yourself getting ahead of what was appearing on the screen. This has now been cured, and that cure is amazingly simple. The POKEs listed below will solve all problems.

For 8000 series machines,
POKE4659,6
For 4000 series machines,
POKE4657,6
For 3000 series machines,
POKE4653,6
and then re-save the program. This then keeps up with even the fastest touch typist.

The reason why only one POKE is required is that originally a delay was put in to stop the average slow typist (i.e. me) who occasionally hit a key too hard, from getting two of each letter on the screen. The above POKEs remove that delay.

Another comment I made was the lack of Wordform's ability to print form letters, i.e. having the form letter in the main text area, and all the personalised information (name, address, Dear Bob etc.) lying in extra text (or storage buffer). Well, I did Landsoft a disfavour there. Having spoken to Ted Landsler of Landsoft the other day, I've now been shown how to perform this particular aciton, Of course, it's very easy, and it's in the manual if I'd burrowed far enough into it. Sorry Landsoft : it makes Wordform even better!

The author of Wordform was recently taken to our distributor in Israel, on an all expenses paid trip, to convert Wordform to Hebrew! The flight was paid for, accomodation paid for, the entire works. And, of course, he also got a fee for doing it! Having completed the conversion it now runs succesfully in Hebrew. Does this mean it runs backwards (or should that be backwords ?!)

I'm also informed that Wordform 2, a distinctly up-market version of the program, will be available from January 1982 onwards, for something under 200 pounds. It is claimed that it does it all. We'll let you know!

Reviews : Invaders in ROM, from Supersoft

A parcel arrived in the office the other day, containing yet another ROM! ROMs abound these days, and the sight of another one was enough to drive one to despair. However, despair was not to be the case : this ROM replaced one of the existing PET ones, and so didn't mean continual swopping of Toolkits, Powers, Faster Basics etc.

The ROM that bit the dust was the UD7 one, and with the insertion of the latest Supersoft ROM none of the PET's normal functions were lost, and a very important one was gained : the ability to play Space Invaders, accessed with just one command!

The idea of changing one of the PET's ROMs for a custom built one is a very good one. Generally you will find a fair amount of space in any of the ROMs inside the beast, and it seems reasonable to put the space to its optimum use. This removes the need for Spacemakers, Rompagers and the like, whilst keeping the PET (at least as far as it is concerned) to all intents and purposes as a normal PET.

Thus all the programs that you'd been used to using before will still quite happily run in the normal fashion, but if they're written in Basic, and the STOP key is still enabled, you can break into your program, type SYS 59648 and hit RETURN, and hey presto! Invaders is at your command.

This is best done when no-one else (management preferably) is around, since the program uses the 8032's (the only machine for which this chip is currently available, for a very important reason) internal speaker to generate the usual infuriating space game noises.

Better yet, the program uses the whole of the 80 column screen, to give us the first commercially available arcade game on the 80 column PET. Its use of graphics is very good, and the game as a whole is quite exceptional, considering it all fits into

about 2K of code.

To get back into your original Basic program, at any point in the game all you've got to do is hit the left arrow key, and there you are! Clear the screen, type CONT and away you go again. From dreary accounting packages, to a few minutes of joy blasting aliens to smithereens, and then back to the accounts program when the boss re-appears. All it needs now is a link to Wordpro and Wordcraft and they're away!

What else can I say ? A superb game, and at only 19.95 it looks like Supersoft have got another winner. 'Phone 01 861 1166 for further details.

Technical Software

The Technical Software Centre, based at BHRA Fluid Engineering, is carrying out a survey of technical software for desk-top computers on behalf of the National Research Development Corporation (NRDC).

It will identify programs written by individual engineers and designers to solve particular technical problems with a view to assessing and adapting suitable programs for general use.

Suitable programs will, after testing, be produced in TSC's TEC-PAC format as Commodore Approved Products.

Authors and users of 'in-house' technical software written for Commodore computers are invited to contact Tony Swann, Senior Engineer, at:

BHRA Fluid Engineering,
Cranfield.
Tel: 0234 750422

Attention All Software Editors

(The following letter should speak for itself)

Dear Mrs Gulliford, (CPUG)

Having recently been appointed Programs Editor of "Personal Computer World", my first job is seeking out original programs for all sorts of micros, not only in Basic but also in Pascal (as you are possibly aware we do not publish Machine Code as the listings are too long. Assembler is covered in PCW Sub Set).

It is for this reason that I write to you. Although we do get a fair number of programs from our readers, a lot of them are full of bugs and still more are games or applications that turn up over and over again. I'm sure that among your members you must have at least a few talented programmers who would like to have software published — naturally we pay for it at good rates — and in view of this I'd like to hear from them.

I would be grateful if you would mention to your members that I am looking for as much good amateur software as I can get on behalf of PCW and that we also have a paid referee register going for those (who know what they're at) who would like to check programs and provide listings from cassette and disks. Anyone interested is more than welcome to write to me at the address below.

Maggie Burton
Personal Computer World
14, Rathbone Place,
London W1P 1DE

VIC meets the Stars

Despite splitting the proton a while ago, myself and Chris Palmer (VIC Centre supremo) decided once again to boldly go where no man has boldly gone before (Warwick Avenue tube station), and set the controls for the heart of the BBC Radiophonic Workshop : the place where most of the wonderful soundtracks for the BBC are produced.

Unlike our last jaunt with the BBC, somewhere in the wilds of Ealing, this journey presented no problems in the form of getting up early. Indeed, I ended up leaving Gerrard Acres later than I normally do for work, and had great pleasure watching everybody else squelching to work through the rain whilst I remained in the warmth listening to the Blue Ridge Rangers and drinking my cup of coffee. By the time I had to leave to foray forth to Taplow station the rain had decided that it had had enough for the day, and all was nice and dry.

From Taplow station to the VIC centre was a model journey, helped by the fact that the train, like myself, was somewhat late in arriving at Taplow. Perhaps, as someone once said, if British Rail re-scheduled all their time-tables by 13 minutes all their trains would run on time. But I digress.

The VIC Centre was very easy to find, and conveniently situated next to a pub and a betting office. I wonder who decided it's location ?! The front half of the shop is the actual showroom area, filled with VICs and a very vast array of books on microcomputers : programming them, choosing one, and so on. If you're interested in visiting the VIC centre, they're at 154 Victoria Road, in Acton, London W3, or give them a ring on 01-992-9904.

Around the back of the showroom is the area where all the work gets done : cigarettes and cups of coffee abounded. A mixture of VICs and PETs, all intgerwoven with an array of disk drives and printers, were much in evidence. Also much in evidence were ringing telephones : they hardly ever stopped! We managed to escape however, and set off for the readiophonic workshop.

Arriving at Warwick Avenue, we discovered where taxis go to die. There were hundreds of them, all driveless, moping in a large group near to the station. One of them eventually moved, and we duly arrived at the BBC.

And what, you're probably asking by now, was the purpose of all this ? The BBC just acquired a fabo synthesiser, all 15 grands worth of it, and we thought they might be interested in a low cost micro to produce some of the lesser sounds and thus leave the big machine free for the work it's intended to do. So, we gave them a demonstration of a VIC making various burbles and gurgles : we put it through its paces. When we mentioned speech synthesis they were all impressed, but then we said we hadn't got one there to demonstrate, so we were back to square one. Then we mentioned SID (Sound Interface Device), and they were all impressed again, and then we said we hadn't got one to demonstrate, so back to square one again.

We promised to come back again when we'd got the various extras, and they agreed that they would be very interested in VICs' but as it stands at present they've got more than enough machinery to cope with whatever noises they desire : speech synthesis, and something like SID at the price, they most certainly would want. We will return!

After our demonstration we were treated to a tour of the workshop, and saw more synthesisers than I've ever seen in my life before. We also heard some harrowing tales of productions in the past. To quote one example, the second radio series of Hitch Hikers Guide to the Galaxy was originally going to go out one episode a week, for seven weeks. This was

fine, all the scheduling was done and people started getting on with the work. Then, a change of plan : it was going to go out on successive nights, and seven weeks work had to be done in seven days! By virtue of not sleeping for three days the final episode was finished half-an-hour before transmission.

And so, as in all these expeditions with young Lim-Bim-Wim-Bim Palmer, we repaired to the pub for lunch : a pint of Burtons and some food (note the order!). Another mission accomplished.

Our other encounter with the stars, and to prove that we're unbiased, was a visit to ITN, just off Oxford Circus. Long after deciding that Oxford Circus was not the most intelligent of places to meet someone (five exits and half a million tourists) Chris Palmer and myself finally met up, and retired to a hostelry for lunch, and to plan our campaign for the afternoon.

After losing lots of money on the local version of Space Invaders (Centillion, for those of you in the know), we went over to ITN, and arrived to be greeted with the words "Do you fancy a drink in the ITN Bar ?" Never ones to say no, we trotted off to the bar in the hope of meeting a galaxy of stars, talent scouts etcetera, cursing the fact that Anna Ford had left ITN. Well, I'm still an editor, and Chris is still working for the VIC Centre, so I think we can safely assume that we're not going to be 'discovered' this year. We did encounter fruit machines with jackpots of 100 pounds, but only succeeded in losing yet more money.

The purpose of this visit was a meeting of the ITN Computer Club, and a demonstration of the VIC. Apparently (we were told afterwards) the meeting had been visited by a couple of ITN's managing directors: an unheard of event, seemingly.

I think we scored with them, as the demonstration went very well, and most of the questions fired at us were fielded admirably. We spotted the potential troublemaker (There's always one!) within seconds of starting, but in the end he was turned to our advantage, by the simple expedient of answering his questions, however technical and complex they got. In particular, he even started talking about some computer he'd got that could do this, that and the other, whereas the VIC couldn't. This was all very well, I suppose, but when we pointed out the fact that his machine cost around ten times as much as the VIC, even the rest of his colleagues started to laugh. Fifteen-love, as they say.

We gave a run-down on sound, colour, graphics, altering characters and all the other features of the VIC, and in the end convinced them that as a low cost introduction to the world of microcomputing the VIC was the machine to buy. Mind you, we were fighting a battle; quite a few of the people there had made their own computers, and were quite naturally slightly biased in favour of their own particular, individual machine. We even managed to convince one or two of them that the VIC was a useful machine.

So out of our two visits, we acquired one success and one possible success on a future visit. Not bad really, and thus suitably impressed by our work we took the only course of action possible; we got ourselves a drink!

Do you want to advertise Second-Hand equipment?

Name .....

Address .....

.....

.....

Equipment .....

Price(s) .....

Please fill out the form above if you wish to advertise second hand equipment for sale in Commodore Club News. Entry costs just £1.00 (cheque or postal order), and return it to :
Peter Gerrard
Commodore Business Machines
675 Ajax Avenue
Slough, Berks.

C.B.M. (U.K.) Ltd do not accept any responsibility for the products advertised hereunder and prospective purchasers should satisfy themselves in respect of any representation made.

Second-Hand Corner

The following advertisements have been received:

Dual Drive Floppy Disk CBM 3040, fitted with up-to-date DOS + software. Price £750. Taher Mahmud, 41, Comely Bank Road, Edinburgh EH4 1EJ. Telephone: (031) 332 6406

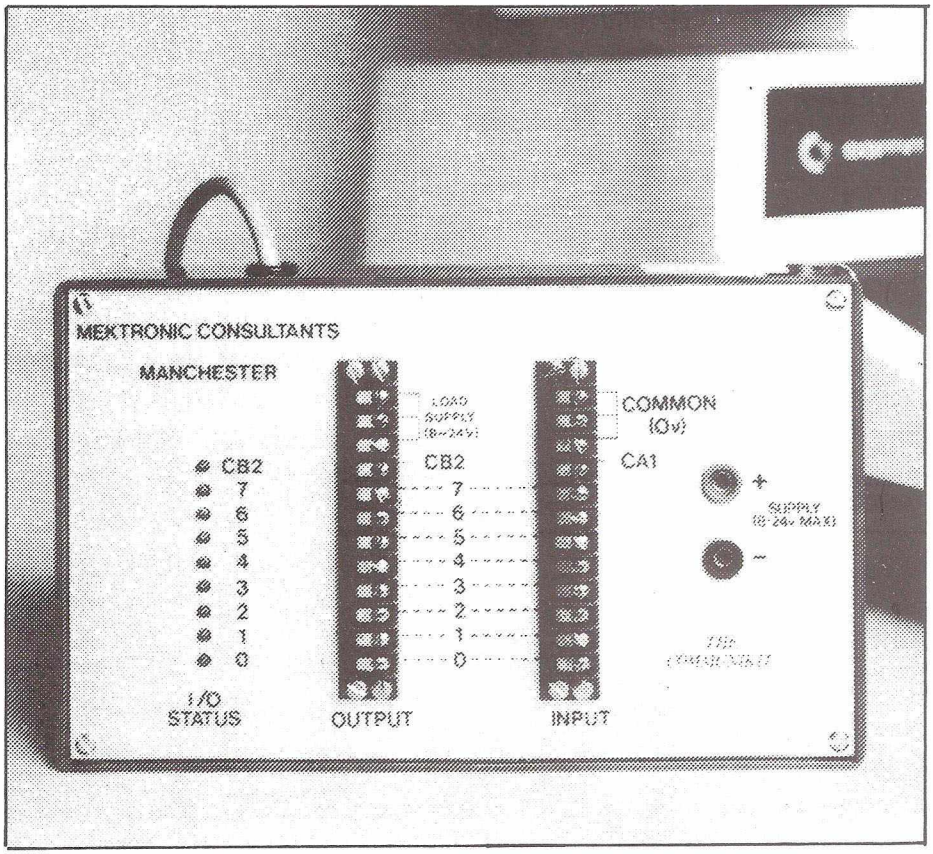
Midlectron M62 Papertape Punch/Reader as new. Run only for testing. Price £800. K.A.Mander, 17, Beacon View, Marple, Stockport, Cheshire SK6 6PX.

Wanted — 3000 Series computer and/or Floppy Disk Drive and Printer. Frank Pickles, 13 Central Drive, Bournemouth BH2 6LQ. Tel: 0202 293650.

Commodore Tractor Printer 3022, Price £250. V.E. Marshall, 33, Turmore Dale, Welwyn Garden City, Herts AL8 6HT.

Using the User Port

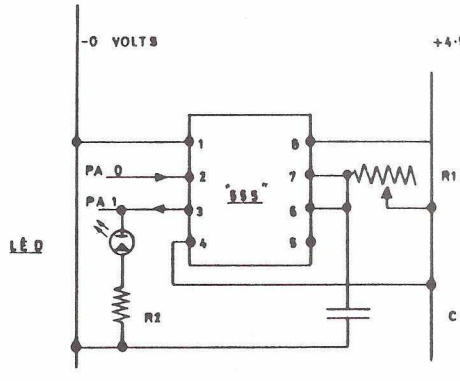
Continued from Volume 3, Issue 10



My next project was a simulation of chemical process control. I assumed that a chemical reaction heated up at a rate of 10 degrees per second and simulated this with a FOR/NEXT loop with a T=T plus 10 condition so that the temperature T was printed every second. If T exceeded 110 degrees an explosion was assumed to occur (and you graphics experts can have a field-day!). To control the reaction a coolant pump can be switched on which causes the condition T=T -10 to replace T plus 10 so that the printed out temperature now drops. If it goes below -10 then the mixture freezes (more graphics). Correct operation of the switch enables the reaction to continue.... The status of the switch was read by PEEK(59471) = S. If S was equal to 254 then cooling was assumed to be taking place and T was set to T-10.

I then decided to use the Communikit to check the operation of a "555" timer. This simple and cheap device is very useful and easily obtained. The circuit I used is given in Figure 6, the LED is merely present to help monitor what is happening. For the same reason I slowed the program down with FOR/NEXT loops and kept the program as simple as possible.

Figure 6




```
10 REM:::TIMER PROGRAM:::
20 N=59459:X=255:Y=59471
30 POKEN,255:POKEN,X-1
40 FORI=1TO1000:NEXT
50 POKEN,X:FORI=1TO500:NEXT
60 B=TI
70 IFPEEK(Y)>1THEN70
80 C=TI
90 PRINT"TIME ELAPSED = ";C-B
100 GOTO20
```

A puzzle from PA O (output) was used as the trigger to start the timer while the output from the timer was fed to PA 1 (input) and monitored as in the previous program. To my great surprise the system worked immediately (NOT my usual experience with electronic circuits) and I checked the time difference found by the PET with the time by theory. Since it was not in very good agreement I wrote a program so that the test was repeated 10 times and the result averaged. I then tried checking several different resistors and soon found the source of the error. It was simply the plus or minus 10% tolerance expected of a cheap resistor! I had therefore calibrated my test capacitor. The obvious next step was to try several different capacitors against my resistances .... and so on. I then had a set of calibrated R's and C's which I arranged so that they could be switched in at will using an 8 position DIL switch and I had made my own multimeter. What next? Any devices which showed a change of resistance could be incorporated, thermistors, an ORP 12 (a light dependent resistance) or a level indicator based on conductivity changes were first to come to mind.

Since I knew or could easily find out the approximate resistance of my device all I had to do was switch in the appropriate capacitor and I was away. My thermistor had a room temperature resistance of about 60 K ohms which dropped to around 16 K ohms at 50 degrees Centigrade. I wrote a program to represent the time difference as a bar-chart so that I had a recording thermometer. (see "RECTHERM") (overleaf).

The ORP 12 had a very high resistance in the dark and very low resistance in normal electric light so it was very easy to detect if a light was turned on (burglar alarm ?). I found that it would detect the light from an L.E.D. and could even "see" the difference in brightness between the print and dark parts of the video screen or between the light from a black or white strip of paper. I also could make a light pen it seemed.

And finally a project still (appropriately) on the drawingboard....a drawing-pad. This could consist of a sheet of special resistance paper or an ordinary sheet of paper soaked in salt

solution. Electrodes in strip form would be placed along two edges at right angles and connected in turn to PA 1 and PA 2 (PA 0 would still be the trigger). Each connection would measure the resistance between a stylus and the electrode connected. The resistance is a measure of the distance of the stylus from the electrode or in other words the X and Y co-ordinates of the stylus postion. This position would then be plotted on screen and the stylus moved on. The principle could be tested with a BASIC program but to be of any real use machine code would have to be used.

RECTHERM (RECORDING THERMOMETER)

```
5 N=59459:X=255:Y=59471
10 POKEN,X-1
20 FORI=1TO1000:NEXT
30 POKEN,X
40 FORI=1TO500:NEXT
45 B=TI
50 IFPEEK(Y)>1THEN50
55 C=TI
60 T=C-B
70 GOTO100
100 A=32768
110 FORI=1TO39-T:PRINT"■";:NEXT
120 POKEA,48:FORI=1TO9:POKEA+I,32:NEXT
130 POKEA+11,49:POKEA+12,48:FORI=1TO8:POKEA+I,32:NEXT:POKEA+21,50:POKEA+22,48
160 PRINT
170 FORI=1TO500:NEXT
180 S=S+1:IFS=23THENPRINT"□"
190 IFS=23THENS=0
200 GOTO5
```

FOOTNOTE. A "555" timer circuit is shown on p 93 of the PET Revealed together with a machine code count program. I am informed that line F0 07 BEQ TEST should be F0 F7 BEQ TEST.

First Programming Steps.

Jim Butterfield, Toronto

The first programs that a beginner writes tend to be simple. That's good, of course: the programmer is developing skills which will be useful when he tackles more ambitious jobs. Here are a few suggestions on how to go about these early projects; the emphasis will be more on sound practices and clear style rather than clever coding methods. Some of the suggestions might be useful for experienced programmers, too...

Try to lay out your programs in "blocks". Each block should have a clear, simple function. One block might do an input job, another might calculate, and a third generate output. If you start planning a program by thinking out the blocks you will need, your program will be better planned. Some programmers make each block into a subroutine so that the main program simply calls in these units as needed.

Title each section or block with a remarks REM statement. You don't have to put comments on each line, but it's useful to be able to find a section of code quickly. Perhaps you think that you can remember the code - after all, you wrote it - but wait a couple of months. It's amazing how a crystal clear program can suddenly become gibberish after you've been away from it for a while. Leave yourself some highway markers so that you can find your way around later.

Name your variables in a semi-meaningful way. Totals can start with the letter T, counts with a C, and so on. I'm not a fan of large alphabetic names, since they have pitfalls: TERRIFIC is a great label, but it doesn't work since the keyword IF is hidden in the middle. Can you find the hidden keywords in GRANDPA, CATNIP, CRUNCH and FRONT? It's fun to play word games, but not when you're trying to write a program. I prefer a single letter followed by a numeric: T4, B7 and so on. By the way, don't forget that variable B has nothing to do with integer variable B% or string B$ or for that matter array variable B(3). They are all completely independent values.

Don't ley anyone hustle you about program size or speed. If others write in less memory and fewer milliseconds, let them. You'll have space enough for most of your programs and the tenth of a second saved in run time won't give you time for a cup of coffee. On the other hand, do look for better methods. Better isn't always faster or smaller, but you'll recognize it when you see it.

Keep track of your variables; it's useful to make a list on a sheet of paper. That way, you won't accidentaly use variable X for two different jobs and get them mixed up. In fact, it doesn't hurt to do paperwork planning before turning your computer on. There's a kind of "heat" in working directly on the machine that sometimes leads to hasty programming. A little leisurely planning befoehand can generate sounder and better programs.

Don't be afraid to write loosely. The fanatic who tells you that you'll save memory and time by compacting FOR M = S TO P STEP V into FORM=STOPSTEPV is steering you wrong in most cases. If legibility costs you four bytes and one millisecond, take it: it's a bargain.

If your program doesn't work right the first time, don't lose heart. It happens to most of us. The easy errors are where the computer tells you where the problem is, most commonly ?SYNTAX ERROR IN ... The problem will likely be obvious when you look at the line, if not, you can try rewriting it slightly to see what happens. The hard errors are where the computer doesn't stop, but gives you the wrong answers.

Debugging can be great fun if you take the right attitude. Look at the variables: you can call them up with direct PRINT statements. Change them if it suits your purpose. Put STOP commands into your program and check everything out when you come to the halt. You can resume where you left off with CONT. Using the RUN/STOP key to break your program in mid-extension is less precise but will also do the job.

Getting a program together can be a rewarding experience - not necessarily rewarding in money, but in a sense of accomplishment. Each program will be a work of art, done in your own style. When you put your signature to your latest masterpiece, you'll feel good about it if you've used good coding craftmanship.

The Friendly PET - Screen editing *Jim Butterfield, Toronto*

One of the friendliest things about the PET, CBM and VIC is the way they allow you to make a change or correction. If the line on the screen is wrong - whether it is a program line or a direct command - we can move the cursor back and type over the line. Pressing the RETURN key will make the change take effect.

Correcting Programs

This is very handy for programs. When your first program attempts result in a message such as ?SYNTAX ERROR IN 350 you can list 350 to see what the trouble is. If line 350 happens to say PWINT X, you can move the cursor back, type R over the W to give PRINT X, and strike RETURN. The line has been corrected with a minimum of typing on your part.

If you need to make an insertion into your program, you may use the INSERT key. If the mistake was PINT X, the technique is to position the cursor over the I, hold down the SHIFT key, and press INST for insert; the computer will open up space and you can type in the missing R. On the other hand, if the error was PHRINT X you'll want to make a deletion: place the cursor over the R, press the DEL key to delete, and the H will disappear. In either case, don't forget to press RETURN to make the change permanent.

If you happen to goof in making the change, start over. In this case, don't press RETURN. Hold down SHIFT and then press RETURN: this will take you to the next line without any program change being made. Shifted-RETURN is quite a handy key combination to know for many reasons. If you wanted to leave a note on the computer's screen for someone to read, you might type MARY - PUT THE CAT OUT. At this point, striking RETURN would cause the computer to try to "perform" the line, and you'd get ?SYNTAX ERROR. If you press Shifted-RETURN, however, you'll just go to the next line and the computer won't try to do anything with the contents of the previous line.

The INSERT key has some special rules. After you have pressed the INSERT key a number of times (don't forget to hold down SHIFT) there will be an open space on the screen where you can insert the new characters. At this point, you'll be in "programmed cursor" mode. This means that the cursor keys don't move the cursor; instead they will print as special reversed characters. This is the same way that the PET behaves after you press the quote-mark, with two important exceptions: the computer remains in this mode only for the number of characters to be inserted; and the Delete (DEL) and Insert (INST) keys work in a different way. More about this another time; in the meantime, you'll get used to them quite quickly.

A problem sometimes crops up if a program line is too long. Sometimes this means that there's no extra space available to make a desired insertion - eighty characters is the screen limit. Worse, the line is too long to start with; it occupies over 80 characters even before we make a change. It might be more sensible to change it to two lines and relieve the crowding; but if you must, the trick is to look through the line to find a keyword that can be abbreviated. PRINT is the most popular, since it can be rewritten as a question mark. Close up the space, making sure that everything is packed into the 80-column work area, and then make the change if it fits.

Direct lines - Basic commands typed in without a line number so they are expected right away - are usually easy to fix. If you mistype LOAD "PROGRAM" so that it comes out LOUD "PROGRAM", don't be dismayed by the ?SYNTAX ERROR. You can slip the cursor back, change the U to an A, press RETURN and the load will take place.

Correcting mistakes in Direct lines can leave a cluttered screen. When I try to load BOTTLESHIPS from the disk, I got several lines which tell me that there's no such program. When I move the cursor back and correct to BATTLESHIPS, the following lines don't go away unless written over. It looks messy, but works OK.

There's a sharper problem when I use a direct statement to print a number. If I ask the PET to calculate 4*5*6, yielding a product of 120, and then decide that I really want addition, I can go back and change the asterisks to plus signs. The PET will now produce a total of 15, but the last digit of the previous answer won't be wiped out; the zero will be left on the screen and our sum will look like 150 instead of 15. The solution? Wipe out any numbers you want recalculated so that the new values will print on a fresh line.

Special screenings

When you press RETURN, the PET sees only what's on the screen. You may have done deletions, inserions, and changes but the final screen result is all that counts. This is true of program lines, direct command, and responses to program INPUT statemens.

You may want to run a program several times while testing, with similar answers to INPUT questions on each run. With screen editing, it's a snap. After you have run once, move the cursor back to the RUN statement. Press RETURN (no need to type RUN: it's on the screen). For each INPUT, the cursor will appear over the answer you typed on the previous run. If you want to go with the same response this time, just press RETURN and the program will accpet the same input from the screen. IF you want to change, type your new input.

Here's a hint of advance techniques that you'll learn as you become more familiar with your computer. You can actually get the PET to type its own input - even its own program changes - to the screen. Then, with a stroke of the RETURN key, you can activate the input or program change. When used for INPUT activities, this provides a "default" input for the user. As a program change, the program could suggest DATA statements that it would like to see included in a future run. Mind boggling! At this rate, the computer could program itself and make us all obsolete.

At least, the computer still needs us to press the RETURN, key; it can't do that by itself. Or can it? Technical tyros suggest that POKE 158,1:POKE 623,13 (or on Original ROMs, POKE 525,1:POKE 527,13) would actually cause the PET to send a carriage return to itself...

Half a dialogue - Inputting

Jim Butterfield, Toronto

Asking a program to go and get input from the user is a subtle thing for beginners. When you write your first program, it's hard to look ahead and see the program independently communicating with the user. "If the program needs a value, I'll program it in right now ..." It takes a level of sophistication to imagine a program accepting working values at a later time, when it runs, and using different values supplied by the user in different runs.

There are three fundamental ways of checking what the user is doing at the keyboard: INPUT, GET, and a PEEK. We'll talk about each, and its uses.

INPUT.

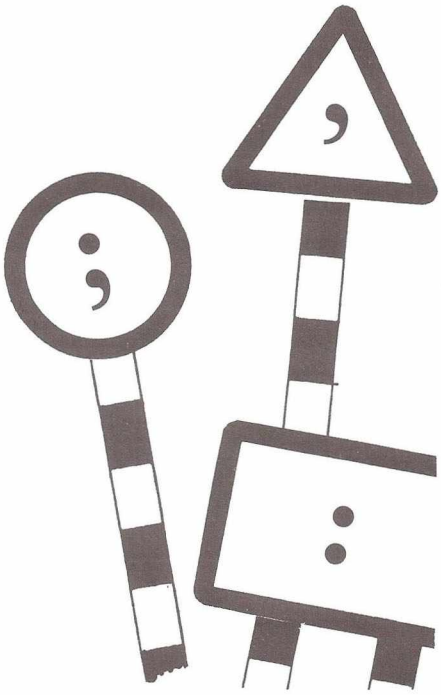
The INPUT statement does a lot of work for you. It's certainly one of the most powerful statements in Basic. Some of us would like to see it more powerful, and some would like to see it less sophisticated; for the moment, we'll have to accept it as it is.

When you give the command INPUT in a program, a prompting question mark is printed and the cursor begins to flash. Your program is held in suspended animiation; it will not resume operation until RETURN is pressed. There's no code which allows something like:
INPUT M:IF (NO REPLY IN 15 SECONDS) GOTO...

Your code will hang on the INPUT statement forever if the user doesn't reply.

When the user presses RETURN, INPUT takes the information from the screen. It doesn't matter if the user wandered back and forth, changing, deleting and inserting, INPUT looks only at the screen which is the result of his actions. In fact, if there's something on the screen that the user didn't type, INPUT will take that too. This can be useful for prompting: you can arrange to type a sample response on the screen, and the user will then be able to press RETURN to have that response entered. As input takes the information from the screen, it trims away all leading and trailing spaces; other than that it takes the whole line, even though it may not necessarily need it.

Now INPUT starts to plow through the line, digging out the information you need for your program. If it's looking for a number it will not like to find a string, and will ask, REDO FROM START. If it's looking for a string it won't mind a number at all: it will accept it as a string.



Road signs for INPUT.

Whether INPUT is looking for a number or a string, it will stop its search when it finds one of three things; comma, colon, or end of line. If if finds a comma it will assume that more information will be needed later in the INPUT statement; if it finds a colon or end of line it assumes that there is no more useful input from the user. If it needs more, it will ask for it.

Suppose you need to input a string that contains a comma or a colon, such as ULYSSES M PHIPPS, PHD or ATTENTION: JOHN, MARY. Since INPUT normally stops at the comma or colon character, we need to do something. The answer is easy: the user must put the desired input in quotes: "ATTENTION: JOHN, MARY" and the whole thing, commas, colon and all, will be received as a single string.

Keep in mind that the INPUT statement allows prompting. INPUT "YOUR NAME";N$ causes the computer to type YOUR NAME? and wait for input. That's a good human interface; help the user along.

If a user presses RETURN without supplying any information on the screen, programs on the PET/CBM will stop. There are several ways to prevent this from happening; the easiest is to add a "canned reply" to the input prompt message. When you are writing the INPUT statement prompt (such as YOUR NAME) add two extra spaces and, say, an asterisk character; then type three Cursor-Lefts (they will print as an odd-looking reversed bar) and close the quotes on the prompt. Finish the INPUT statement in the usual way: a semicolon behind the prompt and then the name of the variable to be input. Now: the askerisk or whatever will print to the right of the prompt and question mark. Unless the user overtypes it, this character will be received from the screen as his input - and the proram won't stop.

One last comment: don't forget that INPUT can accept several values. You can say INPUT N$,A$,C$ and allow the user to type JOE BLOW, CITY HALL, DENVER. It's often better to use separate input statements: users can respond better when prompted for each piece of information.

GET and PEEK: a preview

GET isn't as clever as INPUT, but it has valuable uses. First of all, it doesn't wait; if a key isn't ready in the keyboard buffer, the GET statement lets Basic continue. Secondly, keystrokes received with GET don't affect the screen unless you, the programmer, decide to allow them to do so. This means that you have much more control over what the user can do.

There's a PEEK location PEEK (151) on most PET/CBMs, PEEK (515) on Original ROMs, and PEEK (197) on the VIC that tells you whether a key is being held down or not. This can be useful to avoid the situation where a user needs to press the same key repeatedly to cause some action; you can program so that the key repeats its action if it is held down.

We'll talk in more detail about the GET and PEEK in the next article — Reading Keys.

Half a dialogue - Reading keys

Jim Butterfield, Toronto

We've already discussed the INPUT statement. When you do an INPUT, the program pauses and waits for the user to compose a line on the screen. When the user presses RETURN, the program resumes and uses the information entered.

This is often useful and convenient; but when we use INPUT, we don't have complete control over the user. If the user doesn't answer, the program is stopped forever, and other jobs will not take place. The user might also do undesirable things like clearing the screen, and might even stop the program if he presses RETURN without any input on the screen.

We can deal with the user on a more elemental level by using the GET command.

GET.

GET takes one character directly from the keyboard buffer; the character does not go via the screen. It's usually a good idea to echo the character to the screen so that the user can see what he's typing (GET X$:PRINT X$;). There is a GET numeric (GET X) which gets a single numeric digit, but it's rare since the program will stop if the user inadvertantly presses an alphabetic key.

GET doesn't wait. If there's no character in the input buffer, GET returns with a null string. We can wait for a key to be pressed with a line like:

300 GET X$:IF X$="" GOTO 300

You can see that if we get no character, we go back and try again. More sophisticated versions of the same program might allow us to wait for up to 10 seconds for the user to type a key.

GET receives everything typed at the keyboard. Even cursor movement or insert and delete keys are received as single character strings. The RUN/STOP key and the SHIFT are about the only keys that GET won't receive directly.

Screen control keys - cursor move, reverse, home, etc. - are picked up directly by GET and don't influence the screen when typed. If you want them actioned, you'll have to arrange for it yourself, again by echoing the character with a PRINT. On the other hand, GET is an excellent way to prevent a user from clearing the screen or doing other things that you don't want. The easiest way to identify such characters is by their ASC ascii value, but the obvious also works: GET X$:IF X$=HOME" GOTO... The Reverse-S symbol will appear where I have typed HOME.

Sometimes there are left-over characters in the keyboard buffer. The user might have touched the keyboard accidentally, or the last key pressed might have "bounced" and been registered twice. You can strip out such characters with simple coding like GET X$,X$,X$,X$. If the keyboard buffer contains up to four characters, they will be cleared out; if there were none, GET still doesn't hold anything up.

Remember that GET takes characters from the keyboard buffer. For one key depression, no matter if you tap a key quickly or hold it down for five minutes, only one character will go into the buffer and GET will find it there only once.

PEEK

The value of PEEK (151) will tell you whether or not a key is being held down. If you find 255 there, no key is being pressed - except maybe the SHIFT key which doesn't register there. If there is any value other than 255 in PEEK (151), somebody's holding down a key.

Special note: for Original ROM PETs, the place to check is PEEK (515). And on the VIC, check location PEEK (197); a value of 64 means that no key is being pressed.

It's possible to figure out which key is pressed on the value you find in the PEEK location, but I don't recommend it. Different keyboards are "decoded" in different ways, and what works on one machine won't necessarily work on another. The best way to sort out which key is pressed is to use the PEEK together with the GET statement.

The trick is this: if GET says that there is no character in the keyboard buffer and PEEK says that someone is holding a key down, it's safe to assume that the key being held down is the last one you received with GET. Timing is important here, since a key could be touched in the split second between two Basic statements. I recommend the following kind of sequence:

```
300 X=PEEK (151)
310 GET X$:IF X$ "" 
    THEN X1=ASC(X$)
    :GOTO 330
320 IF X=255 GOTO (...NO
    KEY ACTIVE)
330 ....KEY ACTIVITY
```

This kind of test is very good for movement games, where you are directing something (a ball, a paddle, a tank) around the screen based on whether a key is held down or not.

Summary

GET is more elementary than INPUT. You'll need to do more work with GET, but you'll have more control over the user input.

Use the PEEK where it's necessary to find out if a key is being held down or not ... it can give you a nice interface, especially where the user would otherwise pound repeatedly on a key.

Commodore from the outside - posh stuff eh ?!

The Group implementation of 'large' programs

John Stout

This article goes in to more detail about a proposal mentioned in the article **Basic Compilers,** that of writing 'large' programs, e.g. a BASIC compiler for the PET, by using a number of people, each with access to the required hardware, to write sections of the program so that any one person's job is well within their capabilities and resources.

Most software firms do indeed split up large jobs between individual programmers since it is difficult for one person to retain detailed knowledge about every section of large programs such as operating systems. It is of course far easier to accomplish this when all the programmers are in the same organisation (sometimes under the same roof) and one of the first requirements for a group implementation will be excellent communications between the members. A simple solution using the postal services and floppy disks with the telephone for emergencies seems most practical. The sort of thing I envisage is a pair of floppy disks making their way from member to member, each member taking off the disks the programs and documentation written by other members and adding their own. For a large group this 'chain' would result in obvious delays and so an alternative 'star' could be adopted where one member of the group would act as a 'librarian' to whom each member of the group would send his/her own work and receive back copies of everyone else's. Using member's own disks and sending SAEs with each disk would mean that each member would pay their own way and thus avoid any problems with finance.

A second requirement for the project to have any hope of succeeding would be the agreement of everyone involved to a set of standards, for programming style, documentation and work. I have in mind such things as spaces after every BASIC keyword, no GOTOs to REMd lines, REMarks at the start of every subroutine detailing purpose, variables used and so on. Documentation should be distributed in machine and human readable form so that each member could be supplied with a simple word processing program in order to read other's documentation and to create their own. The program would be designed so that every person's documentation would have the same structure although obviously different contents. Similarly we would need standards of work, e.g. minimum number of hours per week otherwise a number of energetic programmers could end up supporting the rest of the group. The aim of all these standards would be to have a situation where any member of the group could take over the programming of any other member and continue that work with as little wasted time as possible.

Members should be able to take criticism since everyone's work would be open to examination and evaluation by the rest of the group. Procedures for dealing with criticism would need to be worked out and an arbitrator appointed for cases where the two sides to an argument cannot come to some agreement or compromise. Alternatively the group members not involved could arbitrate which would lead to a fairer decision but one which would take longer to reach.

Another requirement, technical this time, would be a program to merge together member's work from disk, either into memory or into another disk file and do this reliably time after time, coping with duplicate line numbers, out of memory errors and so on. It would be crippling to have completed one's section of programming only to have it corrupted by trying to merge in some other section.

Before the program gets under way the group members would need to come to some arrangement about copyright and the possibility of selling the program when completed. Sharing any profits equally, in the ratio of number of lines produced or in the ration of time taken are just a few of the financial ideas to be considered.

Among the projects amenable to a group implementation would be such things as compilers (each member being responsible for the subroutine to generate code for a particular statement type), word processors (each member being responsible for the programming for a particular command) and so on. Special interest groups could collaborate on such things as accounting software, symbolic mathematic systems (possibly using LISP) and financial modelling programs. As long as a clear division can be made between separate parts of the program and agreement reached on the points mentioned above, the philosophy of 'divide and conquer' can be applied to programming as well as to warfare.

Once upon a time, to a select group of attendees, an event of historic importance took place. Graham "Houdi" Sutherland, of Commodore Business Machines, John Kyle-Price, of the Bristol Software Factory, and Richard Pawson, of Printout, made a very special announcement. To an astonished world, Silicon Office was launched for the first time!!

Forth coming

Peter Bengston

FORTH is something which I will readily admit comes under my list of things I know next to nothing about. So, the following article landed as a very nice surprise on the desk the other day, which was as ever snowed under with all kinds of goodies. If, after reading the article, you want further information, write to:—

Peter A. Bengtson
Software Development Manager
Datatronic AB Sweden
P.O. Box 42094
S-126 12 Stockholm
Sweden

FORTH - The Language of the Eighties?

How about a language that is interactive, structured, modular, extensible, very fast, compact, portable, supports virtual memory, works in any numeric base, allows you to freely mix assembler and high-level code, cuts development time in half, and occupies just 8.5 kilobytes - **including** a resident macro assembler and a resident text editor?

A dream? No - the language is FORTH, a not-so-new language that has been around for over ten years by now. It is rapidly gaining acceptance in wide circles thanks to the efforts of a FORTH Interest Group started in 1978, now having 3000 members worldwide and growing all the time.

FORTH was created by Charles H. Moore in 1969 at the National Radio Astronomy Observatory, Virginia. He was fed up with the tediousness of programming a computer at that time. Job Control Languages, Linker Languages, Macro Languages, FORTRAN, COBOL, PL/1, etc, etc. He wanted to replace all these languages by just one - FORTH. FORTH developed over a period of ten years, slowly and carefully. Thus it benefits from the consistency gained from being the product of one mind.

However, FORTH is not a "frozen" language, as most conventional languages are; it is extensible. When you are programming in FORTH, you are actually extending the language, making it have new properties and capabilities. You can add your own data structures, or even extend it with new types of program structure. This feature is unique to FORTH, and is not shared by any other programming language.

FORTH is modular and completely structured. Programming consists of combining pre-defined or user-defined modules (called "words") into increasingly powerful units. Finally you will have one word that is your whole program. Debugging is made simple; all modules may be tested individually before combining then into higher levels.

The implications of this are far-reaching. If you mainly are interested in business applications, you will soon have extended FORTH with words turning it into a business-oriented language. If your interests are in process control, the language will turn into a dedicated process control language. In fact, you can have both at the same time.

This means that you will create a library of words, suited to your particular area of programming. They may then be used over and over again, since they are independent of their context. You may find, when writing an application, that it is already written to 80% by your previous applications. This will bring down the developement time dramatically; a tenfold increase in productivity is claimed by some.

FORTH is also very fast and compact. It is implemented as **threaded code,** a very powerful language implementation technique used in, for example, many COBOL compilers. FORTH employs a variant called **indirect** threaded code, to which it ows much of its flexibility and compactness. For example, a typical FORTH macroassembler, written in FORTH, usually occupies just over 1500 bytes. FORTH programs are generally shorter than machine code, due to the threaded code principle. As for speed, 1000 empty loops take under one second. Commodore BASIC - a fast one - does the same thing in thirteen seconds. FORTH is usually 30 to 70 percent slower than pure machine code, compared to BASIC that is over 1000 percent slower.

If your application requires it, in a time-critical part for example, you may re-code parts (or all) of your program in assembler code, using the FORTH resident macro assembler. It is structured, meaning it has IF-ELSE-THEN (sic), BEGIN-UNTIL, BEGIN-WHILE-REPEAT, and BEGIN-REPEAT instructions.
With a FORTH assembler, it is not uncommon to have the code working the first time. Modules created by the assembler are treated in exactly the same way as all other FORTH words, standard or user-defined; not even the compiler is able to detect any difference.

FORTH can also work in any desired numeric base, without affecting computation speed. You may work in vinary, trinary, octal, decimal, unidecimal, hexadecimal, dodekadecimal, or in any other base of your choice. When printing or displaying values, you can format the value as you wish. In accounting, for example, you may wish to put parentheses around a negative value: you simply extend FORTH with a new printing operator to do this; it is defined in half a line.

All arithmetic in FORTH operates on a stack. If you own an HP calculator you will be familiar with this type of arithmetic. FORTH uses **postfix notation** of expressions, meaning that the operators **follow** their operands. For example, the expression "(1 + 2) * 3" is written in postfix notation as " 1 2 + 3 *". This has many advantages apart from making all parentheses unnecessary; in fact, it is the way you really think when evaluating an expression. After the "initial shock" you will rapidly become proficient in postfix notation - you will realize, perhaps reluctantly, that it really is the only way to go. Traditional (infix) notation is not something "natural"; it is just something we happened to learn in school.

Postfix has other advantages too. FORTH procedures do not need any parameter lists; they simply take their parameters from the stack. Any results is put on the stack, where later other procedures may pick them up. Natrually, this scheme of parameter passing elegantly allows recursion, a very useful programming technique.

Normally, FORTH operates on integers. This is partly because history in automatization; integer arithmetic is very fast. FORTH has a multitude of single- and double-precision operators that partly eliminate the need for floating point arithmetic. In fact, complete Fourier transforms have been written in FORTH **without** floating point words. Floating point is however becoming available. Many FORTH vendors supply floating point packages as extensions, to be loaded when needed. If the computer has floating point software in ROM, FORTH can easily link to it. This is the method adopted in PET-FORTH, a full, extended FORTH supplied by my company, Datatronic AB. The floating point software, complete with trigonometric functions, does not occupy memory space when not desired, since you (or your programs) load it only when it is needed in your particular application.

As for portability, FORTH systems are very much alike, and programs are usually portable from one computer to another with very little change. The FORTH Interest Group has published compatible source listings for almost every microprocessor available (1802, 8080, PACE 6502, 8086/8088, ALPHA MICRO, 6800, 9900, PDP-11, 6809, NOVA, and more to come) at a very low cost ($20). However, these versions need customization before they can run on a computer. FORTH Inc, a company owned and operated by Charles Moore, have even more versions available, and they can provide training on-site, when installating. They have put FORTH on - watch out - the IBM 1130, Burroughs 5500, Univac 1108, Honeywell 316, IBM 360, NOVA, HP 2100, PDP-8, PDP-10, and PDP-11, Varian 620, Mod-Comp II, GA/SPC-16, CDC 6400, Computer Automation LSI-4, RCA 1802, Interdata, Motorola 6800, 6809, and 68000, Intel 8080 and 8086, Mos Technology 6502, Four Phase, ILLIAC, and the TI 9900, just to mention a few. And they will not stop there, I think. Soon, the production of a special FORTH processor will start; it will have FORTH as its machine code language.

FORTH does not need much memory, unlike many popular high-level languages like Pascal and FORTRAN. The famous ADA languages will not comfortably fit into even 64k! FORTH programs usually fit into 16k of memory, and 32k is almost luxury. If your program is huge, you can use segmentation and/or virtual memory for your data areas and program modules. Memory is normally not a problem in FORTH.

You're probably wondering "why this sudden activity around FORTH **now,** instead of ten years ago?" A few years ago, not even computer scientists knew of it. The few people who were using FORTH were (and are) very enthusiastic about the language and its capabilities. To make the FORTH language wider known, a handful of FORTH programmers formed the FORTH Interest Group (FIG) in 1978. Through their version of FORTH (fig-FORTH; now almost an industry standard), they have managed to do just that in a very short time. Another important event was that BYTE magazine devoted their entire August 1980 issue to FORTH, thereby creating a floodwave of enquiries to FIG. FIG estimates to exceed 5000 members in a year.

Who is using FORTH, and for what? The first applications were in astronomy observatory automation, in fact, that was what FORTH Inc was founded to do. Quickly they realized, after having automated almost every observatory in the world, that you couldn't support a company on that rather exclusive market. So they went into the administrative world. They have developed many powerful database systems, and are now entering a major expansion phase.

Atari is using FORTH to replace machine code in arcade games. It is used in science fiction movies to do animation and control space ship models, for example in the movie "Battle Beyond the Stars." A FORTH-like language called SNAP is used inside the Hand Held Computer, developed by Friends Amis, U.S.A. FORTH controls satellites orbiting the Earth, prints invoices, calculates salaries, controls communication nets, sorts peaches, monitors laboratories, analyzes pictures, synthesizes music, and many other things. LISP and PASCAL interpreters have been written in FORTH. Datatronic AB has used it to control residual gas analyzers, involving a lot of fast graphics and computations. We have also written a scheduling system, using n-dimensional virtual matrices. I've written an Adventure game in FORTH in my spare time. In fact (and this is not an overstatement), you can program **any** application in FORTH.

By using a Target Compiler, you can get a compacted version of your program, suitable for burning into a PROM, with a typical overhead of 500-800 bytes. Then, you can use this PROM without any other software to do whatever you wish. Target Compilers are usually hard to get, since they are intended for industrial use; but with some effort you can usually find one.

How can you get FORTH for your computer? If you are good at machine code programming, and will enjoy typing 80 pages of assembler listings into your computer, you can get a listing from FIG (Forth Interest Group, P.O. Box 1105, San Carlos, CA 94070) for your particular microprocessor. It will take some time to customize, but the cost is after all only twenty dollars.

You can also purchase a customized version from a vendor. If you own a PET 8032, you might want to contact us. PET-FORTH includes double-precision arithmetic, random numbers, IEEE handling, trig functions and a powerful string package with string searching capabilities. A textbook and reference manual of 322 pages is included in the price ($390). Floating point routines are available as an extension. The program has a life-time guarantee. Versions will be available for the CBM 8096 and the MMF 9000. We do also sell a Target Compiler.

Another of FIG's goals is to raise user expectancy of the behaviour of higher-level languages, using FORTH as an example. FORTH is much more than a language; it can also be regarded as an operating system. Some have even questioned whether or not it **is** a language. Someone proposed the name "meta-language". Charles Moore: "...Is FORTH an operating system? Is it a language? No programmer can afford to bypass it.

Peter Bengsten

Bulletin Board

Fred Brown

One of the fastest uses for microcomputers in the United States, is for exchanging information between computers over the phone.

Micro owners now have access to many networking systems and databases, like the SOURCE and MICRONET where you will find a wealth of information ranging from stock prices to the latest news from this New York times.

But the most popular is local networking systems called bulletin boards also known as electronic mail systems, permitting users to enter and retrieve messages or information.

In the past 18 months bulletin boards have been set up all over the states, nearly every town or city have at least one, some large cities have several.

Some bulletin boards specialise in subjects like education, medical, there is even one for other bulletin board numbers, and in San Francisco there is one for gays.

Bulletin boards have now been operating in the U.K. for over a year and all operate electronic message system, programme library and information retrieveal, plus other items like Forum-80 (HULL) as TRS-80 and PET user's sections.

To access a bulletin board system you need an RS232(if your computer hasn't got one) and a modem plus software and any microcomputer can be used.

Once you have set up your RS232 (see list2) and loaded your programme, you are ready to call a bulletin board.

Pick up the telephone and call one of the system numbers (see list no 1)'

When the other end answers, you should hear a tone on the line, now set your modem to originate (or place your handset in the cups if you are using an acustic modem) and if all goes well a message will appear on your CRT, if not press carriage return a few times until the distant system responds.

Example of signing on to a FORUM-80 Bulletin Board

The FORUM-80 is designed to accept a wide range of 300 baud configurations including 7 or 8 bit words. However, the standard configuration of 7 bits, no parity and 1 stop bit is best. When you sign on, a greeting will be displayed

WELCOME TO THE FORUM-80 OF (HULL, ENGLAND).

the system will ask you the following questions to which you give appropriate answer:

WHAT IS YOUR FIRST NAME?
BOB
WHAT IS YOUR LAST NAME?
SMITH
WHERE ARE YOU CALLING FROM?
SWINEFLEET, YORKS.
NAME: BOB SMITH
FROM: SWINEFLEET, YORKS.
IS THIS CORRECT, BOB?

Checking users files.

At any point, our FORUM-80 system is checking to see if you have signed on to this system recently. If you had signed on recently, a record of your terminal configuration would exist. This is one of the handiest features of FORUM-80. Suppose you are using an Apple II to talk to FORUM-80. Your control characters for various functions (clear screen, input prompt, etc.) are unique to the Apple. The same problem exists with many other microcomputers and dumb terminals. FORUM-80 solves this problem by allowing you to set up your own configuration table that the FORUM-80 uses every time you sign on. The user file is not permanent. You must call the system fairly regularly to keep your user file in system. Creating a user file for your terminal/micro is done by entering command "C". If you have never called a particular FORUM-80 system before, a "UNIVERSAL" configuration will be used. Then the system will give you the following information:

YOU ARE THE CALLER %%%%
LAST MESSAGE IN SYSTEM
ON YOUR LAST CALL: %%%%
VERSION 3.1 08/08/81

At this point, the system will tell you if there are any messages in the system addressed to you (assuming correct spelling of your name was observed by the sender). To retrieve the message(s), simply enter an "F" for Flagged Retrieval once you enter the command mode. The system automatically flags these messages for you.

** BULLETINS **
(HIT 'S' TO SKIP BULLETINS)
(HIT 'P' TO PAUSE)

The bulletins that the system operator has entered will appear next. If you want to skip them, press "S" key. You can use the "S" key in many other parts of the FORUM-80 system to stop output. If you are reading the bulletins and you want to pause the output for a moment, press the "P" key. Once the bulletin has been displayed or bypassed, you enter the command mode. The following will appear every time you enter the command mode:

00:45 COMMAND:

The numbers just before the word "COMMAND" indicate how long you have been on the system (in this example, 45 seconds). If you press Carriage Return, "Enter" key or otherwise send an invalid input, the command table will automatically be displayed to you. If you are using a video-type terminal and have entered any invalid input, the command table will automatically be displayed to you. If you are using a video-type and have properly configured the system to your terminal (using the command "C'), the screen will clear after a command is entered.

Main commands

00:47 COMMAND: (send a c/r will print a list of Subcmds)

S = SUMMARIZE MESSAGES
E = ENTER MESSAGES
F = FLAGGED MSG RETRIEVAL
O = OTHER SYSTEM NUMBERS

H = HELP WITH SYS OPERATION
C = CONFIGURATION CHANGES
T = TERMINATE CONNECTION
R = RETRIEVE MESSAGES
K = KILL MESSAGES
M = MESSAGES IN SYSTEM
I = INFORMATION ABOUT SYSTEM
U = USER LOG
L = LOCAL FEATURES SECTION
02:04 COMMAND:

The above are the valid commands of FORUM-80 (3.1). Some FORUM-80 systems may also have an "L" command in this list. Entering a "L" command gives you access to special LOCAL features that may be supported by a particular FOURM-80. Lets examine each command.

Summarize messages

Typing "S" and pressing "ENTER" or "CARRIAGE RETURN" lets you enter the message summary mode. In this mode, you can look at a summary of the messages currently in the system. You can examine all the message headers, which contain information like who the message is to and from, the subject of the message and the category of the message (-M-Miscellaneous, -C- Commercial, -G-Graphics or Experimental, -P-Personal passworded). Here is the list of subcommands in the SUMMERIZE MESSAGE mode:

00:30 COMMAND:
LOADING FILE - (S) SUMMARY
03:40 (S) SUBCMD:
C = COMPLETE (FULL) SUMMARY
Q = QUICK (ABBREVIATED) SUMMARY
S = SEARCH SUMMARY FILE
F = FLAGGED RETRIEVAL
A = ABORT AND RETURN TO COMMAND MODE
03:48 (S) SUBCMD:

Enter a message

Message entry is full of handy features that, to the unaware user, can cause some difficulty! These features are explained in Volume 2 of the Users Guide. Here we can only say "experiment at your own risk!". Below is an example of the standard method of entering a message into the system 09:10 COMMAND: 'E'
LOADING FILE - (E) ENTER MESSAGES
09:19 (E) SUBCMD:
M = MISCELLANEOUS (GENERAL INTEREST)
P = PERSONAL (PASSWORD PROTECTED)
C = COMMERCIAL (FOR SALE, WANTED TO BUY, ETC.)
G = GRAPHICS OR EXPERIMENTAL
A = ABORT AND RETURN TO THE COMMAND MODE
09:27 (E) SUBCMD:

These are the categories discussed in the summary search function. Passworded messages are accessible only to those who know the message password. Graphic or Experimental, Miscellaneous and Commercial messages are accessible by the user. Graphics and Experimental are described in further detail in Volume2 of the Users Guide. Choose the Commercial category only if you have something to buy-sell-trade etc.

In summation, remember these two things about message entry:

1. Leave BLOCK Entry alone unless you know what you are doing!
2. SAVE before ABORT, or the system will "forget" the message you just entered!

Other system numbers

Entering the letter "O" puts you into the directory of other system phone numbers. In this mode, you can selectively look at a list of system numbers for a particular city, area code, state, or system type (ABBS, CBBS, FORUM-80, etc) or all system numbers in the listing. The listing is as accurate as humanly possible and is updated regularly by the SYSTEM OPERATOR, however-we do not promise or guarantee that you will always get a computer at the other end! Here is an example of how the "O" command works:

13:05 COMMAND 'O'
LOADING FILE - (O) OTHER SYSTEMS PHONE NUMBERS
13:19 (O) SUBCMD:
F = FORUM-80 SYSTEMS
O = OTHER SYSTEMS
S = SEARCH FILE
A = ABORT AND RETURN TO COMMAND MODE

Help with system operation

Most of the contents of this part of the system are explained elsewhere in this users guide. However, there is one section that is not and describes one of the most POWERFUL features of the FORUM-80 system. It is the MULTIPLE COMMAND ENTRY feature. If you are calling long distance and you want to conserve your system access time (resulting in a SMALLER phone bill) you should memorize the following explanation from the HELP file on MULTIPLE COMMAND ENTRY:

The system is equipped to receive multiple command in a single input.

Multiple commands are accepted only in the COMMAND MODE input, subcmd prompts do not accept multiple command. The multiple command string may consist of from 1 TO 20 sequenced commands as long as all commands operate within the same function. The multiple command sequence is terminated when you re-enter the COMMAND MODE (or execute the last command of the sequence).

The choice of the delimiter between commands is up to the user, any non-alphanumeric ASCII character may be used but the same delimiter must be used throughout the multiple command string. For example, if you are inputing a mult/cmd string which will not include a search string, then a space may be used as delimiter. However, if your mult/cmd string will include a search string with an embedded space, such as 'JIM BROWN' then another delimiter such as, or ; or : should be used.

Configuration changes

If you wish to take advantage of the full potential of the FORUM-80 systems, you will want to set up a configuration table to match the FORUM's control characters to your terminal. For example: the Apple II computer, when using the D.C. Hayes MICROMODEM, clears the screen every time the FORUM Sends a OE(HEX) character. On the TRS-80, this character turns on the cursor., In earlier versions of the FORUM-80 software, every time the system looks for input from the user, it would send a QE character. An Apple User would just barely get a chance to read the question asked by the system because the screen would go blank when the QE character was

sent!! To eliminate this problem and other similar ones due to lack of control character standards for terminals, the CONFIGURATION CHANGES function was designed. It allows you to set up a configuration table that the FORUM-80 software uses to process the control characters you wish to be sent by the system.

Retrieve messages

Entering an "R" in the command mode lets you enter the Message Retrieve mode. These are the SUBCMDs available:

21:20 (R) SUBCMD:
F = FORWARD SEQUENTIAL RETRIEVAL
R = REVERSE SEQUENTIAL RETRIEVAL
I = INDIVIDUAL MESSAGE RETRIEVAL
P = PERSONAL MESSAGE RETRIEVAL
S = SEARCH MESSAGE FILE
G = GRAPHICS / EXPERIMENTAL RETRIEVAL
A = ABORT AND RETURN TO COMMAND MODE

Forward sequential retrieval example:
FORWARD SEQUENTIAL
(MSG 2677 to 3039)
START AT MSG:
The messages will be displayed starting at 2800 continuing to 2677 or until you press the "S" key to stop.
Reverse sequential retrieval example:
REVERSE SEQUENTIAL RETRIEVAL
(MSG 26 to 3039)
START AT MSG:
The messages will be displayed starting at 2800 and continuting to 2677 or until you press the "S" key to stop.
Individual retrieval example:
INDIVIDUAL RETRIEVAL
(MSGS 2677 to 3039 C/R TO END)
MSG NO:
Message 2765 will be displayed, assuming that a message is assigned to that number. As messages are purged from the system by users, message numbers are not reassigned to new messages or existing messages in order to avoid confusion in identifying a specific message with a specific message number. The system will ask you for another message number. If you are done, enter a cariage return to exit to the SUBCMD function. Otherwise, enter another message number to view another message.

Kill messages

The kill function let's you kill messages from the system as necessary. You MUST know the message password in order to kill the message!

22:34 COMMAND:
LOADING FILE - (K) KILL MESSAGE
22:43 (K) MSG NO:
MSG: 2767
DATE: 04/10 -22.34
FROM: JIM CAMBERON
TO: KILL TESTERS
SUDJ: TEST KILL
IS THIS THE CORRECT MSG (Y/N) ?
ENTER PASSWORD:
MESSAGE KILLED

Information about system

Entering a "I" will display information concerning the hours of operation of the system, the equipment configuration used, and the address of the FORUM-80 Headquarters. System hours of operation are controlled by the local system operator and will vary from system to system.

User log

Entering a "U" will display a list of users who have called the system in reverse chronological order. The user log shows only the most recent call of a particular user.

Terminate connection

To terminate your connection with the FORUM-80, enter a "T".

26:30 COMMAND:
If the system is equipped with a comment file, the following will appear:
WOULD YOU LIKE TO LEAVE CONFIDENTIAL
COMMENTS TO THE SYSTEM OPERATOR?
USE LINES SHORTER THAN 80 CHARACTERS
HIT CARRIAGE RETURN ALONE WHEN FINISHED.
1
2
When you are finished with your comment and have hit a single carriage return, the following will be displayed:
THANKS FOR CALLING FORUM-80, BOB
LAST MESSAGE:2800
TIME ON SYSTEM: 28:14
DATE: 08/08/81
CONNECTION TERMINATED

That's it! Study this guide carefully and you should get great satisfaction out of using bulletin systems. As mentioned throughout this guide, Volume 2 deals with the more sophisticated features of the FORUM software and will include notes on updates to the software that will affect user operation.

A system is only as good as it's users, so feel free to leave messages on your local system on the headquarters system if you have any comments or questions about the bulletin board system. REMEMBER, the system is designed to operate with a wide variety of terminal or microcomputer configurations. It is not written especially for any particular system. If you have problems communicating with the FORUM-80, we want to know all the details of the problem. Leave comments explaining your difficulty on any FORUM-80 system or drop a line to FORUM-80 HEADQUARTERS, this will enable us to make the FORUM-80 a "UNIVERSAL" hobbiest Electronic Message System.

LISTING no 1.

1)FORUM-80 HULL. (Forum-80 H,Q) tel: 0482 859169
The worlds first international bulletin board.
System operator FREDERICK BROWN.
International electronic mail. Library for up/down loading software.
Forum-80 users group, Pet users section, Shopping list.
System hours, 7 days a week midnight to 8.00 am
Tues/Thurs 7.00 pm to 10.00 pm
Sat/Sun 1.00 pm to 10.00 pm

2)FORUM-80 LONDON tel: 01-286 6207
System operator LEON JAY
Electronic mail, Library for downloading.
System hours Tues/Fri/Sat/Sun 7.00 pm to 11.00 pm

3)FORUM-80 MILTON (TRS 80 users group 80-NET) Tel: 0908 566660
System operators LEON HELLER & BRIAN PAIN.
Electronic mail, Library, Newletter, TRS-80 information
System hours 7 days a week 7.00 pm to 10.00 pm

4)FORUM-80 HOLLAND Tel: 010-313512 633
System operator NICO

Communications Editorial

Welcome to this, the third in our series of communications specials. When starting this central pullout feature, my only 'aim' was to have, every other month, a feature on education, and let the other months be taken up with whatever came along. However, owing to the excellence of the articles contributed in August and October by Dr. Philip Barker (principal lecturer in the department of Computer Science at Teeside Polytechnic) I had no hesitation in devoting those month's features to the role of the PET in differing aspects of communications : firstly 'Using a Microcomputer as an Interactive Terminal', and secondly 'Algorithms for Intelligent Terminal Operation'.

After publishing those two it seemed only fitting to round off the year with a third article by Dr. Barker, and so here we have 'Computer Networks and Program Distribution'. This month the article concerns itself with an introduction to computer networks, and explains clearly and precisely just what computer networking means. We then move onto the implications of distributed computing for users of personal computers, and an outline of some of the problems associated with software exchange. Finally, he describes a technique for transmitting programs over the public switched telephone network.

As in the two previous specials, we are concentrating in particular on the role of PETs in the world of communications. Although the majority of Dr. Barker's article is aimed at microcomputers in general, the algorithms and routines that he describes are aimed at the PET itself : a 3032, to be precise. So the article becomes useful not only to 3032 owners, but also to owners of other PETs as well.

As you may be aware, 1982 is Information Technology year, so you can guarantee that there'll be more articles coming up. If you've developed anything in this field, write and let me know. All contributions should be sent to the usual address at the front of the magazine.

Well, I've said enough, and its time for Dr. Barker to take over. Let's start communicating!

Computer Networks and Program Distribution

Philip G Barker, Department of Computer Science, Teeside Polytechnic, County Cleveland.

Introduction

One of the most interesting and useful areas of mathematics is that which deals with graph theory and its applications to science and engineering. In computer technology graphs are used for a variety of different purposes. Some of the more well known uses include: the formulation of flow charts and data structure diagrams in programming; the representation of electronic circuit diagrams; and, as a notational tool for use in the design of data bases and distributed computing systems. Of the graph structures that are commonly encountered one particular type, the network, is of significant importance. It is used widely in software engineering and in many areas of computer design. Essentially, a network consists of a collection of 'point locations' called nodes (or vertices) that are inter-connected by a series of directed arcs or edges.

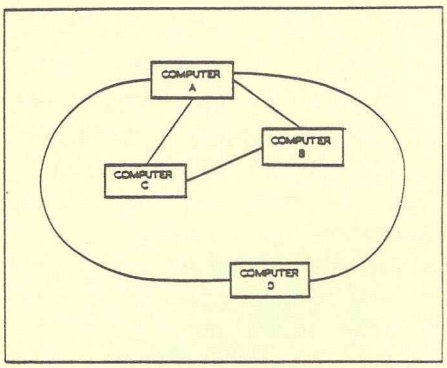
When designing or building computer systems graphs are often employed in order to model the entities under consideration. Nodes of the graph structure are used to denote the various components from which the system is (to be) constructed -these could be resistors, chips, tape decks or complete computer installations. The directed arcs of the graph then represent the inter-connections between the building blocks of the system. In reality these might be copper tracks on a printed circuit board, strands of wire on a patch board, optical fibres, telephone lines, microwave beams or lasers.

Computer systems technology embraces a wide variety of components: micros, minis, mainframes and supercomputers (IEE80). The latter are based upon the highly parallel interconnection of many processing elements to produce multiple CPU configuration and array processors. The latest trend in this domain is towards the use of ultracomputers (Sch 80). These require a means of putting together computing assemblages consisting of thousands of inter-linked elements. The practical realisation of such an arrangement depends upon the use of VLSI technology. The motivation for connecting computing elements together lies in the fact that with such machines it is possible to achieve faster computational speeds (through parallelism), greater reliability (through redundancy) and more flexbility as a result of dynamic sub-task allocation. Computers linked together in this way are usually located within fairly close proximity - often within the same room. The term multiprocessing is often used in order to describe this type of inter-connection.

There are many other reasons for wanting to link up computer systems - particularly, in situations where the distances between the elements to be linked is geographically large -perhaps, thousands of miles. The term computer networking is used to describe this approach to digital systems inter-connection. Figure 1 shows the technology of a typical network configuration.

figure 1.

TOPOLOGY OF A SIMPLE COMPUTER NETWORK



Computer Networks - an Overview

In this diagram individual computers in the network are represented by the nodes of the graph. The connecting lines (arcs) now represent data transmission links that enable data and information to flow between the various nodes. The term communication sub-network is often used to refer to the underlying data transmission arrangements that support the integrated computer system. The geographical distribution of the nodes in the network will usually not influence its operation or the functions it is designed to perform. Thus, in figure 1, computer A might be located in London, B in Paris, C in New York and D in Oslo. Sometimes there will be more than one direct link between given nodes (as in the case of A and D) in order to provide greater overall system reliability.

Closely dependent on the idea of computer networks and a concept of growing importance is that of distributed computing. Here the computational tasks to be performed in solving a problem are serviced by the resources of the computer network as a whole rather than those associated with any particular individual node. Suppose, for example, that the simple network described above was to be used for some scientific experiment. It is feasible that laboratory data collected by a data acquisition system attached to computer D could be transmitted to computer A for processing; then, following this, the results could be sent to computers B and C for storage. Retrieval requests for inspection of particular items of data might then arise from users of any of the four computers, A, B, C or D. Distributed processing of this type can offer many advantages such as: high availability and greater reliability; improved work throughput and response time; distributed data processing, storage and retrieval; load levelling and resource sharing; greater security, integrity and privacy of data; and, system modularity and the implications that this offers form highly structured approach to implementation.

Essentially, a distributed processing system may be thought of as an inter-connection of geographically distributed digital sub-systems. Each has certain processing capabilities and communicates with other sub-systems through the exchange of messages of various sorts - a more rigorous list of criteria has been given by Enslow (Ens78). Within such a distributed system each host node may have its own local operating system and applications software. This may be unique to a particular node. The various hosts will communicate with each other using common message transmission protocols. Two commonly used techniques for transmission of information around a network are message switching and packet switching, these will be described in more detail later. One important feature of the network is that the route information takes from an originating node to its destination node will not be guranteed since this will be influenced by the state of the network at any time. To the user the system will present a common command language via the network operating system. This will usually provide a set of high level commands that enable the user to control the services and facilities that the network offers - for example, CREATE, SEND, FETCH, FIND to control the manipulation of files of stored; DATABASE XYZ, to establish connection with a particular data base system, and so on.

Since their inception, computer networks have grown from simple in-house affairs to systems that span both national and international boundaries. They have expanded across continents in order to interconnect major computing faciities around the world. Nowadays, with the advent of inexpensive micros and minicomputers (and equipment based upon them), computer connections are extending between buildings and along corridors to enable the linking together (via Local Area Networks or LANs) of officies and laoratories on a world-wide scale. Progress in this area is very rapid - particularly since the development of satellite links.

Types of network and their purpose

Computer networks may be classified by any of a wide range of possible attributes: by their topology (star, tree, loop, etc); by the control discipline used (central or distributed); by the type of information that the network carries: the mechanisms for transmitting it (message or packet switching); by the communication links involved (cable, twisted pair, radio, etc); and, the nature of the computers involved - these may be of the same type or may differ quite significantly (homogenous and hetergeneous networks, respectively). These attributes represent just a few of those that are widely used to describe and classify the different types of computer networks that currently exist. In this section some of these attributes will be briefly explained.

Network Topology

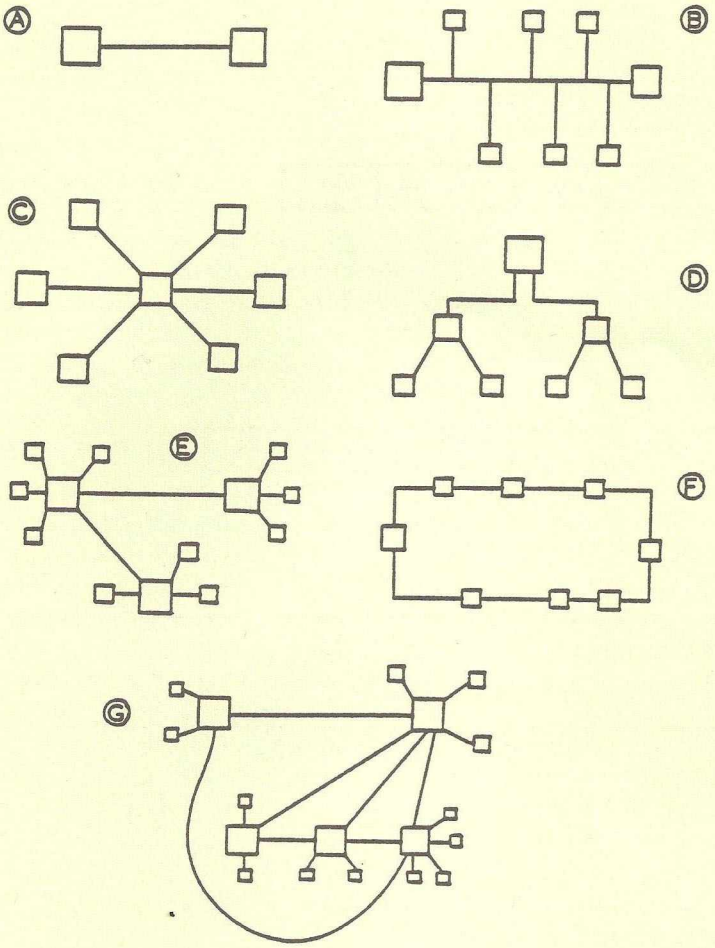
The term topology refers to the geometrical arrangement of links and nodes of a network. Within these nodes it is possible to locate several different types of hardware and software depending upon the function that an individual node is to perform. When designing a network, many different factors must be evaluated in order to choose the most suitable topology. One major factor that is likely to strongly influence this choice is the type of participation required by each of the nodes. Thus, it is possible for a node to act

1) exclusively as a consumer of resources,
2) exclusively as a provider or resources, or,
3) as both a consumer and provider of network resources.

Depending upon the likely resource utilisation and the way in which nodes need to communicate with each other, about half a dozen "standard" types of network topology are commonly used. These are summarised in the seven diagrams (A through G) contained in figure 2.

The simplest type of configuration is the point-to-point agreement shown in diagram A. Here two nodes are joined by a single communication link. This may be a private wire or a switched line - as in the public dial-up

FIGURE 2 BASIC NETWORK STRUCTURES



telephone network. A more complex arrangement of nodes is illustrated in the multi-point system depicted in diagram B. This requires that several nodes share the same communication link. One of the nodes in the network is designated as the controller and the others then become tributory stations. The controller manages network traffic by means of polling, that is, it invites other stations to send messages in turn. Multi-point networks are usually established over non-switched leased lines.

Diagram C shows another popular network arrangement called a centralised star system. In this configuration all users communicate with a central point that has supervisory control over the system. Peripheral nodes can only communicate with each other via the central controller. This thus provides a central message switching service for the other nodes.

A typical hierarchical structure is shown in sketch D. Such an arrangement is often employed in industrial environments to supervise a variety of real-time process monitoring applications. A hierarchy of computers is used to control various processes, synchronise them and report on their status. Both microcomputers and minicomputers are used as nodes in this type of arrangement. These occupy the lower levels of the tree structure with, perhaps, a mainframe or large minicomputer at the top.

Many organisations design their computer networks in the form of a loop or ring structure (diagram E). In an arrangement of this type there is a common communication loop to which all nodes are attached. The data to be transmitted is then looped around the nodes in turn. A loop or ring arrangement of this type is very economical when several remote stations and host processors are located near to each other - perhaps within the same building or distributed over a manufacturing plant. However, when the stations are geographically dispersed over long distances the line costs would probably be too expensive for a loop structure and a cheaper form of distributed network would probably be required. Two of these are described below.

The multi-star network similar to that shown in diagram F is an often used configuration in which there are several supervisory or exchange points each with their local cluster of attached nodes. The local hosts usually service the requirements of their attached nodes but also permit general communication between any of the other points in the network. If properly designed, distributed networks can offer significant reliability advantages, since a failure at one node does not affect the rest of the configuration. Indeed, in applications where the reliability of continuous communication is important, a fully distributed network (diagram G) in which every point is connected to several neighbouring points may be preferred. The additional transmission paths provided by this type of structure improves the overall performance of the system quite substantially. When using this type of topology, detailed traffic analysis must usually be performed in order to determine where the links are required.

The network structures described above and illustrated in figure 2 represent the most common types of discrete network architecture. It is feasible, however, to use these as basic building blocks to construct even more complex arrangements. Thus, two, three, or more, networks having topologies similar to that shown in diagram G may be interconnected to form a highly distributed arrangement of nodes. Logically, the arrangement will appear as three separate networks linked at particular points. Because the individual networks will require to retain various attributes of autonomy, and, because they will differ considerably from each other in their characteristics, special modes of interconnection are required. Nodes that are used to interlink networks of different types in this way are called gateway nodes. Their design has been described by a number of authors (Hig 75, Wal 75). A description of one such gateway that connects the University of Rochester to the APRA network in the USA has been presented by Ball et al (Bal76).

Circuit, Message and Packet Switching Networks

There are three basic methods for routing communications traffic from a source to a destination within a

computer network: circuit switching, message switching and packet switching. In a circuit switching network - similar to the public switched telephone system - the role of the switching centre(s) is to establish a direct connection between nodes in the network. Once established these may then carry on one-way or two-way communication with minimal delay between the transmission of a message and its arrival at its destination. When communication is complete, the switching centres disconnect the circuit and restore the system in readiness for other connections. Circuit switching often requires long connect times and ties up transmission capacity for long periods. This arises because of a fundamental property of circuit switching - once a path is determined through the network nodes, all traffic between a source and destination pair then follows the same path.

An alternative mode of transmission which does not require a fixed route between source/destination could have many advantages. Using such an approach, two possibilities exist - depending upon the volume of data to be transmitted: message switching and packet switching. In message switching, each item of data is sent into the network as a discrete unit and is then routed to its destination. The format of the unit that is transmitted is illustrated in diagram A of figure 3.

A message makes its way through the network to the destination whose address is specified in the header. Each node in the network uses an appropriate routine algorithm in of order to decide which node the message has to got to next in order to reach its destination. Since some stations may be busy, a message may often have to be stored at intermediate nodes before it is passed on. For this reason an arrangement such as its often called a 'store and forward' system.

Packet switching is essentially similar to message switching and is used when large volumes of information are to be transmitted. At the source station a large message is subdivided into a series of fixed length segments (called packets) of size 1,000 -8,000 bits. Each packet has a unique number associated with it to enable

FIGURE 3 MESSAGE AND PACKET STRUCTURE

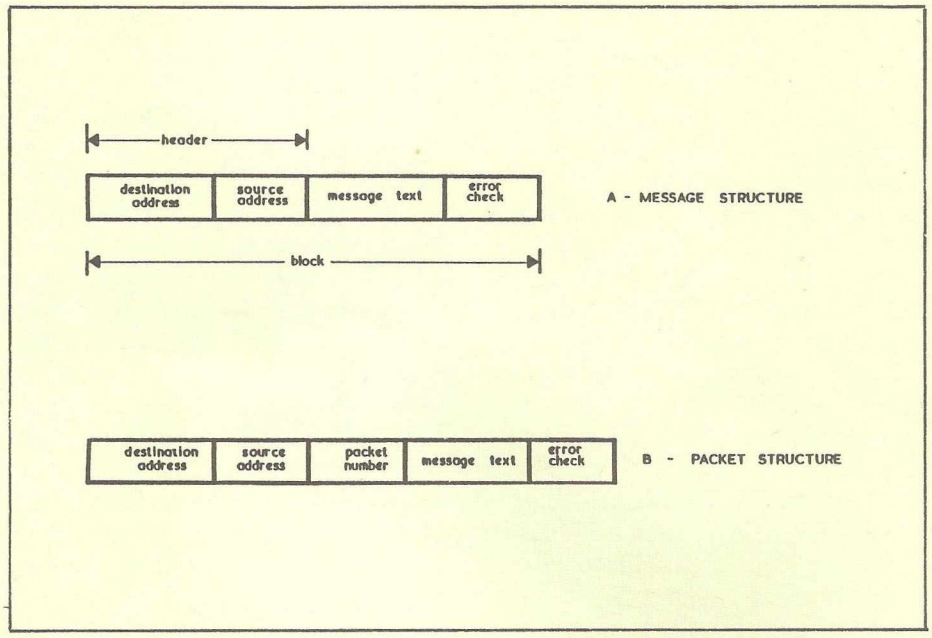
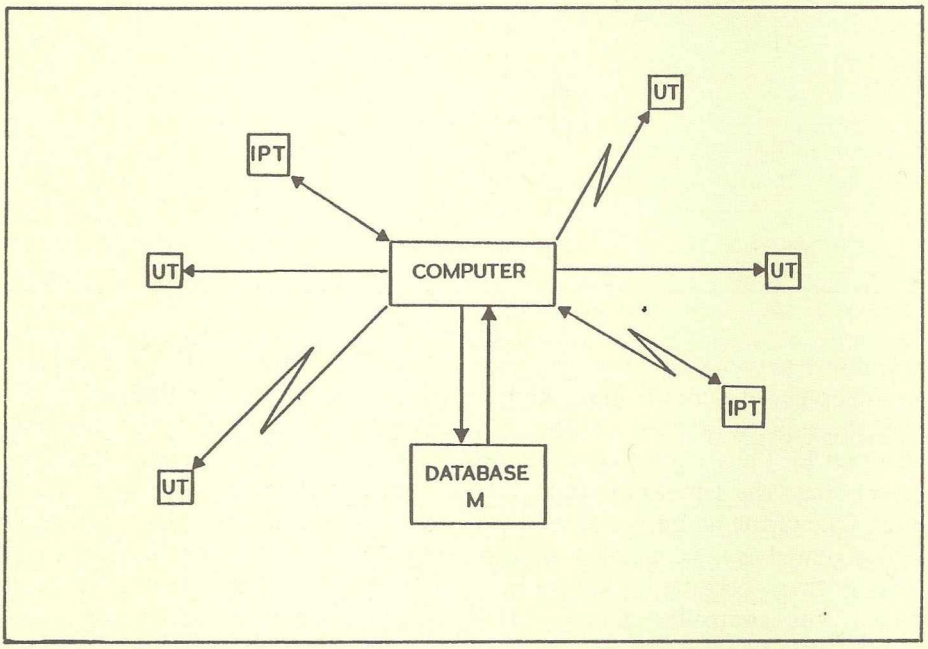


FIGURE 4 SIMPLE VIDEOTEX NETWORK



the reconstruction of the complete message at the destination. The format of a packet is similar to that of a message and is shown in figure 3 (diagram B).

Packets are treated individually and are forwarded along the best available route, that is, the route with the shortest transmission delay. Each packet is checked for errors at each node along the way by means of the error checking field contained in the packet. Because long messages are broken up and sent over different routes it is possible for them to arrive at their destination more quickly. Furthermore, since intermediate nodes in a packet switched network only have access to segments of the whole they are unable to assemble the entire message. Thus, if data encryption is not being used, transmission is more secure.

Videotex Networks

Videotex networks (Bal80) were orignally introduced to provide low cost public data and information retrieval networks based upon broadcast TV signals or a switched telephone network. Intended primarily for use as public information utilities the systems were designed around the use of a single tree structured data base. Modified TV sets are used as user terminals through which could be implemented a variety of

menu selection techniques in order to facilitate data/information retrieval operations. Figure 4 shows the way in which the components of a typical videotex network might be interconnected.

There are two types of terminal: IPT - the Information Provider's Terminal and UT - the User's Terminal. The information provider is the person responsible for entering data into the data base and ensuring its correctness. The arrangement of components is essentially a star network with the computer at the centre and the terminals and videotex data base (M) attached as peripheral nodes. This type of equipment is often used for the provision of in-house information systems - for a laboratory, operations room or sales office. In additon to their prime use as information retrieval tools, the two-way communication capability of many of these systems enables the implementation of a wide variety of 'electronic mail' and 'electronic journal' facilities. On a larger scale such systems are used to provide global or national information utilities. Typical examples of systems of this type include PRESTEL (UK), BILDCHIRMTEXT (Germany), ANTIOPE (France), and TELIDON (Canada). These videotex networks are widely used for the provision of a variety of commerical, scientific and technical information to the general public.

The Need for Standards

One of the basic pre-requisites necessary for the construction of a viable computer network is the ability to easily connect various kinds of computing equipment to a communication sub-system. Unfortunately, this is not always as easy in practise as it is in theory. Most people have probably experienced the problems encountered when an attempt is made to interface one electronic device (such as a computer) to another (say, a printer or laboratory instrument). The problems can be quite substantial unless some form of standard interface is used. For peripheral interfacing the IEEE-488 or RS-232-C conventions are now commonly used. Analogous interfacing standards also exist to facilitate computer networking.

When network components are inter-connected many problems can arise unless agreed upon conventions are used. Because of their much greater complexity, there will obviously be many more difficulties associated with the formulation of standards for networks compared with those necessary for simple computer-peripheral interfacing. Usually, two classes of standard are needed: one to deal with hardware related matters and another to cater for software factors. Hardware related standards specify how components are to be physically connected with each other. Here, typical considerations include electrical signal characteristics, signal speeds, signal types, electrical connector dimensions, etc). Software standards deal with the way in which information is transmitted around the network. Important factors in this category are line protocols, error checking algorithms, encryption methods and so on. Most of these are adequately documented in the technical literature.

The numerous documents that are available cover a wide variety of data transmission and networking standards. Most of these originate from manufacturers (product specific) or from international standards such as CCITT, EIA and ISO. Those originating from the latter are usually product independents. Much of the more important material relating to standards is covered in text books on telecommunications (Dav73, Can80). One of the more useful introductory hand-books is the "V-Series Report" (Boo81). This explains most of the important technical telecommunications terminology in a readable and digestable fashion. It also contains details of the popular data transmission standards and describes their relevance to networking. Anyone intending to use a data communication system would find this book a valuable asset. At a more advanced level another useful publication is the "Network Independent File Transfer Protocol" (HLP81). This specifies a file transfer scheme that permits the exchange of information files of any kind over any type of data communication network (public or private) regardless of the nature of the hardware that is used. This latter standard is thus orientated towards some of the software aspects of networking.

In the future it is likely that more and more organisations will become involved in computer networking -perhaps, through public packet switching systems such as the UK's PSS. As this happens the continous evolution of appropriate standards will be a necessary prerequisite if acceptable user-orientated systems are to be produced. As will be discussed in the next section, one the the largest potential growth areas for networking facilities is likely to be users of personal computer systems. Activity in this domain will undoubtedly necessitate the formulation of yet more networking standards.

The Implications of Computer Networks

A few years ago, pocket calculators were very much the fashion. Most people seemed to own one -shopkeepers, business personnel, managers, house-wives, teachers and students. Unfortunately, despite their popularity and significant utility, for the average user calculators were never exceptionally exciting machines. After all, what could they do? Their repertoire was extremely narrow. They were only capable of performing numerical calculations in a way which was faster, more precise and (one trusts) less error prone than computations performed manually, via a slide rule or using log tables.

More recent developments in electronics, computing and communications technology now put us at the forefront of what is likely to be yet another significant era in the development of the human species - the widespread availability of the personal computer. Unlike the pocket calculator, this new device is far more interesting since it has a potential to act as:

1) a powerful numerical calculator,
2) a processor for many different kinds of textual, sonic and graphic information,
3) a large capacity storage archive for a wide variety of personal data and information,
4) a sophisticated learning medium, and,
5) a channel for interpersonal communication.

As society moves towards an age of 'personal computing' and to enable each of the above objectives to be realised, there is a going need to facilitate the easy exchange of pro-

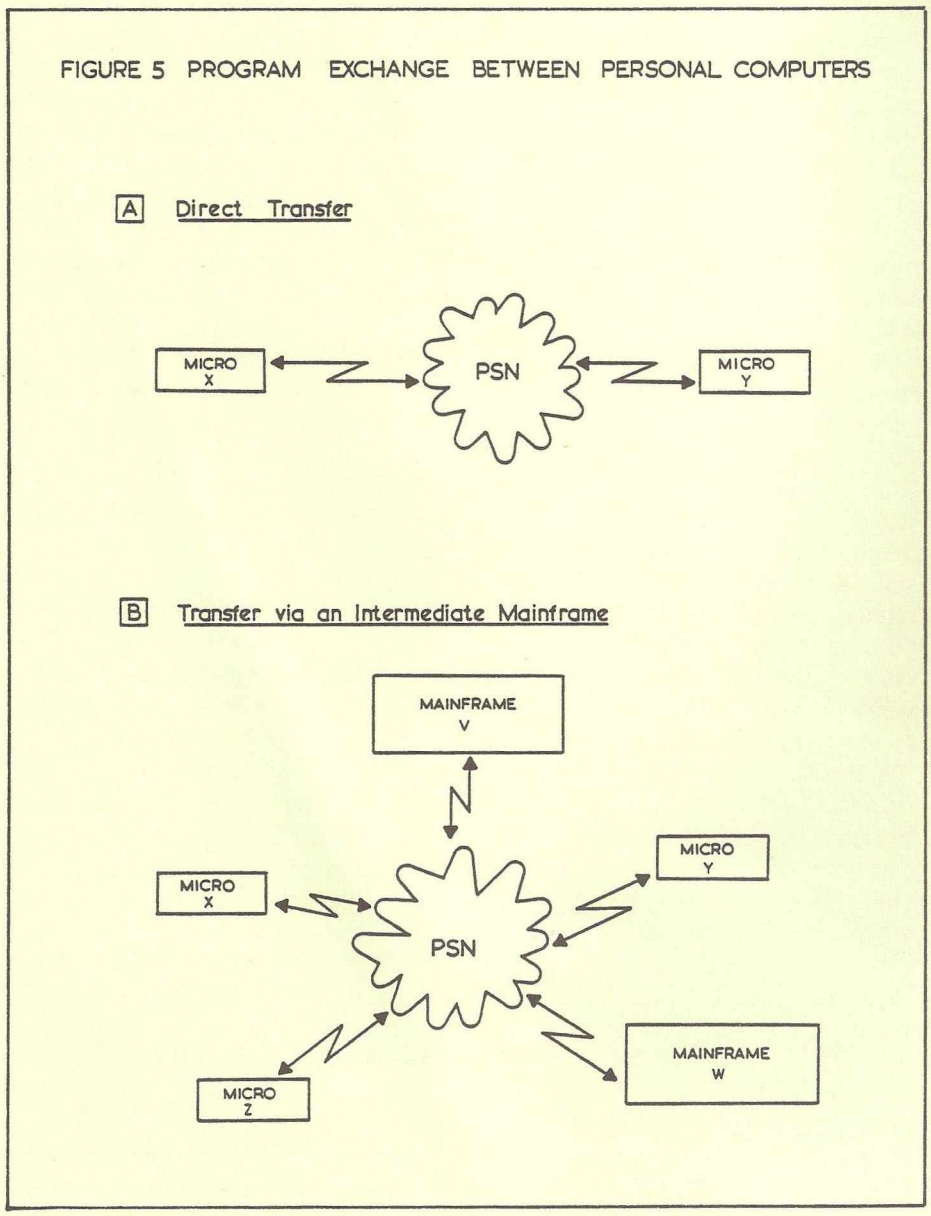
grams and data between one person and another. Ideally, this will be achieved by means of some form of commonly available global communication network. The more important approaches to networking have been described in the previous section. Many of these techniques are now being explored. Some of the more important developments within the UK are:

a) dissemination via broadcast teletext facilities, (for example, ORACLE and CEEFAX),
b) distribution via viewdata systems, (for example, PRESTEL), and,
c) use of a distributed computer network, (for example, NUNET, POLYNET).

Each of these approaches has its advantages and disadvantages. ORACLE and CEEFAX obviously only allow one way transmission of information from a central depository to a consumer population. Both the BBC and ITV are currently investigating this type of 'telesoftware' distribution (Hay81). The PRESTEL approach partially removes the unidirectional limitations of CEEFAX and ORACLE. This is possible because some degree of 'two-way' exchange of information is permitted. In this type of system, there are two categories of user - the ordinary customer (who just retrieves and examines pages of stored material) and the information provider. The major drawback of this system is that not all users can be information providers. The Council for Educational Technology is currently investigating this mode of information dissemination in conjunction with a number of schools and colleges (CET 81).

The third approach to program and data distribution is via a truly distributed computing network (Can80, Dav73). Such a system has the advantage of enabling totally unrestricted bi-directional interchange of data between any two users. In the remaining part of this article the use of the public switched network (PSN) as a means of distributing programs and data between users of personal computers will be described.

FIGURE 5 PROGRAM EXCHANGE BETWEEN PERSONAL COMPUTERS



Tranmission of Source Programs between Microcomputers

The architecture of the distributed computing system will significantly influence the types of data transfer that it will support. However, two broad types are generally feasible - depending upon whether or not there is any intermediate storage of material. These are illustrated schematically in figure 5.

In case A, the micro owner at site X is able to dial the telephone number of the micro at site Y and then transmit information to it. In the context of data exchange, transmission takes place as if the two micros were directly linked together (Cam81). However, because messages passing over the communication network are likely to be influenced by noise, suitable error detection and correction procedures need to be added to the software in the micros. This mode of data exchange does not utilise any intermediate storage of the information being transmitted. Consequently, if either micro fails to respond to a call from its incipient partner, no transfer can take place.

An alternative method, shown in diagram 5B, illustrates a situation in which a micro at location X can archive material in a mainframe at site V or W. At some later stage the stored material can be retrieved by micro owners at sites X, Y or Z - provided, in the case of the latter two, they can meet the necessary access control requirements. Through this kind of technique the sharing of information is a trivial problem. What is more important, however, is that it now becomes a simple matter to physically distribute the information to many different geographical locations. Details of using a microcomputer as an interactive terminal device over the public switched telephone network have been described elsewhere (Bar 81a, Bar 81b), Similarly, the use of a microcomputer as an

intelligent terminal has also been outlined (Bar81c). In this latter work algorithms to enable the transfer of files of information between a mainframe and a microcomputer were described in some detail. Implementations of these algorithms have also been presented. Essentially, the programs that have been constructed allow complete mobility of files between micro and mainframe systems. These files may contain machine code programs, high level (source code) programs or data. Using this software the movement of programs between one micro and another (via a mainframe) is a reasonably straightforward task. However, a decision must be made regarding whether these should be 'shipped' around in machine code or source level format.

Historically, program portability has always been significantly influenced by four important factors,

a) the use of high level languages,
b) the availability of internationally accepted language standards,
c) the extent to which programmers remain within the limitations imposed by (b), and,
d) the architectural differences between the computers on which the programs run.

These factors alone are probably sufficient to justify any decision to transmit program files in source code format rather than as machine code memory images. In this context we have been examining the problems associated with transmitting both PASCAL and BASIC programs over the PSN between micros and mainframes. Some interesting results have been observed - one of which is described in the next section of this article.

The files that are transmitted between the two computers are organised in the form of a contiguous set of characters. Certain special ones (for example, end of line-$OD) interspersed in the sequence serve to impose a simple record structure on these files. Although they might not be physically stored in this way on either the source or destination computers, this simplistic view of their structure is sufficient to enable the ensuing discussion to be followed.

Loading BASIC Programs from Secondary Storage

Once a BASIC program has been transmitted from a remote computer and stored locally on a secondary storage device (either tape or disk) it is a simple matter to load this into memory for subsequent execution. The exact mechanism for loading a program will obviously depend upon the type of microcomputer that is used. For the purpose of illustration, in the example that is described below a Commodore PET (3000 series) has been employed.

The function of a 'loader' program is quite simple. It has to read BASIC statements contained in a secondary storage file, convert these to the appropriate internal format and store them at the correct location within PET's memory space. The functional requirements of such a program are summarised in diagram A of figure 6. As can be seen from this simple illustration, the storage area for BASIC programs commences at $0400 and extends upwards to $7FFF (for a 32K machine). However, because the loader program itself occupies a section of memory just below the upper limit, there will be an inherent restriction on the size of the programs that it can handle.

One of the major tasks that the loader has to perform is the conversion of program code from its external format into its internal representation in 'tokenised' form. When represented in this way each of the keywords and special operators in the language is internally stored as a single 8-bit numeric value. A comparison of the external and internal forms of a small BASIC program is shown in diagram B of figure 6 shown overpage. In internal format, each statement consists of a two byte pointer, a two byte encoding of the statement number, a sequence of bytes representing the tokenied version of the original souce line and finally, a byte containing an end-of-line marker.

In figure 6B underlining is used to indicate the position of the statement numbers in the internal representation of the program. The structure of BASIC statements and programs are discusse in more detail in most of the standard PET handbooks (Don81). Once a statement has been tokenised, it has to be inserted into its correct position in memory. Both tokenisation and insertion are usually achieved by means of special routines built into the PET operating system (residing at ROM locations $C34B through $C43F). There is no inherent reason why this code cannot be employed by user programs. However, in order to utilise it in the desired way a slight modification to the ROM code would be necessary. Because it is not possible to alter this, the relevant section of the PET's operating system must be copied across to RAM and used from there. This is easily achieved by means of a simple assembly language program that is written in such a way that when it is assembled a gap is left in the normally contiguous sequences of address allocated to it. Later, when the program is executed, this gap can be filled by the appropriate copied code.

The basic algorithm upon which the implementation of the loader depends is shown in figure a.

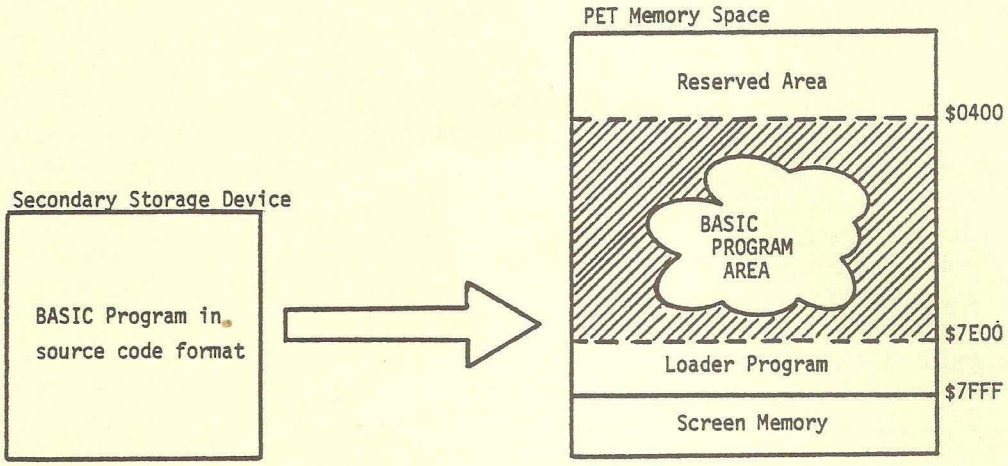
As was mentioned earlier, step 7 will be performed by 'borrowed code' while the remaining steps will be implemented via user written statements (see figure 6C). A listing of an assembly language program that implements the above algorithm - for BASIC source files residing on casette tape - is shown in figure 7. Ac-

figure a.

```
Start:   Borrow code from the operating system.
Step 1:  Initialise  PET BASIC  system - (perform a NEW).
Step 2:  Read input file (Get next source character).
Step 3:  If end-of-line detected then go to step 6.
Step 4:  If end-of-file detected then go to step 8.
Step 5:  Store source character in  BASIC  buffer area.
         Go to step 2.
Step 6:  Prepare for entry into the operating system
         routines.
Step 7:  Tokenise the source statement held in the  BASIC
         buffer area.
         Enter tokenised statement into  BASIC  memory area.
         Go to step 2.
Step 8:  Pass control back to  BASIC  command mode with a
         "READY"  message.
```

FIGURE 6 FUNCTION OF A SOURCE CODE LOADER

[A] PRINCIPLE OF OPERATION



[B] COMPARISON OF INTERNAL AND EXTERNAL FORMS OF A BASIC PROGRAM

SOURCE CODE

```
10 PRINT"HELLO"
20 X=3+2
30 Y=3*2
40 PRINT X,Y
50 PRINT"GOODBYE"
```

INTERNAL FORMAT

```
0400  00 0E 04 0A 00 99 22 48
0408  45 4C 4C 4F 22 00 18 04
0410  14 00 58 B2 33 AA 32 00
0418  22 04 1E 00 59 B2 33 AC
0420  32 00 2C 04 28 00 99 20
0428  58 2C 59 00 38 04 32 00
0430  99 22 47 4F 4F 44 42 59
0438  45 22 00 00 00 AA AA AA
```

[C] MEMORY MAP FOR A TYPICAL LOADER

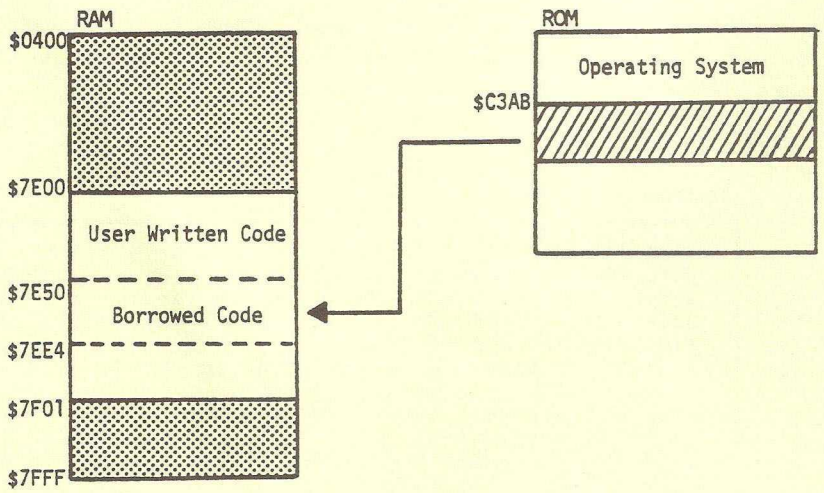


FIGURE 7 BASIC SOURCE CODE LOADER WRITTEN IN ASSEMBLER

```
0001  0000                ;
0002  0000                ;TAPE LOADER PROGRAM
0003  0000                ;*******************
0004  0000                ;
0005  0000                TAPERD=$F855              ;TAPE READ ROUTINE
0006  0000                DEVICE=$D4                ;INPUT DEVICE NUMBER
0007  0000                BASBUF=$0200              ;BASIC BUFFER
0008  0000                BUFFER=$027A              ;TAPE BUFFER
0009  0000                NEWSYS=$C55D              ;PERFORM "NEW"
0010  0000                CHRGET=$0070              ;LINE FETCH ROUTINE
0011  0000                OSCODE=$C3AB              ;START OF CODE TO BE COPIED
0012  0000                BASIC=$C389               ;BACK TO BASIC
0013  0000                      *=$7E00
0014  7E00  A0 00       START  LDY #$0
0015  7E02  B9 AB C3    COPY   LDA OSCODE,Y       ;COPY OS CODE
0016  7E05  99 50 7E           STA INSERT,Y       ;INTO THIS PROGRAM
0017  7E08  C8                 INY
0018  7E09  CC FE 7E           CPY NUM            ;ALL DONE?
0019  7E0C  D0 F4              BNE COPY           ;NO - GET SOME MORE
0020  7E0E  20 5D C5    STEP1  JSR NEWSYS         ;INITIALISE SYSTEM
0021  7E11  20 F0 7E           JSR TPREAD         ;READ A TAPE BLOCK
0022  7E14  20 F0 7E           JSR TPREAD         ;READ A TAPE BLOCK
0023  7E17  A2 00              LDX #$0            ;INITIALISE X-REGISTER
0024  7E19  A0 01              LDY #$1            ;INITIALISE Y-REGISTER
0025  7E1B  B9 7A 02    STEP2  LDA BUFFER,Y       ;GET A CHARACTER FROM TAPE BUFFER
0026  7E1E  C9 0D              CMP #$0D           ;IS IT END OF LINE?
0027  7E20                ;
0028  7E20  F0 1B       STEP3  BEQ STEP6          ;YES
0029  7E22  C9 00              CMP #$00           ;IS IT END OF FILE?
0030  7E24  D0 03       STEP4  BNE STEP5          ;NO
0031  7E26  4C EC 7E           JMP STEP8          ;YES
0032  7E29  9D 00 02    STEP5  STA BASBUF,X       ;STORE IN BASIC BUFFER
0033  7E2C  E8                 INX                ;INCREMENT X-REGISTER
0034  7E2D  C8           LY     INY                ;INCREMENT Y-REGISTER
0035  7E2E  C0 C0              CPY #$C0           ;END OF DATA BLOCK?
0036  7E30  F0 03              BEQ GBLOCK         ;YES - GO GET ANOTHER
0037  7E32  4C 1B 7E           JMP STEP2          ;NO - GO GET NEXT CHARACTER
0038  7E35  20 F0 7E    GBLOCK JSR TPREAD         ;GET ANOTHER BLOCK
0039  7E38  A0 01              LDY #$1            ;INITIALISE COUNT
0040  7E3A  4C 1B 7E           JMP STEP2          ;RETURN TO MAIN LOOP
0041  7E3D  A9 00       STEP6  LDA #$00           ;PUT EOL INTO BASIC
0042  7E3F  9D 00 02           STA BASBUF,X       ;BUFFER AREA
0043  7E42  A2 FF              LDX #$FF           ;SET UP POINTERS TO THE
0044  7E44  86 77              STX $77            ;ADDRESS OF BASIC
0045  7E46  A2 01              LDX #$01           ;SOURCE TO BE PROCESSED
0046  7E48  86 78              STX $78
0047  7E4A  8C 00 7F           STY YIND           ;SAVE CONTENTS OF Y-REGISTER
0048  7E4D  20 70 00    STEP7  JSR CHRGET         ;INVOKE CHRGET ROUTINE
0049  7E50                INSERT=*                  ;TOKENISE STATEMENT
0050  7E50                      *=*+$94             ;AND INSERT RESULT INTO
0051  7EE4                                        ;BASIC PROGRAM AREA
0052  7EE4                                        ;GET READY TO GO BACK AND
0053  7EE4                                        ;PROCESS NEXT STATEMENT
0054  7EE4  AC 00 7F           LDY YIND           ;RESTORE Y-REGISTER
0055  7EE7  A2 00              LDX #$0            ;INITIALISE X-REGISTER
0056  7EE9  4C 2D 7E           JMP LY             ;GO BACK TO MAIN LOOP
0057  7EEC  EA          STEP8  NOP                ;END OF FILE
0058  7EED  4C 89 C3           JMP BASIC          ;RETURN TO BASIC
0059  7EF0                                        ;
0060  7EF0                                        ;ROUTINE TO READ A TAPE BLOCK
0061  7EF0                                        ;
0062  7EF0  A0 01       TPREAD LDY #$1            ;LOAD INPUT DEV
0063  7EF2  84 D4              STY DEVICE         ;PASS TO OPSYS
0064  7EF4  8E FF 7E           STX XIND           ;SAVE X-REGISTER
0065  7EF7  20 55 F8           JSR TAPERD         ;READ A TAPE BLOCK
0066  7EFA  AE FF 7E           LDX XIND           ;RESTORE X-REGISTER
0067  7EFD  60                 RTS                ;RETURN TO CALLER
0068  7EFE                                        ;
0069  7EFE                                        ;DATA STORAGE AREAS
0070  7EFE                                        ;
0071  7EFE  94           NUM    .BYTE $94          ;NUMBER OF BYTES TO COPY
0072  7EFF  00           XIND   .BYTE 0            ;PLACE FOR X-REGISTER
0073  7F00  00           YIND   .BYTE 0            ;PLACE FOR Y-REGISTER
0074  7F01                      .END
```

companying this, in figure 8, is a complementary data flow diagram.

When invoked the initialisation code copies $94 bytes (starting from $C34B) down to the vacant slot previously reserved for it by suitable manipulation of the assembler location counter. As soon as this has been completed the loading operation starts. The program depends upon a subroutine called TPREAD to transfer a block of data from cassette into its associated buffer area. In turn, this routine calls the operating system utility code commencing at $F855. Characters are then copied one at a time from the tape buffer ($027A) across to the BASIC input buffer ($0200 - $0250) using the Y- and X-registers, respectively, as pointers in indexed load and store operations. Each time an end-of-line character ($OD) is encountered in the input stream (INCHAR) an end-of-statement marker ($OO) is sent to the output stream (OUTCHAR) for placement in the BASIC buffer. Subsequently, at step 6, the pointers at $77 and $78 are set up so that they point to the memory area containing the new statement. A subroutine call to the operating system utility CHRGET routine has been printed essentially a line fetch routine that sets up the next basic statement for processing. Its detailed mode of operation is described elsewhere (Ham80, Doy80a, Doy80b). Once the CHRGET routine has been primed the code for tokenising/inserting the new line into the BASIC program area can commence execution. Further source statements are then processed one at a time until an end-of-file code ($OO for tape files) detected on INCHAR terminates the loading process and passes control back to BASIC direct command mode with the prompt "READY".

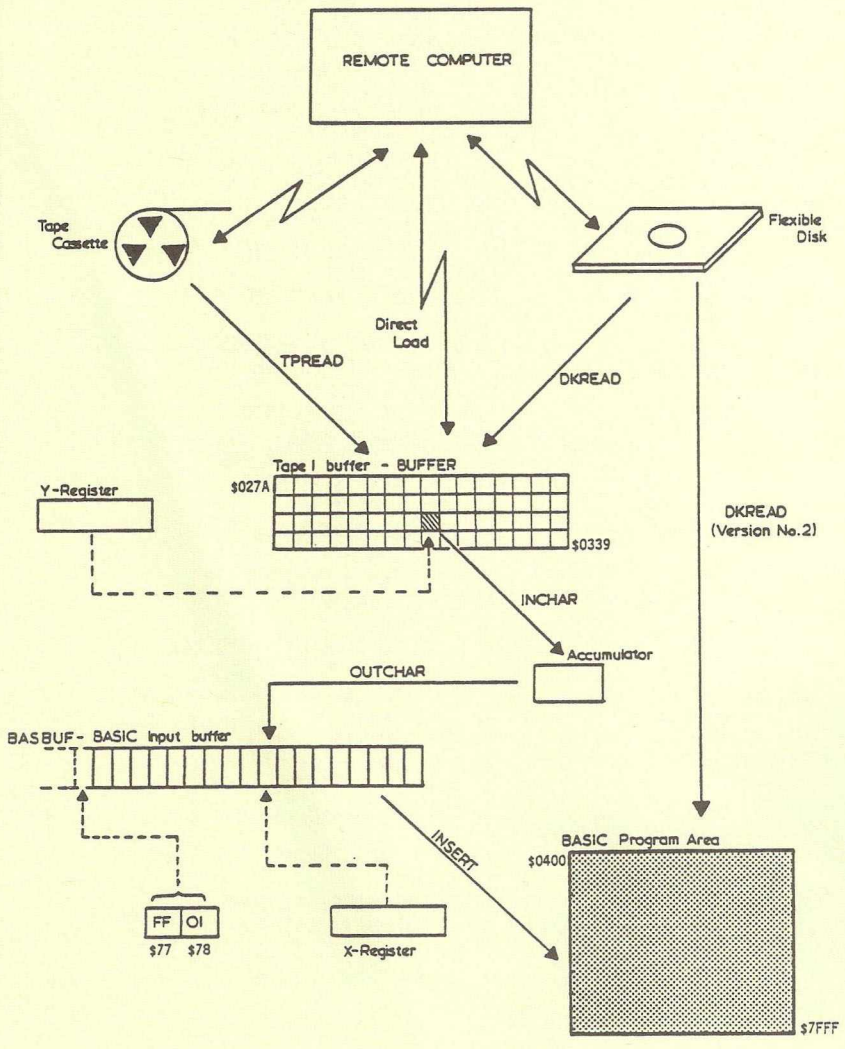
A major disadvantage of the loader shown in figure 7 is its lack of identity checking. Inherent in the program is the assumption that the tape will be positioned at the point from which loading is to commence; the first (program identity) block is then skipped over. If it was necessary it would be a simple matter to replace the first reference to TPREAD (line 21) by a call to a subroutine that provides a better user interface. This could be designed in such a way that it interrogates the user in order to ascertain the name of the file to be loaded and then, automatically, positions the tape ready for loading. A routine of this type is, of course, a necessity for the version of the loader used to handle source programs that are resident on disk (Bar81d).

It is interesting to compare the load times of BASIC programs in their different formats. In order to do this a 19 Kbyte program was transmitted over the PSN and stored on cassette tape. This was then loaded into memory using the tape loader and its load time was recorded. The loaded program was then SAVEd (in memory image format) onto cassette tape and reloaded using a LOAD command. As might be expected the latter process is much faster (by factor of about 2.9) than source code loading. However, in view of the fact that programs in source form are more easily exhanged than their equivalent machine code versions, the overhead is an acceptable one.

Analogous programs exist to enable the loading of source programs from flexible disk (DKREAD in figure 8). These embody the same princliples as the tape loader but, as might be anticipated, are much faster in operation. Details of these are given elsewhere (Bar81d).

The program illustrated in figure 7 is assembled to reside in a section of the available RAM area of the microcomputer. This means that the amount of storage available for loading BASIC programs is less than might normally be the case. Should the need arise, this limitation may easily be overcome by utilising the expansion ROM sockets of the PET. It is a simple matter to re-assemble the loader program so that it resides in a write-enabled IROM module (Bar81e) located a base address $9000. All demands of the loader for BASIC RAM can thus be totally removed.

FIGURE 8 DATA FLOW DIAGRAM FOR SOURCE CODE LOADER



Conclusion

The fundamental nature of a computer based communication system has been described. Currently, three broad approaches to data and program distribution are being actively investigated: broadcast teletext, viewdata systems and distributed computing networks. Of these, only the latter provides true bi-directional information exchange mechanisms to be implemented. Such systems thus have significant potential for program exchange between users of personal computers. As those in the public sector gain easier access to appropriate communication systems (such as PSS and IPSS) program interchange (particularly between educational establishments) should become much easier. The nature of the software necessary to achieve source program exchange has been described by means of an appropriate illustration.

Start of a new training course with Mike Gross-Niklaus!

References

Bal76 Ball, J.E., Feldman, J., Low, J.R., Rashid, R. and Rovner, P., RIG: Rochester's Intelligent Gateway: System Overview, **IEEE Transactions on Software Engineering,** SE-2, 4, 321-328, December 1976.

Bal80 Ball, A.J.S., Bochman, G.V. and Gecsei, J., Videotex Networks, **IEEE COMPUTER,** volume 13, No 12, 8-13, December 1980.

Bar81a Barker, P.G., **Using a Microcomputer as an Interactive Terminal,** Interactive Systems Research Group Working Paper, April 1981.

Bar81b Barker, P.G., **Using the PET as an Interactive Terminal,** Interactive Systems Research Group Working Paper, June 1981.

Bar81c Barker, P.G., **Algorithms For Intelligent Terminal Operation,** Interactive Systems Research Group Working Paper, July 1981.

Bar81d Barker, P.G., **Program Transfer via the Public Switched Network,** Interactive Systems Research Group Working Paper, July 1981.

Bar81e Barker, P.G., **Experiments with IROM and EPROM,** Interactive Systems Research Group Working Paper, October, 1981.

Boo81 Bootstrap Ltd., 9 George Ave., Blackrock, Co. Dublin, Ireland, **The V-Series Report -Stardards for Data Transmission by Telephone,** ISBN: 0-9507550-0-1, 1981.

Cam81 Campbell, G., The Commodore 8010 Modem, **CPUCN,** Volume 3, Issue 6, page 18, July 1981.

CET81 Council for Educational Technology, Paper CID 81.2, Technical Developments Programme - **Telesoftware Project. 1981.**

Can80 Cannon, D.L. and Luecke, G., **Understanding Communications Systems,** Radio Shack, ISBN: 0-89512-035-6, 1980.

Dav73 Davies, D.W. and Barber D.L.A, **Communication Networks for Computers,** - John Wiley & Sons, ISBN: 0-471-19874-9, 1973.

Don81 Conahue C.S. and Osborne, A., **PET/CBM Personal Computer Guide,** Osborne/McGraw-Hill, ISBN: 0-931988-55-1, 340-341, 1981.

Doy80a Doyle, D., Dimp: A Machine Language Routine for the PET to Handle Algebraic input, **CPUCN,** Volume 2, Issue 8, 19-20, 1980.

Doy80b Doyle, D., Dimp Revisited, **CPUCN,** Volume 3, Issue 2, page 31, 1980.

Ens78 Enslow, P.H., What is a "Distributed" Processing System?, **IEEE COMPUTER,** Volume 11, No. 1, 13-21, January 1978.

Ham80 Hampshire, N., **The PET Revealed,** Computabits Ltd., 77-78, 1980.

Hay81 Hayman, M., Brighton Project sets out on the Micro Road, **PRACTICAL COMPUTING,** Volume 4, Issue 8, 75-76 August 1981.

Hig75 Higginson, P.L. and Hinchley, A.J., The Problems of Linking Several Networks with a Gateway Computer, 453-465 in **Proc. of the European Computing Conference on Communications Networks.** Online Conferences Ltd., Uxbridge, UK, ISBN: 0-903796-05-8, 1975.

HLP81 High Level Protocol Group, Data Communication Protocols Unit, National Physical Laboratory, Teddington, Middlesex, TW11 OLW, **A Network Independent File Transfer Protocol,** February 1981.

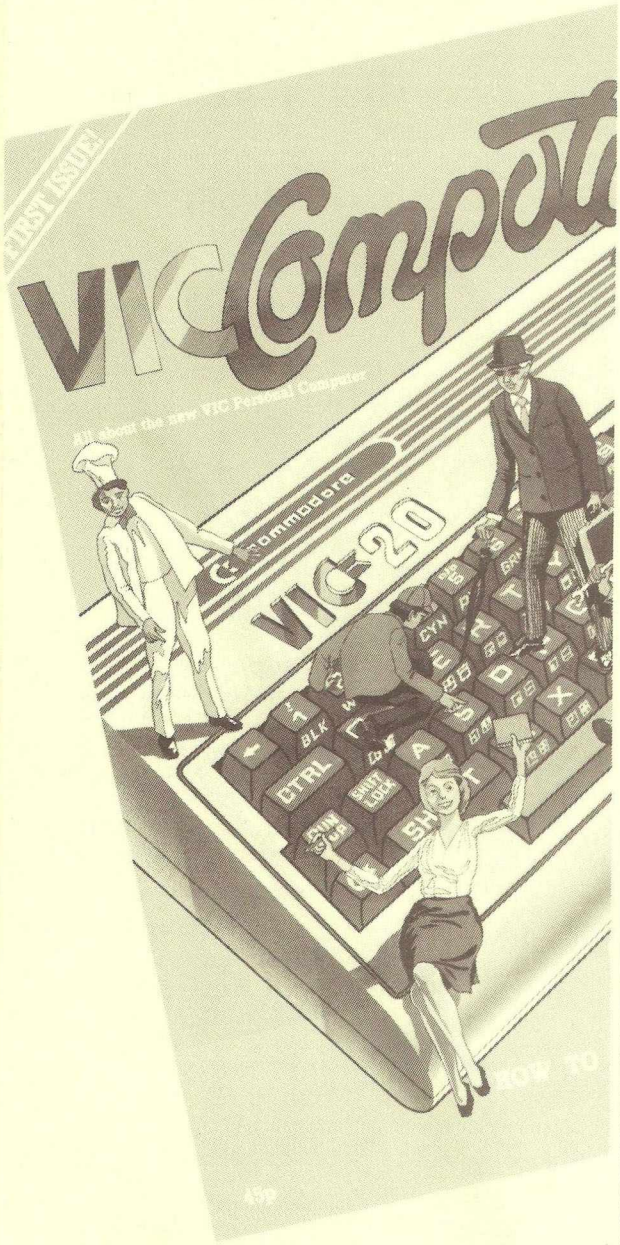
IEE80 Institute of Electrical and Electronic Engineers Inc., Supersystems for the 80's, **IEEE COMPUTER,** Volume 13, No. 11, November 1980.

Sch80 Schwartz, J.T., Ultracomputers, **ACM Transactions on Programming Languages and Systems,** Volume 2, No. 4, 484-421, October 1980.

Wal75 Walden, D.C. and Rettberg, R.D., Gateway Design for Computer Network Interconnections, 113-128 in **Proc. Of the European Computing Conference on Communications Networks,** Online Conferences Ltd., Uxbridge, UK, ISBN: 0-903796-05-8, 1975.

VIC Computing

The 'how-to' magazine all about Commodore's VIC computer



VIC Computing is a great new magazine for users of the VIC. Each issue is packed with valuable programming hints, software reviews, 'how-to' articles and program listings.

You don't have to be an expert to enjoy *VIC Computing*. It is written in straightforward English for beginners – not computer experts.

Features in the first colour packed issue included: "Anyone can Program", an article to teach you to program in one hour; "But What Can It Do?", an introduction to VIC's capabilities; "Expansion of the VIC", a guided tour of its add-on capabilities; "Using Graphics" covering programming in colour; "Converting Software for VIC" – how to convert PET programs; "VIC Sound", data sheet on sound generation; plus "Dear VIC", "Beginners Queries" and "VIC hints".

It costs just £6 a year to subscribe to *VIC Computing*. Can you afford not to?

To: Printout Publications, PO Box 48, Newbury RG16 0BD

Please enter my subscription to VIC Computing.
I enclose a cheque/Postal Order for [ ] £6 UK [ ] £IR8.50 Eire
[ ] £9 Europe [ ] $20 USA Surface [ ] $30 USA Air
[ ] £9 Rest of World Surface [ ] £16 Rest of World Air
or Charge my Access/Mastercharge/Eurocard or Barclaycard/Visa card
No: .....

NAME: .....

ADDRESS: .....

.....

POSTCODE: .....

VIC Computing

KARSSEMEYER.
Electronic mail, Library for up/down loading, Shopping list
System hours Tues to Sat 6.00 pm to 7.00 am
from 6.00 pm Sat to 7.00 am Tues

5)CBBS LONDON tel: 01-399 2136
System operator PETER GOLDMANN.
Electronic, mail, Library for downloading. Peter says more features will be added when time allows.
System hours

Wed	7.00 am to 9.00 am
	7.00 pm to 10.00 pm
Fri	7.00 pm to 10.00 pm
Sun	4.00 pm to 10.00 pm

To any person or group wishing to run or write a bulletin board program. In order to keep all computer bulletin boards compatable commands and terminal software write for details of controls, codes etc used in the above systems and most systems in the United States.
To
Frederick Brown.
421 Endike Lane.
HULL HU6-8AG.

Just to show that even Commodore can't sometimes meet the demand!

Would you buy an Approved Product from this man?

A Concurrent Clock for the PET

V.P. Cheah

This is the realization of one of those 'Wouldn't it be nice if ...' dreams. It is an online digital clock (from now on called klock) which can be used concurrently with any Basic or Assembler program which does not alter the IRQ vector in zero page. If the IRQ vector has to be altered (eg to disable RUN/STOP) then some modifications will have to be made.

Our reason for writing klock may sound peculiar to some: neither of us are watch wearers and have to rely on either the large digital clock supplied by the Birmingham Post, which is visible from most of Birmingham (except our terminal room) or on other people in the room. This soon got the better of us as we often missed the important issues of the day (eg lunch, opening times etc.) when were at the terminals. One day, someone said "Wouldn't it be nice if the PETs had online clocks like the ICL". That was when everything started ticking.

The algorithm used by klock is simple. On every 30th IRQ, the jiffy count is converted into hours, minutes and seconds. This is in turn converted into its display equivalent in screen memory and punched directly onto the screen. Although this does not cause 'snow' to fall on the 30XX and 80XX series machines, it may do so on the 20XX series machines. For minute details please refer to the program listing.

On the 80 column PET, klock is displayed on the top line. On the 40 column PET, it is displayed on the second line. The reason for this will become apparent if CLEAR and HOME are used frequently. If the display were on the first line, the cursor would have to be moved down by one line everytime otherwise a syntax error would result. Somehow this never becomes second nature: in fact it gets rather annoying after a while.

There are two routines (DIVIDE and BINBCD) which could be made more efficient (either in space or in time) but being hard pressed postgraduates with little time to spare, we have not done anything about it. DIVIDE divides a three byte number by 60. BINBCD converts a binary number in the range 0 to 59 to Binary Coded Decimal (BCD for those who only recognise short forms). This is equivalent to the DAA instruction on the 808X and 680X microprocessors. The program is fairly small (248 bytes); it may have to be modified for the ever growing range of CBM machines.

Klock given in the program listing was written for the CBM 8032. Assuming that you have typed in the program correctly, all you have to do to get it to execute concurrently with another program is given below:

POKE 52,0:POKE53,120:CLR
SYS machine language monitor
. L "O:KLOCK",80
. X
Load your program
SYS clock on
RUN

If klock in the given listing is implemented, it may be enabled by SYS 30729 and disabled by SYS30736. It should be disabled before loading any programs. If you forget, the machine will hang. If this happens, all you have to do is to hit RUN/STOP. Klock will stop ticking but the whole program will be in core.

Finally, we've found klock so useful that we have 'engraved' it in ROM on one of our machines.

```
0:KLOCK.....PAGE 0001

LINE# LOC   CODE        LINE

0002  0000               ;       PUT "@0:KLOCK,S,W"
0003  0000               ;       PUT "@1:KLOCK,S,W"
0004  0000               ;
0005  0000               ;       STARTED: 10/04/81, CUP
0006  0000               ;       UPDATED: 13/04/81, CUP
0007  0000               ;
0008  0000               ;       PURPOSE: RUN A CLOCK CONCURRENTLY WITH BASIC
0009  0000               ;                THIS IS DONE BY ALTERING THE INTERRUPT
0010  0000               ;                VECTOR.  THE CLOCK IS UPDATED AFTER
0011  0000               ;                EVERY 30TH INTERRUPT.
0012  0000               ;
0013  0000               ;-----------------------------------------------
0014  0000               ;
0015  0000                       *=  $7800
0016  7800               CLOCK   =   141          ; JIFFY COUNTER
0017  7800               CNTDWN  *=  *+1          ; COUNTDOWN
0018  7801               DISPOS  =   $8000+56     ; DISPLAY POSITION
0019  7801               HI      =   1            ; THE HIGH BYTE
0020  7801               INTVEC  =   144          ; INTERRUPT VECTOR
0021  7801               OPND    *=  *+3
0022  7804               REM     *=  *+1
0023  7805               RSLT    *=  *+3
0024  7808               RVS     =   $80          ; REVERSE FIELD
0025  7808               STDINT  =   $E455        ; STANDARD 80XX INTERRUPT
0026  7808               STOX    *=  *+1          ; TEMPORARY VALUE OF X
0027  7809               TOPSCR  =   224          ; TOP OF SCREEN
0028  7809               ZERO    =   $30          ; ZERO IN ASCII


0030  7809               ;       PURPOSE: INITIALIZE OR CANCEL CLOCK
0031  7809               ;                BY ALTERING THE INTERRUPT VECTOR
0032  7809               ;       INPUT  : CHKTIM, STDINT
0033  7809               ;       OUTPUT : INTVEC
0034  7809               ;-------------------------------------------
0035  7809               CLKOFF  = *
0036  7809  78                   SEI              ; CLOCK OFF
0037  780A  A9 55                 LDA #<STDINT
0038  780C  A2 E4                 LDX #>STDINT
0039  780E  D0 05                 BNE SETVEC
0040  7810               CLKON   = *
0041  7810  78                   SEI              ; CLOCK ON
0042  7811  A9 20                 LDA #<CHKTIM
0043  7813  A2 78                 LDX #>CHKTIM
0044  7815  85 90         SETVEC  STA INTVEC       ; SET INTERRUPT VECTOR
0045  7817  86 91                 STX INTVEC+HI
0046  7819  A0 1E                 LDY #30          ; SET COUNTDOWN
0047  781B  8C 00 78             STY CNTDWN
0048  781E  58                   CLI
0049  781F  60                   RTS
```

```
0:KLOCK.....PAGE 0002

LINE# LOC   CODE        LINE

0051  7820              ;       PURPOSE: CHECK IF IT IS TIME TO ALTER THE
0052  7820              ;                DISPLAY.  IF IT IS, THEN CONVERT THE
0053  7820              ;                JIFFIES INTO HOURS, MINUTES AND
0054  7820              ;                SECONDS OTHERWSE JUST CARRY ON WITH
0055  7820              ;                THE STANDARD INTERRUPT.  THE CLOCK
0056  7820              ;                IS UPDATED EVERY HALF SECOND.
0057  7820              ;       INPUT  : CLOCK, CORFAC
0058  7820              ;       OUTPUT : -
0059  7820              ;       USE    : OPND, REM, RSLT
0060  7820              ;       CALLS  : BINBCD, CLEAR, DIVIDE, STDINT
0061  7820              ;-------------------------------------------------------
0062  7820              CHKTIM = *
0063  7820  CE 00 78            DEC CNTDWN      ; TIME TO UPDATE?
0064  7823  D0 2A              BNE GOINT
0065  7825  A9 1E              LDA #30         ; RESET COUNTDOWN
0066  7827  8D 00 78           STA CNTDWN
0067  782A  D8                 CLD
0068  782B  A2 02              LDX #2          ; TRANSFER JIFFIES TO WORKSPACE
0069  782D  B5 8D    XFER      LDA CLOCK,X
0070  782F  9D 01 78           STA OPND,X
0071  7832  CA                 DEX
0072  7833  10 F8              BPL XFER
0073  7835  A2 00              LDX #0          ; CONVERT JIFFIES TO HMS
0074  7837  20 68 78           JSR CLEAR
0075  783A  A2 07              LDX #7
0076  783C  20 56 78           JSR DIVIDE
0077  783F  A2 04              LDX #4
0078  7841  20 56 78           JSR DIVIDE
0079  7844  AD 07 78           LDA RSLT+2
0080  7847  8D 04 78           STA REM
0081  784A  A2 01              LDX #1
0082  784C  20 C1 78           JSR BINBCD
0083  784F  A9 01    GOINT     LDA #1           ; CLOBBER TOP OF SCREEN WITH
0084  7851  85 E0              STA TOPSCR      ; ENSURE CLOCK IS NOT CLEARED
0085  7853  4C 55 E4           JMP STDINT      ; CONT WITH STD INTERRUPT


0087  7856              ;       PURPOSE: DIVIDE RSLT BY 60: INITIALLY RSLT IS
0088  7856              ;                SHIFTED INTO OPND.  THE REMAINDER IS
0089  7856              ;                IN REM.  THIS ROUTINE FALLS INTO BINBCD.
0090  7856              ;       INPUT  : X [PRINTINT POSITION]
0091  7856              ;       OUTPUT : REM
0092  7856              ;       USE    : OPND, RSLT
0093  7856              ;       CALLS  : -
0094  7856              ;-------------------------------------------------
0095  7856              DIVIDE = *
0096  7856  AD 07 78           LDA RSLT+2      ; XFER RESULT TO OPND
0097  7859  8D 03 78           STA OPND+2
0098  785C  AD 06 78           LDA RSLT+1
0099  785F  8D 02 78           STA OPND+1
0100  7862  AD 05 78           LDA RSLT
0101  7865  8D 01 78           STA OPND
0102  7868  A9 00    CLEAR     LDA #0           ; CLEAR RSLT & REMAINDER
0103  786A  8D 04 78           STA REM


0:KLOCK.....PAGE 0003

LINE# LOC   CODE        LINE

0104  786D  8D 05 78           STA RSLT
0105  7870  8D 06 78           STA RSLT+1
0106  7873  8D 07 78           STA RSLT+2
0107  7876  AD 01 78           LDA OPND        ; FIND NO OF BITS IN DIVIDEND
0108  7879  F0 04              BEQ BIT16
0109  787B  A0 18              LDY #24
0110  787D  D0 1C              BNE SHIFT
0111  787F  AD 02 78    BIT16   LDA OPND+1
0112  7882  F0 0D              BEQ BIT8
0113  7884  8D 01 78           STA OPND
0114  7887  AD 03 78           LDA OPND+2
0115  788A  8D 02 78           STA OPND+1
0116  788D  A0 10              LDY #16
0117  788F  D0 0A              BNE SHIFT
0118  7891  AD 03 78    BIT8    LDA OPND+2
0119  7894  F0 28              BEQ DISRQD
0120  7896  8D 01 78           STA OPND
0121  7899  A0 08              LDY #8
0122  789B  0E 03 78    SHIFT   ASL OPND+2       ; SHIFT OPND
```

```
0123  789E  2E 02 78          ROL OPND+1
0124  78A1  2E 01 78          ROL OPND
0125  78A4  2E 04 78          ROL REM
0126  78A7  2E 06 78          ROL RSLT+1
0127  78AA  2E 05 78          ROL RSLT
0128  78AD  38                SEC             ; CHECK IF > 60
0129  78AE  AD 04 78          LDA REM
0130  78B1  E9 3C             SBC £60
0131  78B3  90 06             BCC NEXBIT
0132  78B5  8D 04 78          STA REM
0133  78B8  EE 07 78          INC RSLT+2
0134  78BB  88          NEXBIT DEY
0135  78BC  D0 DD             BNE SHIFT
0136  78BE  8A          DISRQD TXA              ; DISPLAY REQUIRED?
0137  78BF  F0 29             BEQ ENDDIS


0139  78C1              ;      PURPOSE: CONVERT A BINARY NUMBER TO TWO DECIMAL
0140  78C1              ;               NUMBERS IN ASCII FORM.
0141  78C1              ;      INPUT  : REM, RVS, ZERO, X [PRINT POSITION]
0142  78C1              ;      OUTPUT : DISPOS
0143  78C1              ;      USE    : STOX
0144  78C1              ;      CALLS  : -
0145  78C1              ;-------------------------------------------------------
0146  78C1              BINBCD = *
0147  78C1  8E 08 78          STX STOX        ; FIND MS AND LS
0148  78C4  A0 00             LDY £0          ; NIBBLES OF BCD NO
0149  78C6  AD 04 78          LDA REM


O:KLOCK.....PAGE 0004

LINE£ LOC   CODE        LINE


0151  78C9              ;
0152  78C9              ; THIS IS SIMILAR TO THE DAA INSTRUCTION ON THE
0153  78C9              ; INTEL 808X AND THE MOTOROLA 680X.
0154  78C9              ;
0155  78C9  38          SUB10  SEC
0156  78CA  AA                 TAX
0157  78CB  E9 0A             SBC £10
0158  78CD  90 03             BCC CONVLS
0159  78CF  C8                 INY
0160  78D0  D0 F7             BNE SUB10
0161  78D2              ;
0162  78D2              ; CONVERT TO ASCII.  FOR SOME ODD REASON THE
0163  78D2              ; CBM ASSEMBLER DOES NOT LIKE EXPRESSIONS
0164  78D2              ; CONTAINING ASCII LITERALS
0165  78D2              ;
0166  78D2              ; E.G.  LDA £RVS+':
0167  78D2              ;
0168  78D2              ; NO ERRORS ARE FLAGGED BUT THEY ASSEMBLE INCORRECTLY
0169  78D2              ;
0170  78D2  8A          CONVLS TXA              ; LS BYTE
0171  78D3  09 B0             ORA £RVS+ZERO
0172  78D5  AE 08 78          LDX STOX
0173  78D8  9D 38 80          STA DISPOS,X
0174  78DB  CA                 DEX
0175  78DC  98                 TYA             ; MS BYTE
0176  78DD  09 B0             ORA £RVS+ZERO
0177  78DF  9D 38 80          STA DISPOS,X
0178  78E2  CA                 DEX
0179  78E3  30 05             BMI ENDDIS
0180  78E5  A9 BA             LDA £RVS+$3A    ; SEPARATOR (:)
0181  78E7  9D 38 80          STA DISPOS,X
0182  78EA  60          ENDDIS RTS
0183  78EB                   .END


ERRORS = 0000


SYMBOL TABLE

SYMBOL VALUE
 BINBCD   78C1    BIT16    787F    BIT8     7891    CHKTIM   7820
 CLEAR    7868    CLKOFF   7809    CLKON    7810    CLOCK    008D
 CNTDWN   7800    CONVLS   78D2    DISPOS   8038    DISRQD   78BE
 DIVIDE   7856    ENDDIS   78EA    GOINT    784F    HI       0001
 INTVEC   0090    NEXBIT   78BB    OPND     7801    REM      7804
 RSLT     7805    RVS      0080    SETVEC   7815    SHIFT    789B
 STDINT   E455    STOX     7808    SUB10    78C9    TOPSCR   00E0
 XFER     782D    ZERO     0030
END OF ASSEMBLY
```

ADD POWER TO YOUR COMMODORE COMPUTER

£49 + VAT

POWER produces a dramatic improvement in the ease of programming BASIC on Commodore computers. POWER is a programmer's utility package (in a 4K ROM) that contains a series of new commands and utilities which are added to the Screen Editor and the BASIC Interpreter. Designed for the CBM BASIC user, POWER contains special editing, programming, and software debugging tools not found in any other microcomputer BASIC. POWER is easy to use and is sold complete with a full operator's manual written by Jim Butterfield.

POWER's special keyboard 'instant action' features and additional commands make up for, and go beyond the limitations of CBM BASIC. The added features include auto line numbering, complete tracing functions, single stepping through programs, line renumbering, and definition of keys as BASIC keywords. POWER's 'WHY' command enhances debugging by listing the appropriate program line and highlighting where BASIC stopped executing. The cursor movement keys are enhanced by the addition of auto-repeat, and text search and replace functions are added to help ease program modification. POWER can even execute a sequential tape or disk file as though it were typed on the keyboard, allowing the user to merge two BASIC programs together. Cursor UP and Cursor DOWN produce **previous** and next lines of source code. COMPLETE BASIC PROGRAM listings in memory can be displayed on the screen and scrolled in either direction. You can even add your own commands to BASIC. Like our very successful Word Processing Programs (the "WordPro" series), POWER even includes convenient "stick-on" keycap labels which define new functions on the keyboard. POWER is a must for every dedicated CBM user.

Call us today, for the name of the Professional Software dealer nearest you.

Professional Software, Ltd.
153 High Street
Potters Bar
Hertfordshire EN6 5BB
Tel: (STD 0707) 42184 / (STD London 77)

Quotes Test

Dear Sir,

I am writing in response to an offer in the Editorial of the latest copy of CPUCN about the "Commodore Data Entry Environment" System. I have a small software house Grove Data Systems providing custom built programs for PETs and I feel that standardization is very important. I would be grateful to receive it.

On a different subject I was very impressed with the program listing routine given in the previous CPUCN.

I have added another few routines which I have put into it. I enclose the whole program which you might like to publish.

I will describe the extras

1) It prints the title and date at the top of the listing (the title may be different from the filename).

2) It pages, printing the page number and title at the bottom of each page. In addition it always finishes a program line before going onto another page.

3) All cursor controls, clear, home II etc are printed as CLS RVS SPC and so on making the listing much clearer.

The SPC can be a bit overwhelming but it makes it possible to put in the exact format of say printed headings.

4) The line numbers are right justified which looks neater.

5) The "in the left hand side margin is eliminated by changing CHR$ (34) to CHR$ (98) before it prints.

6) The problem with most listings is that anything in " " in the program which is in upper case will appear as graphics in the listing. There is a routine unclustered which prints text as it would appear on the screen. If the program switches mode, the listing of the text will change mode. This makes the listing very readable and I haven't seen it done elsewhere. The default value is graphics mode. I have included a small demonstration listing called "Quotes Test" which show it. In an ordinary listing lines 15 and 25 would appear identical.

I hope this is of interest, as you mention that you are short of Basic Programs to include. It could I am sure be polished up but it does work.

Yours faithfully, Phillip Deakin

```
PROGRAM NAME : QUOTES TEST    DATE  16-03-81

   10  POKE59468,14
   15  OPEN1,4,1,PRINT#1,"CCccAAaa"
   20  POKE59468,12
   25  OPEN1,4,1,PRINT#1,"—CC♠♠AA"
   30  END

                    QUOTES TEST      PAGE  1

PROGRAM NAME : PROGRAM LIST   DATE  16-03-81

10000  DIMOP$(75):SC$="59468
       ,1"
10010  PC=1:LC=0:LW=35:X=0:NQ=1:
       OPEN15,8,15:OPEN3,3:E=256:QU$
       =CHR$(34):GOTO10040
10020  INPUT#15,ER,B$,TR,SE:IFSE=0
       THENRETURN
10030  PRINTER,B$;TR;SE:GOTO10050
10040  GOSUB10150
10050  IFCH=4THEN10080
10060  PRINT"[CRD][CRD][CRD][CRD]
       [CRD][CRR][CRR][CRR][CRR]
       PRESS[SPC]ANY
       [SPC]KEY"
10070  GETA$:IFA$=""THEN10070
10080  POKE158,0:END
10090  FORI=0TO75:READOP$(I):NEXT:
10100  DATAEND,FOR,NEXT,DATA,INPUT#,
       INPUT,DIM,READ,LET,GOTO,RUN,I
       F,RESTORE
10110  DATA[SPC]GOSUB,RETURN,REM,STO
       P,ON,WAIT,LOAD,SAVE,VERIFY,DE
       F,POKE,PRINT#
10120  DATA[SPC]PRINT,CONT,LIST,CLR,
       CMD,SYS,OPEN,CLOSE,GET,NEW,TA
       B(,TO,FN,SPC(,THEN
10130  DATANOT,STEP,+,-,*,/,↑,AND,OR
       ,>,=,<,SGN,INT,ABS,USR,FRE,PO
       S,SQR,RND,LOG
10140  DATA[SPC]EXP,COS,SIN,TAN,ATN,
       PEEK,LEN,STR$,VAL,ASC,CHR$,LE
       FT$,RIGHT$,MID$,GO
10150  PRINT"[CLS]LIST":B
       =30:PRINT"[CRD][CRD]"
10160  CMD3,;:INPUT"[CRD][CRD][CRD]
       FILENAME[SPC]
       ?[SPC]";F$:F$=LEFT$(F$,15
       ):PRINT
10170  CMD3,;:INPUT"[CRD]DRI
       VE[SPC]NUMBER
       [SPC]?[SPC]";DN$:IFDN$
       <"0"ORDN$>"1"GOTO10170
10180  PRINT:OPEN2,8,0,DN$+":" F$
       ",PRG,READ":GOSUB10020
10190  CH=3:CMD3,;:INPUT"[CRD]SC
       REEN[SPC]OR[SPC]
       PRINTER[SPC]?
       [SPC]S[CRL]";A$:PRINT:
       IFA$="P"THENCH=4
10200  PRINT"[CLS]":OPEN4,CH

                    PROGRAM LIST      PAGE  1
```

```
10210  PRINT#4,CHR$(19)
10220  GOSUB10090
10230  IFCH=3GOTO10290
10240  INPUT"[CRD]WHAT[SPC]
       IS[SPC]THE[SPC]
       DATE[SPC]TODA
       Y[SPC][SPC][SPC][RVS]
       DDMMYY[RVF][CRL]
       [CRL][CRL][CRL][CRL][CRL]
       [CRL][CRL]";D$:X1$=LEFT$(
10250  X2$=MID$(D$,3,2):X3$=RIGHT$(D
       $,2):D$="[SPC][SPC][SPC]
       DATE[SPC][SPC]"+X
       1$+"-"+X2$+"-"+X3$:PRINT
10260  PRINT#4:PRINT#4
10270  INPUT"[CRD][CRD]PRO
       GRAM[SPC]NAME
       [SPC]";P$:PRINT#4,"PRO
       GRAM[SPC]NAME
       [SPC]:[SPC]";!(1)P
       $;!(129);D
10280  PRINT#4:PRINT#4:PRINT#4:LC=LC+6
10290  C=2:GOSUB10650
10300  LM=LN:Q=0:GOSUB10650:IFLN=0
       THENPRINT#4,R$:GOSUB10340:
       CLOSE2:GOTO10050
10310  IFLC<56THEN10370
10320  IFLI=1THEN10370
10330  GOSUB10340:PRINT#4:PRINT#4:LC
       =2:PC=PC+1:GOTO10370
10340  IFCH=3THEN10360
10350  FORZ=1TO62-LC:PRINT#4:NEXT:
       PRINT#4,SPC(18);P$;"[SPC][SPC]
       [SPC][SPC][SPC]PAG
       E[SPC]";PC:PRINT#4:
       PRINT#4:PRINT#4
10370  IFLEN(R$)>LEN(STR$(LM))+7
       THENPRINT#4,R$:LC=LC+1:LI=0
10380  IFLC>56ANDLI=0THENGOSUB10340:
       PRINT#4:PRINT#4:LC=2:PC=PC+1
10390  GOSUB10650:R$=RIGHT$("[SPC]
       [SPC][SPC][SPC][SPC][SPC]
       [SPC]"+STR$(LN),5)+"[SPC]
       [SPC]"
10400  GET#2,A$:IFST<>0THENPRINTCR$;
       EF$:CLOSE2:GOTO10050
10410  IFA$=""THENQ=0:GOTO10430
10420  GOTO10450
10430  IFAR=1THENAR=0
10440  GOTO10300
10450  IFA$=QU$THENQ=NOTQ
10460  IFCH=4THENLW=70
10470  IFAR=0ANDASC(A$)=>128GOTO1061
10480  IFAR=0ANDASC(A$)=34THENAR=1:
       GOTO10550
```

PROGRAM LIST PAGE 2

```
10490  IFAR=1ANDASC(A$)=34THENAR=0:S
       Z=0:GOTO10505
10500  IFAR=1ANDASC(A$)=58THENAR=0:S
       Z=0:GOTO10505
10505  IFSZ=0THEN10550
10510  IFSZ>0ANDMID$(SC$,SZ,1)=A$
       THENSZ=SZ+1
10530  IFSZ=8ANDASC(A$)=50THENSX=0:S
       Y=0:SZ=0:GOTO10550
10540  IFSZ=8ANDASC(A$)=52THENSX=145
       :SY=17:SZ=0
10550  GOSUB10730
10560  IFASC(A$)=98THEN10590
10570  IFAR=1ANDASC(A$)<128THENA$=
       CHR$(SY)+A$+CHR$(SX):GOTO1059
10580  IFAR=1ANDASC(A$)>128THENA$=
       CHR$(SY)+A$
10590  R$=R$+A$:IFLEN(R$)=>LWTHEN
       PRINT#4,R$:GOSUB10690:LC=LC+1
10600  GOTO10400
10610  IFQORASC(A$)>203GOTO10590
10620  A$=OP$(ASC(A$)-128):IFA$="P
       OKE"THEN[SPC]SZ=1
10630  IFLEN(R$+A$)=>LWTHENPRINT#4,R
       $:GOSUB10690:R$=R$+A$:LC=LC+1
       :LI=1:GOTO10400
10640  R$=R$+A$:GOTO10400
10650  GET#C,A$:LN=0:IFA$<>""THENLN
       =ASC(A$)
10660  GET#C,A$:IFA$<>""THENLN=LN+E
       *ASC(A$)
10670  GETA$:IFA$="C"GOTO10050
10690  R$="[SPC][SPC][SPC][SPC]
       [SPC][SPC][SPC]"
10700  IFNQ=0GOTO10720
10720  IFQTHENR$=CHR$(34)+LEFT$(R$,
       LEN(R$)-1)
10730  IFASC(A$)=19THENA$="[HM
       S]"
10740  IFASC(A$)=147THENA$="[CL
       S]"
10750  IFASC(A$)=17THENA$="[CR
       D]"
10760  IFASC(A$)=18THENA$="[RV
       S]"
10770  IFASC(A$)=20THENA$="[DE
       L]"
10780  IFASC(A$)=29THENA$="[CR
       R]"
10790  IFASC(A$)=145THENA$="[CR
       U]"
10800  IFASC(A$)=146THENA$="[RV
       F]"
10810  IFASC(A$)=157THENA$="[CR
       L]"
```

PROGRAM LIST PAGE 3

```
10820  IFASC(A$)=222THENA$="[PI
       [SPC]]"
10830  IFASC(A$)=148THENA$="[IN
       S]"
10840  IFASC(A$)=32THENA$="[SP
       C]"
10850  IFASC(A$)=34THENA$=CHR$(98):
       IFVC=1THENVC=0
10860  RETURN
```

PROGRAM LIST PAGE 4

Programming Tips

In the July issue of the newsletter we published a short machine code program for helping you recover from the disasters of typing NEW when really you knew you shouldn't have done. Well, thanks to Mark Humphrey, resident genius at Supersoft, we have a far simpler method. No, it does not involve reaching for the nearest bottle of whisky, but rather the following sequence of commands:-

BASIC 1 - POKE1026,4:SYS50224 and then RETURN
BASIC 2 - POKE1026,4:SYS50242 and then RETURN
BASIC 4 - POKE1026,4:SYS46262 and then RETURN

After this I would strongly recommend SAVEing (I know it's not spelt like that, but the keyword is SAVE after all) your program, as an attempt to add lines to that program will result in some very strange things happening. However, for those of you with Basic Aid, Toolkit, or anything that has a FIND command, we can get round this, providing that Basic Aid or whatever is up and running at the time. After doing the POKE and SYS commands, type in -
FIND/BANNANAS/
or any other word which is not likely to occur in the program, and hit RETURN. Voila! All is now safe.

Gentle Adventure

Once upon a time, deep, deep in the forest, lived a family of Dongles. They had lived there happily for generations, secure and peaceful. Father Dongle, a trader, used to go to the village to work, and there he would turn the key of his silicon office and with his office mate settle down for a good day's labour. Mother Dongle used to stay at home looking after the little Dongles, especially Graham Dongle, who for a twelve year old was difficult to monitor.

She would entertain them with all their favourite stories. Adventures with the wizzard of ozz were always the stories they wanted to hear, and she found it difficult to stop for them, even to kram some food down them.

Not so father Dongle, who was very skilled at cracking hidden messages, and who always went to the village inn at lunchtime for a pint of Old Netkit, a renowned local brew. He was so good at this skill that in the village he was known as the champion bar code reader. But he never used to stop there for long - soon he'd be back in his office beavering away.

In the evenings the whole family used to gather round the fire, and the children would be told stories warning them not to go into the forest at night. The story that always frightened them the most, and made sure they didn't stray far, was about the evil, corrupt family of Disks who lived in the heart of the forests where the trees were densest, and the ferns were pretty stupid as well.

At the weekends the whole family used to go to one of the prettiest spots in the area. There they would meet their near neighbours, the Mupets. Whilst the fathers would find that there was plenty of rom at the nearby inn (The Prep and Telex), the little Dongles and Mupets would happily play in the nearby river, trying to avoid being stung by the bees.

And so this happy scene continued for many weeks, until one day a new family came to live nearby - once given the key of their new home, the 'Data', the Mafia were ready to move in!

All the little Dongles and Mupets were told to stay away from the Mafia, and this they did. The corrupt Disks may have been a myth, but the Mafia were very, very real.

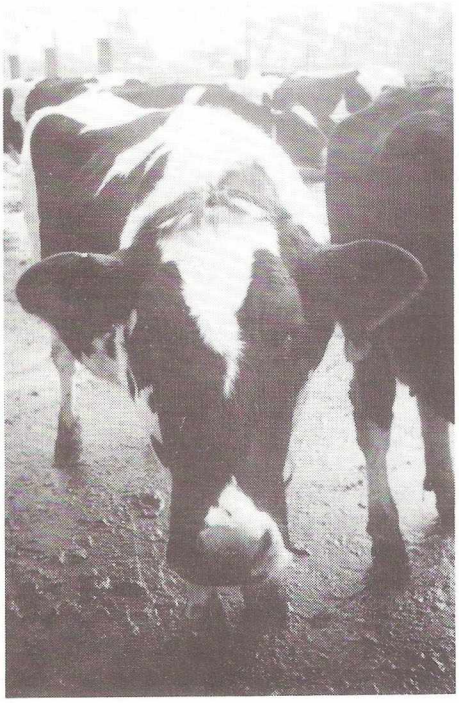
One day, at the height of the summer, Graham Dongle was wandering alone through the forest, minding his own business, when who should appear but C.B., the Mafia's evil Manager. Graham was frightened at first, but soon, under the wicked charm of C.B., he was talking happily away and the two of them began walking together. Getting ever nearer to the 'Data' as they went, it wasn't until they got inside that Graham realised he'd been kidnapped!

C.B., threatening to break Graham's legs if he tried to run away, sent a ransom note to the Dongles. Father Dongle read it, and the whole family went into mourning. The note read "We will return your son, if you send us 1,000,000 Turnkeys".

"But I only earn 20 a week" sighed father.

Mother was distraught. "If only Supercow were here...."

Meanwhile, in a field not too far away, a herd of cows was quietly grazing on the grass, munching contentedly in the sun. One of them, called Clark, was in a particularly good mood that day. No real reason, but life was treating Clark well lately. "It's a cow's life", he mused.



The farmer came rushing into the field, shouting to one of his hands "Its the Mafia! They've kidnapped young Graham Dongle!" "Oh No!" cried his other hand.

"Hmm", thought Clark, "sounds like a case for Supercow. But how can I get away without anyone noticing ?" For mild-mannered bovine Clark was really the astounding Supercow, able to leap tall barns in a single bound, more powerful than a combine harvester, faster than a speeding tractor. And then he thought "Easy. All I need is a telephone kiosk".

Apparently without concern, whistling a little tune as he went, he began strolling over to the edge of the field, where a rather large telephone kiosk was situated. Showing suprising agility for a cow his size he leapt into the kiosk, only to emerge seconds later as SUPERCOW, resplendent in cape and with a large letter 'S' emblazoned across his chest.

As he flew over the barn, a group of cows stared in amazement.
"Is it a wombat ?"
"Is it a flying VIC ?"
"No, it's Supercow!!!"

Supercow, flying faster than an express dairy, raced to the scene. C.B. looked in amazement as a cow with a cape and red underpants burst in through the door. He fired bullets, threatened to chop legs off at the knees, but to no avail. Mighty Supercow picked up young Graham Dongle and whisked him back to the safetly of his family.

"Thank you Supercow", said mother Dongle. "How can we ever repay you ?"

"No repayment ma'am. Simply remember what we're fighting for. Truth, justice and the arable way of life". And with that he was away, and Graham Dongle had learnt a lesson he would never forget. A Dongle should always stay at home.

NOTES ON 8032 & 12" 4032

Mark Humphrey

The 8032 and the new 12" screen 4032 both have a single chip video controller, this is new to the PET and is not found on any of the 9" screen machines. This chip completely controls the video display and has 18 registers which control various aspects of the display.

Firstly although the chip has 18 registers it only occupys 2 locations in memory, the first ($E880 Decimal 59520) is an address register and the second ($E881 Decimal 59521) is the location of the other registers. Each register is addressed by poking the number of that reigster into the address register, and then that register will ocupy address 59521. Unfortunately both the address register and the most of the 18 control registers are read only so it is worth remembering what you poked it with.

Below is a list of all the control registers and their function. To prevent ambiguity I shall explain a few of the terms I have used. The screen is the physical glass screen. The Frame is the area of screen between the upper and lower edge and left and right margins which actually displays the picture. The upper and lower edges are the very top and bottom of the picture. The left and right margins are the extreme left and right edges of the picture, it is however possible to have the left margin to the right margin. A line as an actual scanned line (8 of these make up one character block). A display line is a length of characters usually 8 lines high, from the left margin to the right margin. A column is each individual dot position (there are 8 columns in a character and 640 columns on an 80 character display line). A pixel is one line in height and one column in width.

REG No.	Function	Default Value 8032	4032
0	Horizontal total number of characters on line (Nht) including horizontal retrace. (true value= number +1) Divide by 2 for 8032	49	49?
1	Horizontal number of characters displayed (Nhd) Divide by 2 for 8032	40	40?
2	Distance (in Characters) from left margin to right of screen +1. Divide by 2 for 8032	41	20?
3	Sync width. Low order 4 bits are vertical sync width (in lines). High order 4 bits are horiztontal sync (in characters).	15	?
4	Number of display lines including retrace (Nvt).	32/40	?
5	Vertical position of frame (fine)	3/5	?
6	Number of display lines in frame (Nvd)	25	25?
7	Hieght of upper edge from bottom of screen (in display lines)	29/33	?
8	Intelace and Skew:-		
	bit 0 1=interlaced mode 0=noninterlaced mode	0	0?
	bit 1 if bit 0=1 then interlace and video mode	0	0?
	bit 2 not used	0	1?
	bit 3 not used	0	1?
	bit 4 1=scan from 32770 in memory	0	0?
	bit 5 1=scan from 32772 in memory	0	0?
	bit 6 cursor (not implemented on the PET)	0	0?
	bit 7 cursor (not implemented on the PET)	0	0?
9	Number of lines between top of one display line and top of next	9/7	9/11?
10	Cursor (not implemented on the PET)	0	0?
11	Cursor (not implemented on the PET)	0	0?

12	Control register :-		
	bit 0 add 256 to start address	0	0?
	bit 1 add 512 to start address	0	0?
	N.B. bit 0 & 1 add 512 and 1024 on 8032		
	bit 2 invert fly back	0	0?
	bit 3 invert video signal	1	0?
	bit 4 use top half of 4k character generater	0	0?
	bit 5 (not implemented on the PET)	0	0?
	bit 6 (not implemented on the PET)	0	0?
	bit 7 not used	0	0?
13	Value +32768 is address of first character character (multiply by 2 for 8032)	0	0?
14	Cursor location HI (not implemented on the PET)	0	0
15	Cursor location LO (not implemented on the PET)	0	0
16	Light pen position HI (read only)	0	0
17	Light pen position LO (read only)	0	0

Any register which has two figures is changed when the screen is expanded or compressed the first number is the power up value.

Try this program listed below :

```
10 POKE59520,2:REM SELECT REGISTER NO. 2
20 FORI=0TO40:POKE59521,I:NEXT
30 FORI=40TO0STEP-1:POKE59521,I:NEXT
40 GOTO20
```

One useful hint should you stop the program, and are unable to see what you are typing, if you hold down the two shift keys and tap the quotes key this will restore the display to the normal uppercase/graphics diplay.

This routine could be used as an error warning

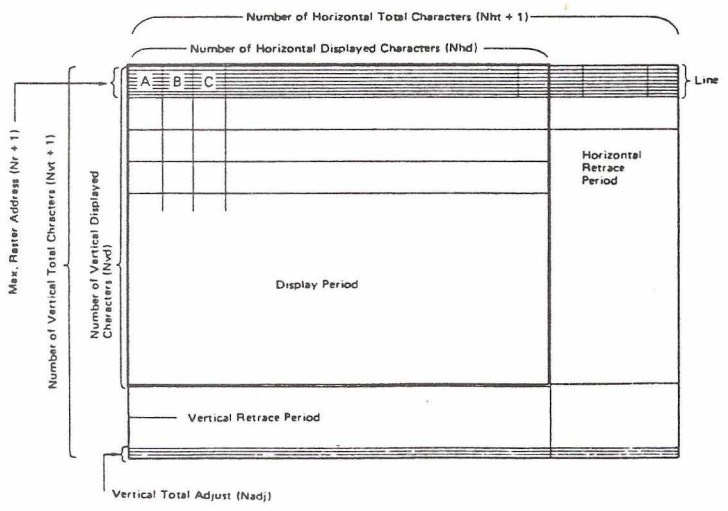
```
10 POKE59520,12
20 FORI=1TO20:POKE59521,0:POKE59521,16:NEXT
```

The chip has the facility to have a cursor (either blinking or not) but this is not connected on the PET. There is also the facility to have a 4K character generator the top 2K can be accessed using :-

```
POKE59520,12:POKE59521,24
```

N.B Some of the registers if poked certain values can cause the picture to colapse, should this happen SWITCH OFF THE PET IMEDIATELY.

CRT Screen Format



Mixing SYS and USR functions on the PET

from Alan Price - Liverpool Polytechnic Department of Mathematics

This article describes the facilities for pasing data to and from machine-code routines called by the BASIC SYS and USR commands, and shows how multiple SYS and USR functions may co-exist. A method of changing the number and type of parameters which USR can accept is described. Specific details of the PET monitor and BASIC interpreter refer to the PET 30XX (BASIC 2).

SYS and USR facilities in BASIC allow the use of machine-code routines to augment the facilities of BASIC, either because they do not exist in BASIC, or because of speed requirements. SYS is a command, followed by a store address of the machine code to be obeyed. This must make its own arrangements to communicate with BASIC, e.g. by using routines in the BASIC interpreter to read the remainder of the SYS command. See, for example, pages 16 and 17 of CPUCN volume 3 issue 3. As I showed in that article, the interpretation of the SYS itself can be quite time consuming, being most efficient if the SYS address is 0. This would appear to make the co-existence of multiple SYS and USR functions impossible.

USR is treated as a function. As described in the BASIC manual, it can have one parameter, which must be a numeric value (constant or expression) and can return one value, which can be numeric or string. Since it is entered via store locations 1 and 2, it would appear that there can only be one USR function at a time.

Multiple SYS commands can be implemented by arranging all SYS commands to go to location 0, which contains a jump to a routine to read the next character from the program, and use it to select one of a number of routines from a Jump Table (see the article by Mike Gross-Niklaus in CPUCN Vol 3 issue 3). If the jump table is set up to correspond to the letters of the alphabet, SYS.A would be sent to the routine with the first address in the table, SYS.B the second and so on. The reason for the full-stop is that BASIC must read a numeric value after SYS, which we want to be zero: and . followed by anything other than a digit or E is the shortest and quickest way of achieving this. The letters A,B and so on are not BASIC variables, and do not take up space in the variable store. This method of entering SYS routines is fast (about 1.02 milli seconds) and the BASIC programmer does not have to know the actual store addresses of the machine-code routines.

This would seem to scupper the use of USR functions. The solution to that is simple, if draconian: to insert a "wedge" into BASIC to trap USR functions before BASIC evaluates then become very much like SYS code, and read as many parameters of whatever type they like. The differences are that USR code should check that the last parameter is followed by a closing bracket, and must leave a value (numeric or string) lying about for BASIC to pick up on exit from the USR function, in the usual way.

The machine-code routine shown provides multiple SYS and USR facilities. It is loaded into the second cassette buffer, but could be modified to sit in high memory if preferred.

Location $827 contains a count of the number of addresses in the jump table, which is stored in $828 onwards. The program starts in $880, leaving room for 26 addresses, corresponding to the letters A-Z. The Monitor routine CHRGET, at store location $112($70) is altered to jump to $880, which tests whether a USR

```
10 REM"DYNARRAY" 22/06/81
20 PRINT"DYNAMICALLY EXTENDABLE ARRAY FACILITY
30 PRINT"**************************************
40 PRINT
50 PRINT"A ONE-DIMENSIONAL ARRAY OF ANY TYPE MAY
60 PRINT"BE EXTENDED AT ANY TIME BY A CALL OF
70 PRINT"THIS ROUTINE, E.G.
80 PRINT"SYS826,A(5)
90 PRINT"THIS WILL EXTEND THE ARRAY A BY 5
100 PRINT"ELEMENTS, MOVING ANY LATER ARRAYS UP
110 PRINT"THE STORE TO MAKE ROOM. THE NUMBER OF
120 PRINT"ELEMENTS ADDED MUST NOT EXCEED THE
130 PRINT"CURRENT SIZE OF THE ARRAY, IF THE
140 PRINT"ARRAY DID NOT EXIST, IT WILL BE
150 PRINT"AUTO-DIMENSIONED WITH A SIZE OF 10
160 PRINT"BEFOR EXTENSION (BY UP TO 10).
170 PRINT"ERRORS:-
180 PRINT"BAD SUBSCRIPT ERROR EXTENSION TOO BIG
190 PRINT"SYNTAX ERROR NO COMMA AFTER 826
200 PRINT"    -**-     MORE THAN ONE DIMENSION
210 PRINT"OUT OF MEMORY ERROR NOT ENOUGH SPACE
220 PRINT"NOTE:-THE NEW ELEMENTS ARE NOT ZEROED
800 FORI=826TO936:READJ:POKEI,J,:NEXTI
826 DATA 32, 248, 205, 32, 109, 207, 196, 44, 176, 7, 197, 45, 176, 3
840 DATA 76, 3, 206, 229, 85, 133, 110, 152, 229, 86, 133, 111, 160, 4
854 DATA 177, 92, 201, 1, 208, 236, 160, 2, 24, 165, 92, 133, 37, 113
868 DATA 92, 170, 200, 165, 93, 133, 38, 113, 92, 133, 93, 134, 92, 165
882 DATA 47, 166, 46, 133, 88, 134, 87, 101, 111, 168, 138, 101, 110, 144
896 DATA 1, 200, 133, 85, 132, 86, 32, 216, 194, 160, 6, 24, 177, 37
910 DATA 101, 97, 145, 37, 136, 177, 37, 101, 98, 145, 37, 160, 2, 177
924 DATA 37, 101, 110, 145, 37, 200, 177, 37, 101, 111, 145, 37, 96
READY.
```

```
10 REM INSTAL CODE FROM DATA TO HIMEM,ADJUST HIMEM
20 FORI=1TO2
30 READZ:W=PEEK(52)+256*PEEK(53)-Z
40 FORY=WTOW+Z-1:READX:POKEY,X:NEXT
50 Z=INT(W/256):W=W-256*Z
60 POKE48,W:POKE50,W:POKE52,W:POKE828+2*PEEK(827),W
70 POKE49,Z:POKE51,Z:POKE53,Z:POKE829+2*PEEK(827),Z
80 POKE827,PEEK(827)+1:REM INCREASE ROUTINES COUNT
90 NEXTI
100 DATA34:REM 16-BIT PEEK- USRA(<ADDRESS>)
110 DATA165,18,72,165,17,72,32,139,204,32,210,214
120 DATA32,242,205,160,1,177,17,170,136,177,17
130 DATA168,104,133,17,104,133,18,138,76,109,210
140 DATA38:REM 16-BIT POKE- SYS.B<ADDRESS>,<VALUE>
150 DATA32,139,204,32,210,214,165,18,72,165,17,72
160 DATA32,248,205,32,139,204,32,210,214,170,104
170 DATA133,17,104,133,18,152,160,0,145,17,200
180 DATA138,145,17,96
READY.
```

function is being processed by looking at the stack; if not, the instructions which were in CHRGET at $112-$117 are obeyed at $906-$911. If USR, the character immediately after USR is saved in X, and the next character checked to be "(".

Location $1-2 is set up so that SYS. will jump to $932. The letter following SYS. letter and USR letter (is executed. ($939-$967). This checks that the saved character is a letter A-, and corresponds to an entry in the jump table, and performs an indirect jump to it.

The machine code initially stored in $826-$864 can be called by SYS 826. This plants an RTS instruction at $826 to prevent re-initialisation, and sets the initial value of the jump table count in $1-2, and alters CHRGET to jump to the USR code. All the user has to do is fill up the jump table, and alter the count in $827 to match.

Note that SYS.E should not be used, as E is not a number terminator: it may be reserved for a USR function, i.e. USRE (---).

Having thrown away the BASIC interpreter USR action, what will you have to do to replace it? The simplest way to tell is to look at what you are missing, i.e what BASIC does to USR. In the general expression evaluation routine, if a symbol with internal 'token' of value $B4 to $CA appears, this represents a function. The $80 bit is removed and the result saved, shifted left one place (i.e. doubled), ready to look up a jump table stored in $C046 onward. The entry in the jump table corresponding to USR (token $B7) is zero, causing a jump to location zero, where a jump to the actual location of the USR function must be stored, e.g. figure 1 The machine code described in para. 6 effectively takes USR functions out from $CE8F, so you are losing the sequence from $CEB3, onward e.g. evaluation of parameters, entry to function, and testing the type of the result. Useful routines are as in fig. 2 If you make your own strings or floating-point numbers, location $7 must be set to indicate the type, e.g. 0 for numbers -1 for strings. See the examples on page 17 of CPUCN Volume 3 issue 3. The evaluate and convert routines described above set $7 correctly. RTS (Return) at the end of your code will return to the address that the code at $CC8E does, i.e. bypassing the numeric test. Note that an ordinary USR function can bypass the test by executing PLA PLA before its final RTS.

As an illustration, the second program installs two pieces of machine code in high memory, and plants pointers to them in the jump table. The first, called by USRA (address), is 16-bit PEEK: the second; SYS.B address, value is 16-bit POKE.

```
10 REM SET UP MULTIPLE SYS/USR FUNCTIONS.
11 REM TRANSFERS MULUSR TO BUFFER ##2,THEN
12 REM INITIALISES SYS/USR AND CLEARS THE
13 REM JUMP TABLE COUNT.
100 FORI=826TO924:READJ:POKEI,J:NEXT
101 REM LOAD MULUSR TO BUFFER ##2.
110 SYS826:STOP
826 DATA 173, 96, 3, 141, 58, 3, 169, 0, 141, 59
836 DATA 3, 169, 164, 133, 1, 169, 3, 133, 2, 160
846 DATA 5, 185, 122, 0, 170, 185, 138, 3, 153, 112
856 DATA 0, 138, 153, 138, 3, 136, 16, 239, 96, 255
866 DATA 0, 255, 0, 255, 0, 255, 0, 255, 0, 255
876 DATA 0, 255, 0, 255, 138, 72, 186, 189, 2, 1
886 DATA 201, 142, 208, 14, 189, 3, 1, 201, 206, 208
896 DATA 7, 189, 4, 1, 201, 110, 240, 11, 104, 170
906 DATA 76, 112, 3, 234, 234, 234, 76, 118, 0, 104
916 DATA 104, 104, 104, 32, 138, 3, 170, 32, 138, 3
926 DATA 32, 245, 205, 76, 171, 3, 32, 118, 0, 170
936 DATA 32, 138, 3, 138, 56, 233, 65, 176, 3, 76
946 DATA 3, 206, 205, 59, 3, 176, 248, 10, 168, 185
956 DATA 60, 3, 133, 82, 185, 61, 3, 133, 83, 108
966 DATA 82, 0
READY.
```

figure 1.

```
$CE89   ASL A           ; 2* token less $80
        PHA             ; save jump table index
        TAX
        JSR $70         ; get character following function token
$CE8F   CPX #$8F        ; test for LEFT$, RIGHT$, MID$
        BCC $CEB3
        JSR $CDF5       ; string functions - check for "("
        JSR $CC9F       ; evaluate expression - first parameter
        JSR $CDF8       ; check for comma
        JSR $CC90       ; check parameter is a string
        ---------
$CEB3   JSR $CDEC       ; evaluate (expression)
        PLA
        TAY
        LDA $BFDE,Y     ; do jump table lookup
        STA $52         ;
        LDA $BFDF,Y     ; ($51 contains "JMP" code)
        STA $53
        JSR $51         ; subroutine jump to function
        JMP $CC8E       ; on return from function, test type
        ---------
$CC8E                   ; in $7 is numeric, RETURN.
```

figure 2.

```
JSR $CC9F    Evaluate expression (numeric or string)
JSR $CC90    Check expression is string
JSR $CC8E    Check expression is numeric
JSR $D6D2    Convert numeric value to integer
JSR $D26D    Convert integer to numeric (floating point)
JSR $CDF8    Check for comma
JSR $CDF2    Check for closing bracket
```

Peripheral Spot

Another collection of disk-based programs for you, and a very overseas based collection at that. The majority of the programs come from Peter Gabor, of Gaborton in Israel, and include programs to inspect sequential files, a revamped version of an earlier program which is an editor for relative files, and finally a program that allows a repeat function to co-exist with DOS support. This latter is most useful, as it goes into great detail on how the programs were written, and should thus serve as a useful guide to those of you attempting to do a similar thing.

INSPECT SEQUENTIAL FILES - *Peter Gabor*

```
10 IFPKTHEN210
20 PRINT"[CLR]    ********************************"
30 PRINT"     *                              *"
40 PRINT"     *  INSPECT SEQUENTIAL FILES    *"
50 PRINT"     *                              *"
60 PRINT"     ********************************[2CD]"
70 IFPEEK(721)=169ANDPEEK(722)=85THENSYS720:REM     DISABLE REPEAT
80 PK=PEEK(59468):POKE59468,14:                     SET TO LOWER CASE
90 PRINT"A MACHINE CODE PROGRAM FOR INSPECTION"
100 PRINT"OF SEQUENTIONAL FILES IS NOW BEING"
110 PRINT"LOADED.[CD]"
120 PRINT"THIS PROGRAM WILL CO-EXIST WITH ANY "
130 PRINT"OTHER PROGRAM TO BE LOADED."
140 PRINT"[CD]TO CALL THE PROGRAM IN THE FUTURE, SET"
150 PRINT"DISPLAY TO LOWER CASE AND ENTER [RVS]SYS32177"
160 PRINT"[CD]TO STOP LISTING, PRESS [RVS]STOP[OFF] KEY."
170 PRINT"TO CONTINUE, PRESS ANY OTHER KEY."
180 PRINT"TO SLOW DOWN DISPLAY, PRESS [RVS]OFF/RVS[OFF]."
190 POKE52,176:POKE53,125:                          REM     LOWER MEMORY (32K PETS!!)
200 DLOAD"SE0/SYS32177"
210 PRINT"[CD]_____"
220 PRINT"[RVS]DO YOU WISH TO INSPECT A FILE NOW? Y/N"
230 GETA$:IFA$<>"Y"ANDA$<>"N"THEN230
240 IFA$="Y"THENPRINT"[CD]ENTER NAME OF FILE!":SYS32177
250 POKE59468,PK:PRINT"[CLR]":NEW
READY.
```

MACHINE CODE LISTING

-Peter Gabor

```
?
.
.:  7DB0 AA A9 0E 8D 4C E8 A9 3E
.:  7DB8 8D 62 E8 20 E2 B4 A9 00
.:  7DC0 85 DA A9 02 85 DB A9 02
.:  7DC8 85 D2 A9 08 85 D4 A9 02
.:  7DD0 85 D3 A2 00 BD 00 02 F0
.:  7DD8 07 E8 20 35 F3 4C D4 7D
.:  7DE0 86 D1 20 63 F5 A2 02 20
.:  7DE8 AF F7 A2 00 A0 00 8E 00
.:  7DF0 10 20 35 F3 F0 03 4C 03
.:  7DF8 7E 20 E4 FF F0 FB A2 02
.:  7E00 20 AF F7 A9 00 20 C0 F1
.:  7E08 A4 96 D0 08 AE 00 10 A0
.:  7E10 00 4C 23 7E A2 00 A0 00
.:  7E18 A5 D2 20 E2 F2 20 CC FF
.:  7E20 20 FF B3 A8 C0 0D D0 08
.:  7E28 20 67 E3 A2 00 4C EE 7D
.:  7E30 98 9D C0 83 E8 E0 28 90
.:  7E38 B5 4C 28 7E AA AA AA AA
.
.
READY.
```

FIXIT-EDITOR FOR RELATIVE FILES

- Peter Gabor

```
10 REM ********************************
20 REM *                              *
30 REM *          F I X I T           *
40 REM *                              *
50 REM *  EDITOR FOR RELATIVE FILES   *
60 REM *             1/6/81           *
70 REM *                              *
80 REM ********************************
90 REM
100 Z=1/254
200 G$(0)="  [RVS]┌─┬─┬─┬─┬─┬─┬─┬─┬─┬─┐
210 G$(1)="  [RVS]│  │0│1│2│3│4│5│6│7│8│9│
220 G$(2)="  [RVS]├──┼─┼─┼─┼─┼─┼─┼─┼─┼─┤
230 G$(3)="  [RVS]│  [OFF]  │ │ │ │ │ │ │ │ │ │
250 GM$="  [RVS]├──[OFF]──┼─┼─┼─┼─┼─┼─┼─┼─┼─┤
260 GB$="  [RVS]└──[OFF]──┴─┴─┴─┴─┴─┴─┴─┴─┴─┘
300 AC$="[CR]":D$="[4CD]":SP$="    ":AD$="[3CD]"
310 FORK=1TO5:SP$=SP$+SP$:AD$=AD$+D$:AC$=AC$+AC$:NEXT
320 C7$=LEFT$(AC$,7):SP$=SP$+LEFT$(SP$,100)
330 PP$="_____":TM$=CHR$(13)
450 GOSUB7000
500 INPUT"[CLR]INPUT FILENAME   [3CL]";A$
510 IFA$=" "THENPRINT"[CLR]";:END:          REM     JUST HIT RETURN
520 IFA$<>"*"THENNF$=A$                             TO EXIT
570 INPUT"[CD]LENGTH OF RECORD";RL
580 DOPEN#1,(NF$),L(RL),D0
590 IFRL>90THEN2000
600 FB=1:LB=RL
610 GOSUB5000:O$="[CD]-":                   REM     PRINT GRID
620 GOSUB950:INPUT"RECORD #   [3CL]";R$
630 GOSUB1000:IFR$=" "THENDCLOSE:GOTO500
640 REC=VAL(R$)
650 RECORD#1,(REC):GOSUB950
660 GOSUB5400:                              REM     READ RECORD
680 PRINT"RECORD # "REC;
690 GOSUB5200:POKE158,0:                    REM     WRITE ON SCREEN
700 GETA$:IFA$=""THEN700:                   REM     EDIT
710 IFA$="[INST]"THEN4000
720 IFA$=CHR$(20)THEN4050
730 IFA$="[CL]"THEN4100
740 IFA$="[CR]"THEN4150
750 IFA$="[CU]"THEN4200
760 IFA$="[CD]"THEN4250
770 IFA$="@"THENPRINTO$:GOTO620
780 IFA$="↑"THENREC=REC+1:PRINTO$:GOTO650
790 IFA$=CHR$(34)THENA$="*":GOTO4300
800 IFA$=CHR$(141)THEN4800
810 IFA$=CHR$(13)THENA$="←":GOTO4300
850 IFA$="[HOME]"THEN590
860 IFA$="[CLR]"THENDCLOSE:PRINT"[HOME]";:END
890 GOTO4300
930 REM
935 REM ****************************
940 REM * CURSOR TO LINE # 24      *
945 REM ****************************
948 REM
950 PRINT"[CLR]"AD$LEFT$(SP$,30)"[CLR]"AD$;:RETURN
955 REM
985 REM ****************************
990 REM *    CHECK ERRORS          *
995 REM ****************************
998 REM
1000 IFDS<20THENRETURN
1010 PRINT"[RVS]"DS$:DCLOSE:END
1970 REM
1975 REM ****************************
1980 REM * SET RANGE FOR INSPECTION *
1985 REM ****************************
1988 REM
2000 INPUT"[HOME]FIRST BYTE TO BE VIEWED   [3CL]";A$
2010 IFA$=" "THENDCLOSE:GOTO500
2020 FB=VAL(A$)
2030 IFRL-FB<81THENLB=RL:GOTO610
2040 INPUT"NR OF BYTES TO BE VIEWED";A$
2050 A=VAL(A$):IFA>80THENPRINT"RANGE TOO LARGE!":GOTO2040
2060 IFFB+A>RLTHENLB=RL:GOTO610
2070 LB=FB+A-1:GOTO610
3980 REM
3985 REM ****************************
3990 REM *         INSERT           *
3995 REM ****************************
3998 REM
4000 B2$=LEFT$(" "+B2$,LEN(B2$))
4010 GOSUB5200:GOTO700
4030 REM
4035 REM ****************************
4040 REM *         DELETE           *
4045 REM ****************************
4048 REM
4050 B2$=MID$(B2$,2)+" "
4060 GOSUB5200:GOTO700
4080 REM
4085 REM ****************************
4090 REM *        CRSR LEFT         *
4095 REM ****************************
4098 REM
4100 IFLEN(B1$)=0ORBN=FBTHEN700:        REM     CURSOR IN FIRST POS
4110 B=1:GOSUB4700
4120 GOTO700
4130 REM
4135 REM ****************************
4140 REM *        CRSR RIGHT        *
4145 REM ****************************
4148 REM
4150 IFLEN(B2$)=0ORBN=LBTHEN700:        REM     CURSOR IN LAST POS
4160 B=1:GOSUB4750
4170 GOTO700
4180 REM
4185 REM ****************************
4190 REM *        CURSOR UP         *
4195 REM ****************************
4198 REM
4200 IFLEN(B1$)<10ORBN<FB+10THEN700:    REM     NO HIGHER LINE AVAILABLE
4210 B=10:GOSUB4700
4220 GOTO700
4230 REM
4235 REM ****************************
4240 REM *        CRSR DOWN         *
4245 REM ****************************
4248 REM
4250 IFLEN(B2$)<10ORBN>LB-10THEN700:    REM     NO LOWER LINE AVAILABE
4260 B=10:GOSUB4750
4270 GOTO700
4280 REM
4285 REM ****************************
4290 REM * PRINT A$ & MOVE MARKER   *
4295 REM ****************************
4298 REM
4300 PRINTA$"[CL]";:B2$=A$+MID$(B2$,2)
4310 GOTO4150
4680 REM
4685 REM ****************************
4690 REM *  SBR FOR CRSR LEFT       *
4695 REM ****************************
4698 REM
4700 PRINTO$:BN=BN-B:GOSUB5500
```

```
4710 B2$=RIGHT$(B1$,B)+B2$
4720 B1$=LEFT$(B1$,LEN(B1$)-B)
4730 RETURN
4735 REM
4740 REM ****************************
4745 REM *  SBR FOR CRSR RIGHT      *
4748 REM ****************************
4749 REM
4750 PRINTO$:BN=BN+B:GOSUB5500
4760 B1$=B1$+LEFT$(B2$,B)
4770 B2$=MID$(B2$,1+B)
4780 RETURN
4785 REM
4790 REM ******************************
4795 REM * REPLACE RECORD (ENTER)     *
4798 REM ******************************
4799 REM
4800 PRINTO$:GOSUB5600
4810 REC=REC+1:GOTO650
4980 REM
4985 REM ****************************
4990 REM *  PRINT GRID ON SCREEN    *
4995 REM ****************************
4998 REM
5000 PRINT"[HOME]"
5010 FORK=0TO3
5020 PRINTG$(K):NEXT
5030 FORK=INT(FB/10)TOLB/10
5040 N$=RIGHT$(STR$(K),2)
5050 GN$="  [RVS]|"+N$+"|[OFF]  | | | | | | | | | |
5060 PRINTGN$:PRINTGM$:NEXT
5070 PRINT"[CU]"GB$:RETURN
5180 REM
5185 REM ****************************
5190 REM *    PUT B2$ ON SCREEN      *
5195 REM ****************************
5198 REM
5200 BN=LEN(B1$)+1:GOSUB5500:          REM      POSITION CURSOR
5210 PRINT"[CD]-[CU,CL]";:FORK=1TOLB-BN+1
5220   BY$=MID$(B2$,K,1)
5230 PRINTBY$"[CR]";:PS=PS+1
5240   IFPS=10THENPS=0:PRINTTM$TM$C7$;:REM      NEW LINE
5250   NEXT
5260 GOSUB5500
5270 RETURN
5375 REM
5380 REM ****************************
5385 REM * READ RECORD INTO A1$      *
5390 REM * DIVIDE INTO B1$ & B2$     *
5395 REM ****************************
5398 REM
5400 GOSUB1000:A1$="":FORK=1TORL
5410   GET#1,B$
5415   IFLEN(B$)=0 THEN B$=" "
5420   IFB$=CHR$(13)THENB$="←"
5430   IFB$=CHR$(34)THENB$="✱"
5440   A1$=A1$+B$:IFST=64THENK=RL
5450 NEXT:B1$="":A=LEN(A1$)
5455 IFA<RL-1THENA1$=A1$+LEFT$(SP$,RL-A)
5460 IFFB>1THENB1$=LEFT$(A1$,FB-1)
5470 B2$=(MID$(A1$,FB))
5475 RETURN
5479 REM
5480 REM ****************************
5485 REM * POSITION CURSOR TO BN &   *
5490 REM * PRINT "⊥" ON GRID         *
5495 REM ****************************
5498 REM
5500 A=INT(BN/10+Z):R=A-INT(FB/10+Z):  REM      ROW NUMBER
5510 PS=INT((BN/10-A)*10+Z):           REM      COL NUMBER (POSITION)
5520 R=6+2*R:C=7+2*PS
5530 PRINT"[CLR]"LEFT$(AD$,R)LEFT$(AC$,C);
5540 PRINT"⊥[CL,CU]";
5550 RETURN
5575 REM
5580 REM ****************************
5585 REM * READ B1$&B2$ INTO RECORD  *
5590 REM ****************************
5595 REM
5600 RECORD#1,(REC):GOSUB1000:B$=""
5605 A1$=B1$+B2$
5610   FORK=1TOLB
5620   A$=LEFT$(A1$,1):A1$=MID$(A1$,2)
5630   IFA$="←"THENA$=CHR$(13)
5640   IFA$="✱"THENA$=CHR$(34)
5650   B$=B$+A$
5660   NEXT
5670 PRINT#1,B$;:GOSUB1000
5680 RETURN
6975 REM
6980 REM ****************************
6985 REM *       INSTRUCTIONS       *
6990 REM ****************************
6995 REM
7000 POKE59468,14:PRINT"[HOME]"TAB(12)LEFT$(PP$,9)TM$TAB(12)"[RVS]* FIXIT *"
7010 PRINTTM$"THIS IS AN EDITOR FOR RELATIVE FILES.
7020 PRINT"FILES HAVING MORE THAN 90 BYTES CAN BE
7030 PRINT"EDITED IN SECTIONS OF UP TO 80 BYTES."TM$
7050 PRINT"THE BYTE BEING EDITED IS MARKED BY '⊥'
7060 PRINT"AND THIS POSITION CAN BE MOVED BY THE
7070 PRINT"REGULAR CURSOR CONTROL CHARACTERS."TM$
7080 PRINT"FOLLOWING COMMANDS ARE ALSO OPERATIVE:-"
7090 PRINT"  @     -   CHANGE RECORD #
7100 PRINT"  ↑     -   GO TO NEXT RECORD, WITHOUT
7110 PRINT"           ENTERING SCREEN
7120 PRINT" 'HOME'-   CHANGE BYTE RANGE
7130 PRINT" 'CLR' -   EXIT FROM EDITOR"TM$
7140 PRINT"TO ENTER RECORD, HIT SHIFTED CARR.RET.
7150 PRINT"TO CHANGE FILENAME, HIT '@' AND 'CR'"TM$
7160 PRINT"CARRIAGE RETURN IS DISPLAYED BY THE
7170 PRINT"EDITOR AS '←', AND QUOTES AS A HEART.
7180 PRINT"[CD,6CR,RVS]HIT ANY KEY TO CONTINUE![CU]";:
7190 POKE158,0:WAIT158,1:POKE59468,12
7200 POKE158,0:PRINT"[HOME]":RETURN
READY.
```

DOS - REP - MIX

One of the most useful Utility Programs for the Disk Drive is the "Universal Wedge" also known as "Dos Support". It makes all disk handling and program loading very easy.

Another extremely useful program is "Repeat" (The Pet Revealed, page 132). It is absolutely essential for fast editing.

Since both programs are in constant use, I decided to make a composite for convenience. Naturally, a few problems had to be resolved:-

1) The second cassette buffer should remain free for all those little routines, that appear in the "Newsletter". Therefore the routine had to be relocated to sit in the first cassette buffer.
2) A command had to be provided for turning the repeat function off. The Pet runs faster without Repeat, and with the "Repeat" activated some commands' such as "DLOAD" do not work properly.
3) Having a notoriously bad memory, I wanted to display the proper 'SYS' command(s) every time the program is loaded.

I came up with the (BASIC4 Version) shown in figure 1.

figure 1.

```
.:  0278 20 ff ff 78 a9 8a 85 90
.:  0280 a9 02 85 91 a9 01 85 02
.:  0288 58 60 a5 97 ea ea ea ea
.:  0290 ea ea ea ea ea ea ea ea
.:  0298 ea ea ea ea ea ea ea ea
.:  02a0 ea ea ea ea ea c5 00 f0
.:  02a8 09 85 00 a9 10 85 01 4c
.:  02b0 55 e4 c9 ff f0 f9 a5 01
.:  02b8 f0 04 c6 01 d0 f1 c6 02
.:  02c0 d0 ed a9 04 85 02 a9 00
.:  02c8 85 97 a9 02 85 a8 d0 df
.:  02d0 78 a9 55 85 90 a9 e4 85
.:  02d8 91 58 60 ff ff ff ff ff
.
ready.
```

The "Repeat" function is activiated by 'SYS635' and deactivated by 'SYS720'.

The train of "ea"s' (NoOp; $208C to $02A5) is for those, who like to fool around with the interrupt routine.

It is easy, for example, to change the display mode without having to POKE something or other (I forgot the exact numbers). Just put

C9 07 D0 05 A2 0C 8E 4C E8 C9 0E
D0 03 8D 4C E8

into locations $028C to $029B and the square brackets (,) will take care of changing Graphics to Lower Case and visa versa.

Another possibility is to use the 'STOP' key to deactivate Repeat instead of SYS720 . Write following sequence into locations $029C to $02A2:-

C9 04 D0 05 20 D0 02

The completed program now looks like figure 2.

Now for the actual merging of "DOS" with "Repeat":-

If you have the "Universal Wedge" Program, that works for both BASIC2 & 4, then load the program, but DO NOT RUN IT. Enter the Repeat program with TIM (type SYS4, .M 0278 02D8, etc). Now exit from the Monitor and list your program. It should look like figure 3.

The Machine Languages Loader for the wedge program is located from $0500 to $08AD. Editing the Basic

figure 2.

```
.:  0278 ff ff ff 78 a9 8a 85 90
.:  0280 a9 02 85 91 a9 01 85 02
.:  0288 58 60 a5 97 c9 07 d0 05
.:  0290 a2 0c 8e 4c e8 c9 0e d0
.:  0298 03 8d 4c e8 c9 04 d0 05
.:  02a0 20 d0 02 ea ea c5 00 f0
.:  02a8 09 85 00 a9 10 85 01 4c
.:  02b0 55 e4 c9 ff f0 f9 a5 01
.:  02b8 f0 04 c6 01 d0 f1 c6 02
.:  02c0 d0 ed a9 04 85 02 a9 00
.:  02c8 85 97 a9 02 85 a8 d0 df
.:  02d0 78 a9 55 85 90 a9 e4 85
.:  02d8 91 58 60 ff ff ff ff ff
.
ready.                    (BASIC4 version)
```

Continued on next page

Continued from previous page

program will displace the lower upwards. To relocate it into its original position, enter following commands in direct mode:

FOR K=1 TO 120: IF PEEK (1280+K) < >234 THEN NEXT (return)
FOR J=1280 TO 2225: POKE J,PEEK (J+K): NEXT (return)
Now enter the monitor (SYS 4) and save the program with the commands:
.S "0: DOS-REP-MIX ",08,027A,08AF
Exit from the Monitor, type 'RUN' and you are in business!

For those, who have BASIC2 with the old version of DOS (displaying all the instructions on the screeen), I should suggest following procedure:

1. Enter the 'Repeat' program, taking care to change following bytes:

Location	Contents	Change to
$02B0	#$55	#$2E
$02B1	#$E4	#$E6
$02D2	#$55	#$2E
$02D6	#$	#$E6

2. If you want to keep the original display of the instructions, you can save 'Repeat' program with the monitor, but be careful to save all bytes to $03FF (.S "0:REP" ,08,017A,03FF) and then append this to the DOS program using the COPY command

(OPEN1,8,15:PRINT#1,"CO: DOS-REP-MIX =0:PET DOS SUPPORT ,0:REP").

3. You might wish to include instruction for 'Repeat'. In this case list lines 250 to 260:
250 print" special commands start in col 1 and
260 print" are followed by a 2040 filename.
ready.

and change them using the editor to:
250 print   sys635 activate repeat "
260 print"   stop deactivate repeat "

Be careful with the number of spaces at the end of each line! They are included to keep the length of the BASIC program constant. To check this, peek locations 1792 & 1793. You should get 234 & 230 respectively.

4. Save the program with the Monitor (.S "0: DOS-REP-MIX ", 08,027A,0900).

In conclusion, I believe you will find that the time spent for entering this utility is very well compensated for by the pleasure you will have using it.

figure 3.

```
 5 a=12*16^3:rem $c000
 10 ifpeek(a)<>76thensys1639:rem basic2
 15 ifpeek(a)=76thensys2151:rem basic4
 20 print"SQQQQQQQQQQQQQ       universal dos support loadedQQQQQQQQ"
 30 new
ready.
```

Change this to:

```
 5 a=12*16^3
 10 ifpeek(a)<>76thensys1639:poke688,46:poke689,230:poke722,46:
    poke726,230
 15 ifpeek(a)=76thensys2151:rem basic4
 20 print"SQQQQQQQQQQ       universal dos support loadedQQ"
 30 print"       activate repeat with sys635Q"
 40 print"       deactivate rep. with Rstop QQQQQQ"
 50 new
ready.
```

KEYPRINT/826 (BASIC 4)

```
 M 0338 03C8
.:  0338 00 00 78 A9 03 85 91 A9
.:  0340 45 85 90 58 60 A5 97 C9
.:  0348 45 D0 03 20 51 03 4C 55
.:  0350 E4 A9 80 85 20 A9 00 85
.:  0358 1F A9 04 85 B0 85 D4 20
.:  0360 D5 F0 20 48 F1 A9 19 85
.:  0368 22 A9 0D 85 21 20 D2 FF
.:  0370 A9 11 AE 4C E8 E0 0C D0
.:  0378 02 A9 91 20 D2 FF A0 00
.:  0380 B1 1F 29 7F AA B1 1F 45
.:  0388 21 10 0B B1 1F 85 21 29
.:  0390 80 49 92 20 D2 FF 8A C9
.:  0398 20 B0 04 09 40 D0 0E C9
.:  03A0 40 90 0A C9 60 B0 04 09
.:  03A8 80 D0 02 49 C0 20 D2 FF
.:  03B0 C8 C0 28 90 CB A5 1F 69
.:  03B8 27 85 1F 90 02 E6 20 C6
.:  03C0 22 D0 A6 A9 0D 20 D2 FF
.:  03C8 4C CC FF FF FF FF FF FF
.\
```

SCREEN PRINT (BASIC 4)

```
 M 0338 03B8
.:  0338 00 00 A9 80 85 20 A9 00
.:  0340 85 1F A9 04 85 B0 85 D4
.:  0348 20 D5 F0 20 48 F1 A9 19
.:  0350 85 22 A9 0D 85 21 20 D2
.:  0358 FF A9 11 AE 4C E8 E0 0C
.:  0360 D0 02 A9 91 20 D2 FF A0
.:  0368 00 B1 1F 29 7F AA B1 1F
.:  0370 45 21 10 0B B1 1F 85 21
.:  0378 29 80 49 92 20 D2 FF 8A
.:  0380 C9 20 B0 04 09 40 D0 0E
.:  0388 C9 40 90 0A C9 60 B0 04
.:  0390 09 80 D0 02 49 C0 20 D2
.:  0398 FF C8 C0 28 90 CB A5 1F
.:  03A0 69 27 85 1F 90 02 E6 20
.:  03A8 C6 22 D0 A6 A9 0D 20 D2
.:  03B0 FF 4C CC FF 00 00 00 00
.:  03B8 00 00 00 00 00 00 00 00
.X
SYS826
```

Note:-
Only Bytes 0349 and 034C have to be changed in order to make the screen dump work on Basic 4.

BUY IT WRITE AWAY!

PAGEWRITER is a machine code word processor crammed into a single 2k chip! Obviously in a mere 2k we couldn't fit all of the facilities of WORDPRO, WORDCRAFT, or our own MICROSCRIPT, but you'll be pleasantly surprised to find how powerful and easy-to-use it is!

PAGEWRITER doesn't limit you to a 40 or 80 character line length, but scrolls the screen left or right, up or down as the cursor nears the edge. In fact, the electronic 'page' that you type onto can be up to 240 columns wide and up to 191 lines long (subject to memory size).

PAGEWRITER prints out your text exactly as you see it on the screen! There are no margin or tab settings to worry about. When writing or editing a document all the normal cursor controls may be used - and in control mode PAGEWRITER has more sophisticated functions enabling you to DELETE or INSERT a LINE, or MOVE a BLOCK of text. If you use the CBM 3022 or 4022 printer then PAGEWRITER gives you full control over the programmable character - as many as 26 characters can be defined at any time (a pre-defined set is included in the chip).

When you've finished writing you can SAVE text to cassette or disk. The whole thing is really so amazingly simple that you'll wonder why nobody thought of it before! And remember, because PAGEWRITER is written entirely in 6502 machine code it's FAST!

PAGEWRITER is available to fit in any spare ROM socket of an 8, 16 or 32k PET with New Roms or Basic 4 (please state socket & model when ordering). And the best thing of all is the price, just £39 plus VAT!

P.S. PAGEWRITER is also available in a 4k chip with ARROW, the chip that can LOAD, SAVE, VERIFY and APPEND at 6 to 7 times normal speed. ARROW on its own is £30 plus VAT, the two together cost £69 plus VAT.

ALSO IN OUR NEW CATALOGUE....

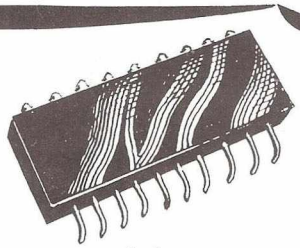
SUPERSORT (£40) heads the list of new utilities in our latest catalogue. Whereas MULTISORT (£25) will sort a string array and move a number of other string arrays around in parallel, SUPERSORT handles numeric arrays too, and will sort on one field within another all the way down the line.

DISK SEARCH 2 (£40) is an improved version of another powerful utility. If you need to search a RELATIVE FILE all you have to do is put the strings you are looking for into an array - then call up DISK SEARCH 2. You can request records that contain ANY ONE of the strings sought - or just those that contain ALL of the them. Why buy a database program when with DISK SEARCH 2 you can design your own?

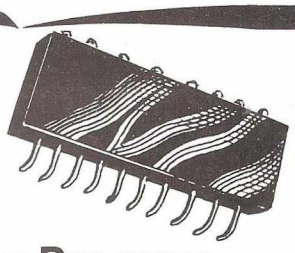
MAKRO DISASSEMBLER (£25) is a true disassembler - it will take a machine code program apart and turn it into ASSEMBLER SOURCE CODE complete with labels! If you own MAKRO ASSEMBLER (£50) you can then make changes and re-assemble the code - just as if it was one of your own programs! And we've developed a special version of MAKRO with a find-and-replace facility that you'll find particularly useful (put in your own labels etc). MAKRO-XR is available as an upgrade to the standard MAKRO at a cost of just £10.

SUPERSOFT

First Floor, 10–14 Canning Road, Wealdstone,
Harrow, Middlesex, HA3 7SJ, England
Telephone: 01-861 1166



audiogenic LTD CHIP SHOP



EDEX 2.0 & 4.1

adds commands to BASIC for use within your Program

IF THEN ELSE ● PLOT ● BEEP ● PRINTUSING ● SWAP MERGE ● HARD COPY ● PLUS A RANGE OF TOOLKIT TYPE FUNCTIONS AND A FAST EDITING SYSTEM

EDEX is an extension to BASIC which considerably enhances the potentialities of the Commodore PET/CBM It consists in a 4K-BYTE ROM which installs inside the PET/CBM.

EDEX is compatible with Commodore disk devices as well as with the DOS Support Program.
EDEX operation is fully transparent towards the Microsoft Basic Interpreter
EDEX is fully compatible with prior programs written without EDEX.

AUTO
Activates automatic line numbering.
APPEND *
Allows the creation of a program with a subroutine library
BEEP
Gives a sound of programable pitch and duration
CALL
Calls a machine language subroutine with transmission of up to 16 arguments
DELETE
Allows multiple line suppression
DUMP
Lists all variables in a program, together with their values
EDITING *
e.g. @ M prints MID$
ERROR
Shows where an error has occurred
FIND
Lists all lines where a given character string is present

HARD COPY □
Dumps screen to printer
IF THEN ELSE
With up to 16 nested conditions
MERGE □
Merge two programs files
PLOT
Plots curves of 50 x 80 or 160 resolution
PRINTUSING
Formats printing on screen or any printer
RENU
Program renumbering
RESET
Suppresses a dot (contrary of PLOT)
SWAP □
Swap one program for another keeping variables
TRACE □
Single line execution (displayed at top of PET)

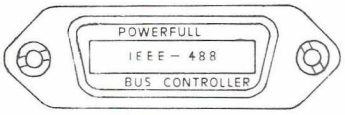
*** EDEX 2.0 only □ EDEX 4.1 only**

EDEX 2.0 for use with BASIC 2 40 Column Pets **£39.50**

EDEX 4.1 for use with 80 Column Pets **£49.50**

Available shortly for BASIC 4 40 Column PETs

IEEE-488 PACK



The end of instrumentation's problems. It resolves all kind of troubles:
- Time-out
- Special characters ("null", and so on...)

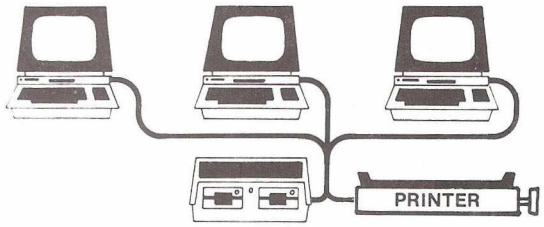
IEEE-PACK allows the use of IEEE-488 Universal Commands:

- DCL (Device clear)
- SPE (Serial poll enable)
- LLO (Local lockout)
- PPL (Parallel poll configure)
- SDC (Selective device clear)
- SPD (Serial poll disable)
- GTL (Goto local
- PPU (Parallel poll unconfigured)

IEEE-PACK also allows BASIC interrupt with functions:
- ONKEY "x", line number
- ONSRQ line number (On Service Request)

IEEE-PACK comes complete with two ROMs. **£89.50**

MULTEX



MULTEX allows several CBM 8032 to work together on the same peripherals.

MULTEX is a ROM which replaces a ROM of the CBM 8032.

Except the substitution of this ROM no other modification is required on the CBM 8032.

MULTEX is much cheaper than any other system.

MULTEX £69.50

Access DINERS

ALL PRICES INCLUDE V.A.T. & P.P.
AVAILABLE FROM ALL GOOD DEALERS OR DIRECT FROM

AMERICAN EXPRESS BARCLAYCARD VISA

AUDIOGENIC, P.O. Box 88, 34-36 Crown Street, Reading, Berks. Tel: Reading (0734) 595269