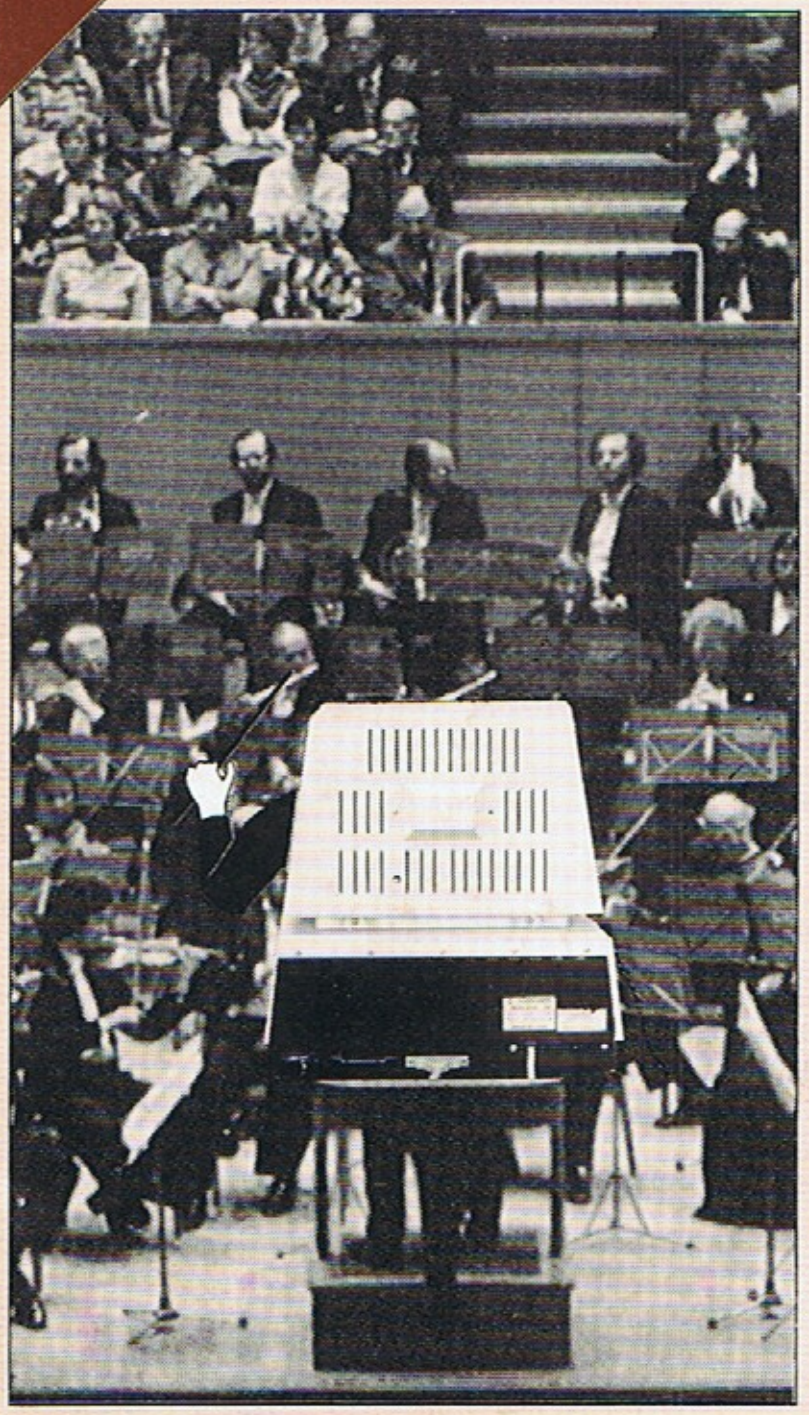
COMMUNICATIONS
ALGORITHMS FOR
INTELLIGENT TERMINAL
OPERATION

COMMODORE CLUB NEWS

October 1981



Volume 3 Issue 9

WORDFORM

Review of the latest word processing package

SORTING TECHNIQUES

The definitive article

DISK USE FOR BEGINNERS

Part 6 of our special course

BASIC COMPILERS

Start of a series of articles on "how to write a compiler"

PLOTTING MULTIPLE FUNCTIONS

How it's done

commodore COMPUTER

Contents

Communications Special

Next Month

In next month's issue we'll have all our well known favourites, including a couple of articles from Jim Butterfield, Disk Drive for beginners, What the Papers Say, and so on. Newcomers include a beginners guide to using the User Port, a number of hints for 8032 users, peripheral spot reverts to disk drives again, and of course the usual Basic and Machine Code sections, as well as the news and reviews we always feature.

The special pullout section is back to education again, and includes a couple of articles from Nick Green, Commodore's Special Projects Manager, one on a visit to a comprehensive school, and the other on topic lists. Of special interest is a report from the Comal Users Group in the States : everything you ever wanted to know about Comal, but were afraid to ask!

Editorial

Autumn is with us once again, but that doesn't stop the newsletter rolling on -we're almost back to the publication schedule that appeared around this time last year! By the end of the year, we'll be back on target once again when volume 3 issue 11 comes out. 1982 will definitely see the first ever 12 issue volume of the newsletter.

Another item you'll be seeing shortly is "The Best of Commodore Club News Volume 3", covering all the best articles, programs etc. that have been printed in Club News over the year 1981. Now some of you may argue "Best of ... Volume 3" ? When did "Best of ... Volume 2" appear, you may well ask. Well, the original "Best of ... " was a compilation of the first two years, and it has now been decided to produce such a book every year ; hence volume three. I was beginning to loose myself there!

What's in store

Anyway, what have we got in store for you this time ? As ever, our base of articles that appear every month. Jim Butterfield, this time talking about the 12" 4032s, and some differences you can expect to find between them and the earlier 9" models. Savour this article, because we might not have one next time around. Canada is having a postal strike at the moment, and as Jim lives in Canada ...! But knowing him I'm sure he'll do his best to come through to us.

Disk Use for beginners continues to wend its merry way inside the Commodore disk drives. If you've been following this series from the beginning (all those lessons ago) you should by now have a very good idea of how to make the most of your disk drive. Certainly these articles, coupled with the many other items and programs on disk drives, should have helped even the complete tyro to get going. I hope so anyway.

What the Papers Say is again full of weird and wonderful stories about PETs and their many uses, culled from a variety of magazines, not all of them computer based either. We get everywhere!

The usual Basic and machine code sections, the Basic covering such items as an all-purpose INPUT subroutine, multi-dimensional arrays, driving the user port in Basic, and a few more besides. For machine code addicts we have a way of making your programs more secure, flipping strings, the start of a series of articles on how to write you own compiler and more.

Peripheral Spot

This time the section concerns itself with the 8010 modem, and features a number of contributors who have each made a name for themselves in the PET world. Jim Butterfield needs no introduction, and we highlight a program he developed to get you on the air. Paul Higginbottom, Commodore legend in his own lifetime, gives us some of his early discoveries with the 8010, in order to save you the head scratching that he went through. Communications is here to stay!
went through. Communications is here to stay!

What other wonders have we got ? The reviews section, this time mainly featuring the major new word processor to appear, namely Wordform. At £75.00 it can't be in the same league as the 'big two' you might say. Read on.....

Sorting techniques get taken apart in an excellent article from Italy. Well laid out, well documented, clear listings and flow charts - a model example for all to follow. I know we've featured sorts extensively before, but as it is probably the most commonly encountered problem in the world of micros I feel it deserves all the coverage we can give it. Especially when the article is as good as this (I can say things like that because I didn't write it!).

Power, an amazing new chip from Professional Software, is also reviewed in this months issue. At only £49.50 this represents very good value for money, as Barry Miles tell us in his review.

Communications

The special section, alternating with the bi-monthly educational features, concentrates once more on communications, proving just how important this particular side of PET activity is becomming. Another excellent article by Dr. Phillip Barker of the Teeside Polytechnic (who wrote the article two issues ago on Using a Microcomputer as an Interactive Ter-

The editor doing some work for a change!

minal), concentrating this time on Algorithms for Intelligent Terminal Operation. I suspect we'll be hearing more from Dr. Barker in the near future, as obviously his knowledge of communications is very good.

So that, along with all the other various goodies, is what you can expect to find this time around.

The Future

But let's, for a while, look to the future.

As I write this I'm happily ensconced in Commodore's brand new premises at 675 Ajax Avenue on the Slough Trading Estate. The view out of the window may not exactly be 'herds of wildebeeste sweeping majestically across the plains, the Sydney Opera House away on the horizon' as someone just about said once, but the exterior is unimportant. It's what goes on inside that counts.

The building's designed to cope with our expected expansion for the next five years. Looking earlier at the vast amount of warehouse space we now have at our disposal (big enough to hold 43,518 elephants - well, little ones anyway) I can well agree with that. The future, at the moment, is looking good for Commodore. And that means good for you.

Introducing products like the VIC at one end of the scale and the micro mainframe at the other, means that we now can cover a complete range of users, from the hobbyist to the big time businessman, from the school teacher to the mainframe man. What next?

Well that is rather difficult to predict. Five years ago no-one would have been able to describe the micro mainframe, for instance. Ten years ago even the (now) humble calculator was almost unheard of. It is said at the moment that a new micro is appearing on the scene every month. So to say what is going to appear in five or ten years time is something of an impossibility. And I'm not going to try! Just keep on subscribing and find out!

Perhaps you would like to write in with your ideas of what the scene will be like in the year 1991. The most original and amusing article will be published with all due fanfare and publicity in a future newsletter.

The address for that, and any other contributions (and keep 'em coming -it's your magazine) is the new one :-

The Editor
Commodore Business Machines
675 Ajax Avenue
Trading Estate
Slough
Berks.

Subscriptions

To subscribe to the magazine, simply send a cheque for £10.00 (or £15.00 for overseas membership to cover increased postage), made payable to C.B.M. (U.K.) Ltd., to the above address. That will give you a year's supply (12 issues) of the magazine. Keep yourself informed!

PET Cake

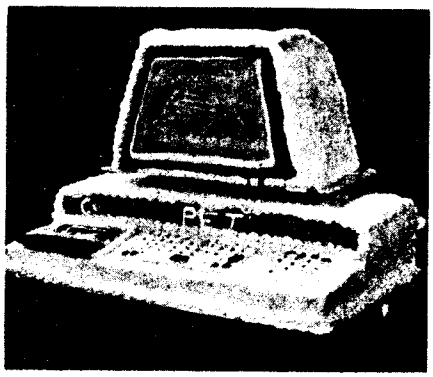
Describing our low-cost versatile PET computer normally requires terms such as ROM, BASIC interpreter, green phosphor screen, RAM, parallel port, cassette ports, etc.

But have you seen a PET computer that contains "ingredients" such as: eight pounds of white frosting, fudge filling, eight white cake mixes, candy bars, and dyed, uncooked alphabet macaroni.

The PET cake was created by Sarah Jo Gilpin-Bishop and her husband Frank Bishop to mark the third anniversary of their computer group.

The PET cake was creted by Sarah Jo Gilpin-Bishop and her husband Frank Bishop to mark the third anniversary of their computer group. By the way, it should be noted that consumption of the PET cake was total.

We've always said PET Computers are easy to use. This just proves they're a piece of cake!
Any other PET recipes I'll happily publish!



Not the last program on the PET but certainly in advance of anything else

Stage One Computers do not claim that THE ADMINISTRATOR is the last program you will ever require on your PET microcomputer, BUT they do however claim that THE ADMINISTRATOR is the first program on the PET and other microcomputers that actually allows the user to set up his own system incorporating records with vairable amounts of information contained there-on. Such applications as: Sales Accounts, Purchase Accounts, Nominal Accounts, Cost Ledgers, Stock Ledgers. All types of recording applications which require variable numbers of entries or postings or visit records may be set up using this system to the customer's own format.

NEIL HEWITT, the Senior Partner of Stage One Computers of Bournemouth, Dorset, states:
"There is no such thing as the ultimate program - there is always room for improvement - the ultimate is a dream - one worth striving for, but never achievable. THE ADMINISTRATOR is the closest thing to the "ultimate" available on the Commodore PET. Systems like this have so far always restricted the user in the amount of information that he can store with one record. THE ADMINISTRATOR overcomes this problem, and covers everything other systems handle".

There are over 2000 users of the now familiar PETAID range of programs and:
"THE ADMINISTRATOR will really be the Winner" - says Neil Hewitt.

VICaid

Over a chat with well known Ruddles entrepeneur Chris Palmer, of the VIC centre (tel. 01 992 9904 for further info) I got the lowdown on a number of developments that were happening there. For instance, the game of games appears to be with us. Called Monster Maze, you are in control of a man walking through a maze, accompanied with the sound of his footsteps. You have to avoid the VIC monsters as they appear, with more and more of them arriving as you succeed in advancing further and further through the maze. Accompanied by the fabulous sound, graphics and colour which the VIC has on board, Chris tells me it will be a real winner. Go down there for a demo if you want to see it, and other astounding games

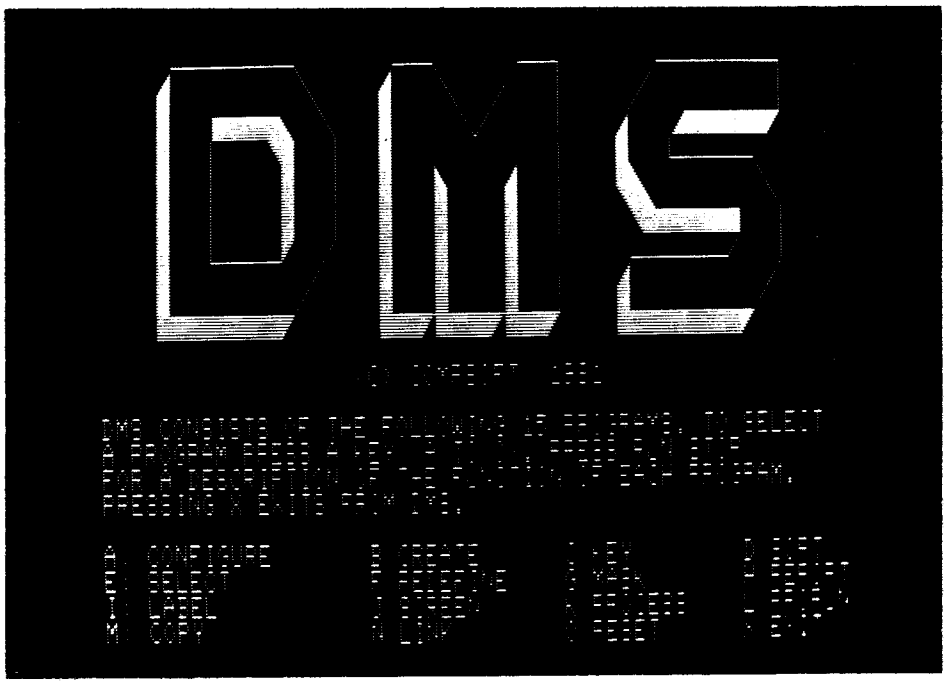
and programs generally. Their address is :-
The VIC Centre
154 Victoria Road
Acton
London W3
As VICs become more and more a way of life, I'll be going down there for an indepth report on what's going on. Watch this space!

Compsoft Launches New DMS

The New DMS for Commodore machines has broken the limitations of current record keeping systems with several features never previously seen on Pets.

But don't worry, users of the existing system can trade in the old version for fifty pounds and reap the benefits that new users have as standard. Particular benefits include speed of operation, in that the New DMS is written almost entirely in machine code, and the ability to create and store records of up to a thousand characters in length. The new record size, four times the limit imposed by the previous system, can also hold long chunks of descriptive text (up to 255 characters in a field,) and is particularly suitable for personnel records, library records, student records etc, where full details are required.

Other major benefits include a report generator that can produce letters or reports along any lines, with totals and sub-totals as required.

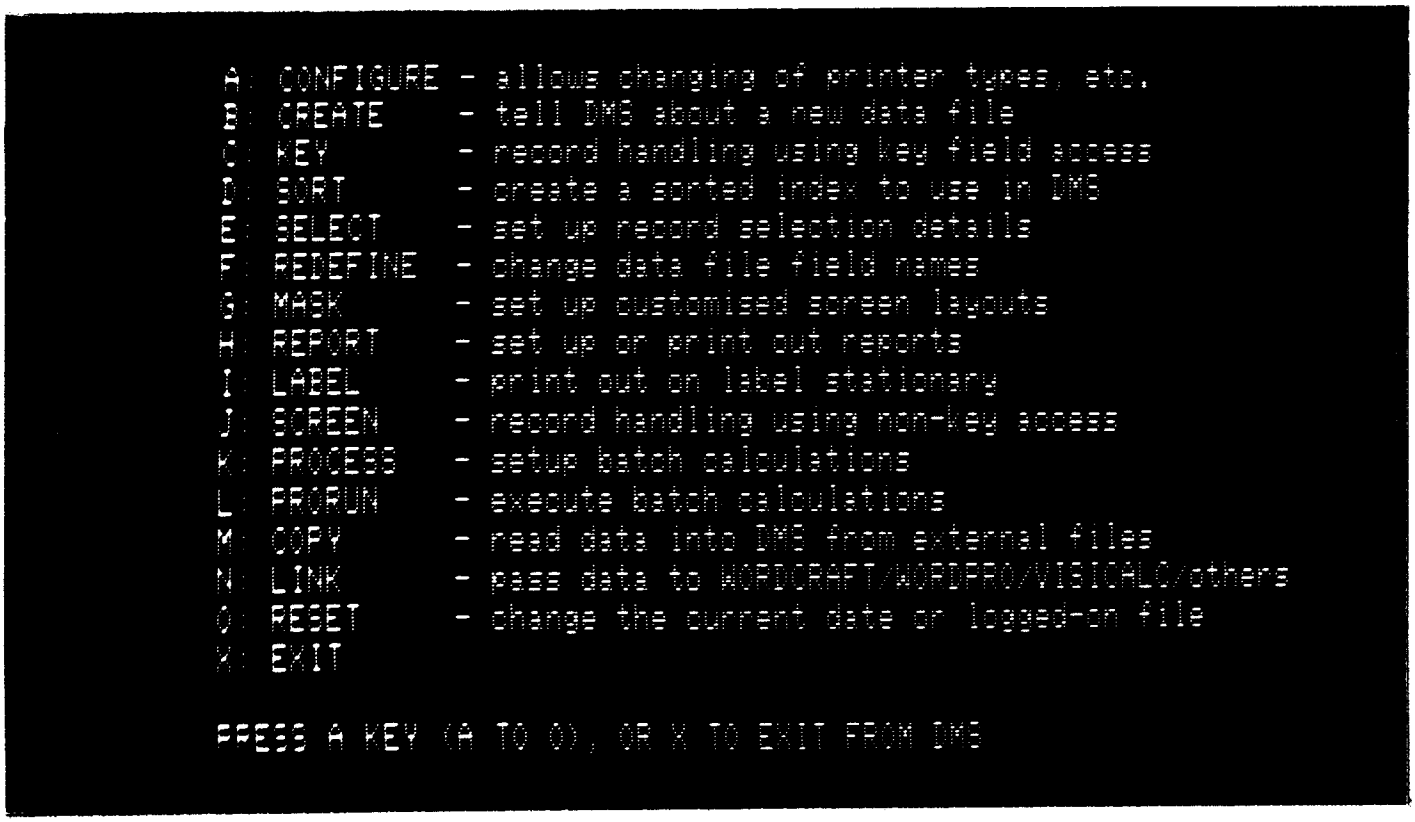


Report formats are storable, and may be recalled and modifed at any time. Any number of special screen displays can be produced and recalled for use or modification in very much the same way.

Compsoft stress that the New DMS is not a modification of the old one, but is a complete rewrite. Many of the features and facilities are based on those available in the CP/M version, such as genuine date programs, on screen editing etc. The links to Wordcraft, Wordpro and Visicalc still come as standard, but the best news is that the price remains unchanged at around 250 pounds, inclusive of links and new manual. Particular care has been taken to increase DMS's ease of use, and speed up the learning time. An interactive method of communication is used throughout, with or screen commands in every day conversational English.

The New DMS (Data Management System) is available for Commodore 4000 and 8000 machines, with the standard DMS available for the 3000 systems. Owners of machines under the CP/M operating system can also buy the DMS suite of programs written specially for them.

Compsoft Ltd, Great Tangley Manor Farm, Wonersh, Guildford, Surrey GU5 0PT Tel Guildford (0493) 505918



Computer Assisted Learning

As you may already know, the Schools Council Project, Computers in the Curriculum has been under way for around eight years, building on the earlier work of the Chelsea Science Simulation Project. Over the last year, the Project has also received funding from the Microelectronics in Education Programme (M.E.P.) under Richard Fotherfill, and is likely to be funded for a further three years from this source.

In the early seventies, computer programs written to provide C.A.L. (Computer Assisted Learning) opportunities at school level in subjects other than Maths and Computer Studies were accessed from the school by means of the school - central mainframe computer telephone link which some forward-looking Local Education Authorities established in their areas. A decision on a suitable computer 'language' was also taken then in advance of the rapid changes of the last few years. As it happens, the current wave of microcomputers in use in education, small businesses and even in the home environment nearly all talk this language. The language is called BASIC (Beginners All-purpose Symbolic Instruction Code) and was developed in America back in the sixties at Dartmouth College, New Hampshire. It was intended then as a programming language for beginners, but has been enhanced by many computer software suppliers (program producers) and even by hardware manufacturers (machine producers) such that it now often bears little resemblance to its original specifications. There is much debate currently about the suitability of the language for all the widespread tasks which it is being asked to perform. However, without becoming involved in the arguments about BASIC, the facts are that it is virtually the ONLY language normally supplied on these current machines by the manufacturers, and software producers must lie with the problems this causes.

With the appearance of the microcomputer over the last three years or so, the Project has been faced with a number of problems with regard to providing access to its CAL materials for the user of the microcomputer. The most severe of these was the fact that different machines generally 'speak' different dialects of the BASIC language, often necessitating minor, or even major, changes in the CAL program to enable it to run on any machine other than the one it was designed upon. Display formats (now on Video rather than paper roll) also differ on different machines and the devices used for sending data to and receiving output from machines are already proliferating and are likely to do so even more in the near future. Examples here include touch sensitive keyboards and speech recognition devices as opposed to the normal QWERTY keyboard for input, and printers, graph plotter and speech synthesizer units, as opposed to the video dispay, for output.

All of these devices require special software (programming) which will involve minor or major changes to the original program.

The Project has therefore found itself in a position that with relatively limited resources, a system was required whereby a CAL program having been developed and coded ONCE, would be guaranteed to produce identical output on any microcomputer system that was available, using any peripheral device that is either available now or likely to be so within the near future.

The Library of Routines has therefore been developed with this aim in mind. The Project accepts that some of the routines are not perfect, but at least it is a working piece of software with a specific aim in mind, that being to reduce the time and effort necessary to bring good CAL material to as wide an audience as possible in the school environment.

All future software developed by the Project will be based upon the Library, but in addition it has been decided to make the package available to any developer working in the CAL or any field who has a requirement for producing machine independent software.

With experience, the software will be improved and provide further facilities (routines). Continued software backup will be provided to all purchasers of the Library.

As far as the user is concerned, the Library consists of a set of subroutines provided either on a floppy-disk or cassette which can then be loaded into the host machine. The actual disk or cassette is still a machine dependent feature of the utility since they usually cannot be made machine independent by software means. The Library itself is **also** machine dependant since the routines are defined functionally and certain routines will contain machine - specific BASIC and machine code. It is only these routines that have to be changed to make the Library operate in a new machine, or handle a new device. These 'versions' of the Library are available from the Project, and currently we can supply full versions available for the Commodore PET and Sorcerer machines. In the near future we expect to be able to provide a full version for the Sharp MZ80K and Tandy TRS 80 machines.

With the Library loaded, all that remains for the developer is to write his 'driver' program in a standard dialect of BASIC (ANSI minimal), using the appropriate subroutine calls to the

Library, and that developer will be writing a program which will produce equivalent output on any other machine on which the Library has been implemented. It must be appreciated however that account must still be taken of possible differences in appearance of output on different 'width' screens, and the fact that high resolution graphics can't be provided on machines without such support hardware!

This program can then either be saved to disk or cassette together with the Library, or transferred alone by wire connection to another machine where it can be linked (merged) with that machine's version of the Library. The combined Library and 'driver' program can then be executed by the new machine.

I applogise if this description is rather long-winded, but the problem is an extemely important and rather technical one, and I feel that the only way to describe what we are now able to do, is to provide you with this layman's summary.

What the Papers Say

Another diverse list of applications this month, with quite a number concentrating on the area of PETs in education. I'm grateful, as ever, to the people who supply these press clippings : I'm sure that it isn't the most exciting job in the world to perform, but thank you. The things we do for readers!

Keeping the educational ones to the end, we commence with PETs making cups of tea. The Sunday Times had a feature on the interface recently developed by I.C.I., and the making of the cup of tea was done to demonstrate the versatility of the device. You get asked by the PET (which is where we fit in) how many lumps of sugar you want, how much milk and finally how hot the water should be. Pretty indifferent tea, according to the Times, but where the system scores is in the home, where you could have the system controlling ventilation, switching heaters on and off, and even watering the house plants. Obviously it's going to be of most use in the laboratory, and indeed this is where most of its sales have been so far. For further details contact Ray Broadbridge (the devisor and assembler of the gadget) of I.C.I. on 0642 553601 extension 3752.

Stepper Motors

Onto Stepper Motors and the BKSTS Journal, who run a feature on the Digital Stepper motor drives, as distributed by The Interlock Shop. These provide a control unit offering absolute synchronisation working with multi-image film projectors, film cameras, dubbing and film studio equipment, special effects and zoom lens drives. These are controlled by a PET, needless to say, and further details are available from the Interlock Shop, 61a High Street, Hampton Hill, Middlesex TW12 1NH.

The Financial Times (only the best!) covered the recent Wall Street Afternoon Recovery, when a number of shares climbed after a very sharp slide. Commodore was up 2 1/8 to $32 3/8. Not very timely I know, but we do try and cover everything.

Moving onto the 8096, Computer Weekly carried two articles on it. The first one, by Claire Gooding, had Commodore's Marketing Manager Keith Hall commenting "The 8096 is going to sell because of the software available for it, and because of the tremendous amount of money available to use that software. Becasue Britain is so rich in software we want to be able to offer it for sale here, quickly." Hence making the decision to make them in this country. At present something like 80% of manufacture takes place in Germany, and 20% in the U.S.A. Although it will only be a small operation at first, it is anticipated to grow in size rapidly.

This decision to make 8096s in this country, and the reasons behind that decision, are also the starting items in the other article, although it goes into more detail on one particular piece of software, namely Silicon Office. Mike McDonald of Bristol Software Factory, the people behind Silicon Office, said "British software really IS the best in the world". Being Canadian, he should be capable of unbiased judgement. A quick run down of Silicon Office is given as well, describing it as having all the elements of an integrated package - data base management system, word processor, communications and many other features included - but much more, as Mike McDonald points out. "It is programmable. It is not a code generator, but it does a great deal of the work for you, at the higher level of designing the system". Communications is also emphasised, stressing that Silicon Offices can be linked up wherever the telephones can reach.

The Business Systems & Equipment magazines has a brief mention for a Stoke-On-Trent housing association which has recently acquired a PET. This is being used for rent accouting, purchase ledger and analysis of the two. According to their director, work that used to take two days a week can now be done on the computer in less than half a day. Should be a lesson for all of us in that.

Business Games

Keeping with business for a moment, a magazine called Training Officer has a report on a company called Belvoir Business Games, who produce a variety of somewhat unusual business games. Selling package holidays in Spain, operating prime sites around a market square in a provincial town and running different types of catering establishments on them, a fully interactive business game for up to 16 teams or competitors in which 4 different types of chocolate bars are produced, and a number of others. For further information ring 0949 (that's Melton Mowbray) 61318, and ask for Mr. Atkins.

Campaign, an appropriate name for a magazine featuring a story on our latest advertising campaign, costing half a million pounds. Ronnie Barker is featured in a series of seven ads aimed at businessmen and middle management executives. Print Promotions Publicity are the people behind this for us, and their creative director Paul Carberry said "We wanted to produce a 'friendly' campaign that seemed right for the target market". Commodore claim 45% of the microcomputer market in the U.K. : this campaign intends to make that higher.

A couple of articles on the recent Personal Computer World Show, which featured a Commodore presence. The Times merely mentioned Commodore, and had a picture of a number of VICs in action, but the magazine Marketing carried a much more Commodore oriented

report. Keith Hall, Commodore's Marketing Manager, was heard to say "We agreed to come because we were promised a separate area for business systems, and ... because of the launch of our consumer micro, the VIC 20. We are finding that visitors are asking the enthusiast questions on the VIC 20 stall, while up here we're doing a brisk trade with serious businessmen".

Education

And so onto education. Educational Computing, surprisingly enough, kick us off with a mention for the forthcoming first ever conference devoted to educational users of the PET, coming up at the end of the November. It discusses the topics to be covered (see the handout last month), and ends by saying "perhaps the conference will at last indicate the real penetration of micros".

Computing carries an item on a recent report from the Local Authorities Management Serivces and Computer Committee (Lamsac), which states that the PET accounts for almost two-thirds of micros used by councils: over 200 machines are already used by local authority departments. A spokesman for Lamsac said of microcomputing that "traditionally some local authority departments have been considered poor relations, not recognised as high priority to make use of hard-pressed central computing facilities. So some users have been buying micros as business machines and done their processing locally".

An unknown magazine (so apologies straight away if you recognise the story) contributes a report about a lucky school that found itself being given a PET by a Northfield firm called Kalamazoo Business Systems. The school in question is the Lickey First and Middle School, and their headmaster, Mr. David Slater, was quoted as saying "this is an extremely valuable asset to the school, because computers are very much the thing of the future". The PET was won by Kalamazoo in a naitonal 'Superstars' competition held in Holland, and sponsored by Commodore. The reason for giving the PET was that the captain of the team, Mr. Richard Jephcott, has two children at the school, and so now they can all benefit from the use of a microcomputer.

Computer Weekly had a letter from Don Walton, headmaster at Houghton Country Primary School in Huntingdon, which raised a number of interesting points. Faced with a couple of problems, kids who are determined to ferret out the inner workings of the PET (no subjective assessment), and needing to introduce this assessment in BASIC programming, COMAL seemed to be the answer. But two further problems emerge. One is the cost of buying a new ROM board in order to be able to use COMAL, and the other is the DoI's 50% funding of the BBC/Acorn /Proton machine for any secondary school which has not already obtained a microcomputer. He concludes thus: "The choice is clear : either the school funds its own hardware 100% and therefore has the structured COMAL, or it gets help from the DoI for a new microcomputer which does not".

Finally for this month we turn to New Scientist. They have a featrure on Lowbrook Primary School in Maidenhead, Berkshire, who've acquired for themselves a PET. Among its many uses will be as a computerrised data base for selecting books from the school library : it gives you a run down on all the books in a particular subject field. As well as this, it (and two other PETs) are also running many educational programs in a bid to get children acquainted with the mighty micro as soon as possible.

That's all for this month : see you next time around!

Basic Compilers

Continued from page 32

8) File handling.
9) Other extensions and operating procedures.

Some of these stages may be short enough to be merged with others, while some, e.g. 2), may need more than one stage to treat in sufficient detail.

A simpler proposal for a BASIC compiler appeared in the PETs Corner section of the 5th edition of the Liverpool Software Gazette, and this proposal has been developed from that original. It was proposed that a number of people with access to the needed equipment attempt a group implementation of the compiler. Anyone interested in this scheme is invited to contact the author.

John Stout
6 College Ave,
Formby,
Liverpool.
L37 3JJ

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.

Review

Peter Gerrard

One of the major applications for any PET system must be word processing. I'm sure there are many systems that are dedicated solely to this application, and probably using one or the other of Wordpro or Wordcraft.

Since these programs have for some time been the only two major word processing packages on the market, the last two issues of the magazine have featured fairly extensive reveiws of both of those products.

However, some companies may have had doubts about buying either of them owing to the cost. To offset that a new product has appeared, going under the name of Wordform, and costing just seventy five pounds.

Wordform from Landsoft

Versions exist for every Commodore machine (I think! Check with your local dealer), and the one under review was being used on an 80 column PET.

Fracton of the cost

How can a package at a fraction of the cost possibly offer the same facilities as its two "big brothers" ? The answer that it doesn't! It doesn't attempt to copy everything that those packages can achieve, but sets about providing the majority of features that you would ever need from a dedicated work processor. What you don't get I'll come to later, but let's first of all examine what you do get.

The manual looked unnervingly simple - just 18 pages of it. This meant one of two things. Either it was a product that required very little explanation and was easy to use, or it had very few features. The logical procedure to follow on getting a new product is to go through every feature on offer, to get an overview of the package and what's available, and then begin to integrate the parts into a whole.

On loading and running the program, and being amazed that it only occupies 6K of RAM, the initial display shows the word WORDFORM, tells you you're in primary mode, and a right arrow sits and waits for you to tell it something. Now is the time where a manual can quite often make or break a program. The program might be the best machine code program ever written, but if the manual doesn't tell you what to do it's wasted.

Congratulations

Congratulations to Landsoft on the manual. It is simple, precise and to the point, and every function is clearly explained. There's also a nice summary index at the back which explains what everything does. Saves fumbling through the manual trying to find the right part.

The first thing to do is tell it what printer you're using. Typing in T, followed by P (for PET) or A (for Any other) does the job. The next thing to do is specify length of page, number of lines to be printed on, on what line the page number will be printed on and on what number the page number must start. You can dispense with the last two if you require it e.g. for letters etc.

In Primary mode there are commands for loading files, saving them, scratching them, clearing text areas, printing, exiting the program, and entering the actual text/editing mode of the package, known as Scan/Edit mode. To do this you specify the maximum number of characters that will be required in a line of text, and away you go!

Unlike Wordpro, for instance, Wordfrom rapidly establishes itself as a "What you see is what you get" word processor. You type inside a screen window, defined by the maximum number of characters you typed in earlier, and by a border at top and bottom. They've tried to make the program act as much like a typewriter as possible, in as much that the bottom of the window displays a "ruler" lined at every tenth character, and as you type the "paper", or screen window in reality of course, moves left and the "thimble", or position of the cursor, stays where it is.

Comfort to typists

This will be a comfort to typists to whom you're trying to introduce word processing. It's a familiar sight, rather than the perhaps daunting (at first sight) appearance of other word processors. What they perhaps won't like so much, and here we come to my first criticism of Wordform, is the speed at which the program reacts to the keys being depressed. Whilst not affecting typists like me so much (two fingers on a good day!), the exceptionally fast touch typist will find him/herself getting ahead of what is appearing on the screen, with the result that occasionally garbage will appear. Off putting to the newcomer to the world of word processing, but I would guess that in the vast majority of cases this wouldn't make any difference to people.

This now brings me to a criticism of word processing packages, generally, but alas one from which there is no escape. It highlights both the strength, and a weakness, of all microcomputers. Not being dedicated word processors the various keys have to have different functions assigned to them. No problem for the secretary who will only ever use the system for that one purpose, but perhaps disconcerting for the computer user who has many different applications. Still, no particular complaint about Wordform but about all of them in general. Mind you, at the end of the day you've still got a microcomputer on your desk!

A feature of Wordform is the way the words appear on the screen. They do not wrap around, and if you've got, say, three spaces to the right hand border, and the word you're about to type contains five characters, as you begin to type the fourth the whole word jumps down to the start of the next line.

Major consideration

As a major consideration before purchasing any word processor ought to be the editing facilities that are on offer, it is time to examine those.

Movement about the screen within the text area ocurs in one of two ways. Either in keyboard mode or out of it, the normal cursor keys will move you around the screen to wherever you want to be. Outside of keyboard mode however you can use the numeric pad to zip you about horizontally and vertically. One complaint though, and that is that I wish they had used the logical process of pressing 2 to go down, 4 to go left etc. Still, you soon get used to it.

For correction of simple spelling mistakes it is a case of going back to the incorrect word and just typing over with the correct letters. To delete text, you use the delete key in the normal way, and to insert text you don't use the insert key in the normal way! Pressing the off/rvs key puts you into insert mode, and then you can just type in as normal. Words will seemingly disappear off the right hand edge of the window, only to reappear again on the next line down. Pressing off/rvs again takes you out of insert mode and into ordinary typing mode. Awkward at first, but as with all functions of this type it soon becomes second nature.

Moving text

Deleting lines and inserting lines are both there, along with deleting blocks of text, copying a block of text from one area to another, and moving a block of text from one area to another (this latter removes the original text, the former retains both it and copied block). All of these functions are very easy to perform, and performing them rapidly becomes very quick and easy. I would imagine that any typist would soon become familiar with these actions, as well as the area covered in the previous paragraph.

Right justification is also offered, but in a somewhat strange manner.

As with every 'action' in Wordform (or indeed any other word processing package) you have to press the right keys in the right order to get the right thing to happen. One thing I am forever forgetting to do, and this is not a criticism but is something you'll encounter if you buy the package, is to press 'K' before commencing to type. This puts you into keyboard mode, and thus what you type appears on the screen. Nothing disastrous has ever happened, as only a couple of letters go by before noticing that nothing is in fact coming up on the screen. However, stopping one's self from doing this is only a question of time, not a function of the program.

Perform right justification

To perform right justification on your text, if it is desired, you press the 'CLR' key to clear you from keyboard mode, followed by 'R'. As is the case with all commands, the major mode you are in is shown at the top centre of the screen, with the 'sub mode' (e.g. keyboard, right justification etc.) at the left hand side of the screen next to a little right hand pointing arrow.

Having pressed 'R' the cursor (or rather a little up arrow which always remains on the base line and tells where you are) jumps to the right hand side of the screen. If for some reason justification is required to a position short of the right hand border, move the cursor back to the desired position before doing anything else, although I can foresee very few places where this would be needed.

One final point to consider before proceeding further is the TAB function of Wordform. Tabs are set, appropriately enough, by the TAB key when the cursor has reached the desired position. Unfortunately you can only have one tab at a time, which at times can be mildly irritating. Hitting the RETURN key at any time then takes you to the set position. I found the most sensible thing was always to have the tab at the left hand boundary of the window.

Tabs having been set, we're ready to go! Right justification takes place between the tab position and the position of the cursor, so if the tab is on the left hand boundary and the cursor is on the right hand boundary then justification will take place over the whole line. Hitting RETURN defines the line you want to start justification from, and then moving the cursor down to the line you want to stop at, and hitting RETURN again defines the bottom limit. It all then happens before your very eyes.

Quick and easy

That may have made the whole procedure sound terribly complicated - it isn't, and as with so many other things soon becomes very quick and easy to perform. No, my main complaint is this ; it justifies the last line of a paragraph as well! In other words, if you've got just two words in that last line, you'll have one at one end of the line and the other at the other end. However, there is an escape route.

That method of escape is as follows, and is a procedure I would thoroughly recommend if you're going to perform right justification. Simply justify your documents paragraph by paragraph as you write them, omitting the last line each time. The action only takes a few seconds to perform, and guarantees a neat looking print-out each time.

Left justification is also offered i.e. the reverse of right justification, in case you decide that perhaps after all it would look better as it was.

Form letter capability

The form letter capability of Wordform is not as great as that of the other two, but nonethless the function exists, and is relatively easy to perform. Where it fails is the lack of an ability to have masses of names and addresses lying in the extra text area (or storage buffer, as Wordform calls it), the standard letter in the main test area, and for them all to be printed out one after another. But again this can be got around by typing in one name and address (say), printing out the one letter, and then typing in the next name and address, and so on. After all, they've all got to be typed in at some point or another.

String finding, and altering and replacing strings, can certainly be done on a local basis, but you lack the ability to find and change on a global basis (i.e. pulling in one file, altering everything, putting that file back and getting the next one, and so on), although you can certainly print on a global basis. What it does have however is the ability to selectively alter strings. For instance you want to change all occurences of "cat" to "dog", but don't want to change "catalogue" to "dogalogue" when the program finds "catalogue" you have an option of changing it or carrying on to the next occurrence of "cat". Very useful.

Another valuable feature is the ability to transcribe text areas from the main text area, and vice versa. As a Wordpro user normally, many times I've wanted to take part of the main document and push it into extra text, call up another document and re-insert that part of the first document into the second one. You can do it, true, but it does tend to get terribly complicated. Here it is refreshingly simple.

On the other hand, with Wordpro it is very simple to change the settings of the margins, or in other words the width of a pargraph. With Wordform it is not so easy, although again it can be done. For the benefit of those of you who already own the package there is a mistake in the manual when it is describing decreasing the paragraph length (page 12). There should be a step 5 1/2 - come out of Insert sub-mode. I've checked this a number of times and my method works each time, although the way described in the manual does not. My only complaint about the documentation.

Final comments

My final comments would be that 1) it is a shame that you cannot look at a disk directory without having to exist the program and re-running it, thus losing whatever text you had at the time - always save it first! 2) is the fact that you can't do a 'save with replace' of a text file, although that is possibly a good idea in view of the hiatus that surrounds that particular command. Instead you must scratch the original file and then save your amended version. As I say though, this could well be a wise move, and is probably the reason why the program operates in this way.

To sum up

To sum up, Wordform does not offer the vast range of functions that Wordcraft or Wordpro do, but nevertheless probably performs about 90% of them (or at least the ones that get used 99% of the time). It also offers one or two extra features itself, such as the swapping of text from one area to another, which really is very useful, and the selective altering and changing for instance.

It will certainly fulfill the word processing needs of the majority of people, and once you've mastered the variety of key strokes needed to do everything is perhaps easier to operate than the current "big two". A very good program, simply presented and packaged, and with none of the worries of yet another ROM to go inside your PET, or a family of dongles living at the back of the machine. At seventy five pounds it represents extremely good value for money, and is a good place to start in the word processing arena.

Review : Power

Barry Miles

This is a4K chip, available for 2000, 3000, 4000 & 8000 series machines. It goes in the 9000 slot.

The author, Brad Templeton is world-famous because of the technique for merging programs from cassette which he invented and refined, and which Jim Butterfield publicised in many journals. Jim's fame as a Pet Guru is too well known to require comment, and the fact that he wrote the manual is a major advantage. There may be some unkind criticism of some of the humour in the manual, but only the Users Guides for Disco-O-Pro and Command-O have anything like the same tutorial content, and the high-quality three-ringed binder in which the product comes in a different league from those normally associated with products in this price range. Templeton's technical appendices are unique, in that they show the advanced programmer exactly how to modify Power and add features. Furthermore, useful routines within Power are identified, and their locations given.

A major advantage which Power possesses over a number of the chips available is that it performs in almost precisely the same way on 8032, 4032, and 3032 machines.

AUTO provides automatic line numbering, and used on its own works from the current last line used in a program, which is very helpful for convenient addition of code at the end of a program.

FIND AND CHANGE: you can specify the range of lines in which this is to take place and the program differentiates between Keywords and other strings. The unique feature is the use of metacharacters. A fullstop matches any character on the same line and the closed square bracket matches the end of a BASIC line. The importance of this is that it makes it possible to look for two significant points regardless of what occurs between them: e.g. IF * THEN , and FOR *NEXT will find these instructions if they occur on the same line.

The ability to continue to SEARCH for a particular string, without having to define it again, merely by hitting @ is very convenient.

INSTANT KEY WORDS: When these are switched on, any shifted character produces keywords in full. This is attractive for poor typists, and offers two advantages over the more familiar abbreviations supplied within the operating system: You can see the words in full, and you do not exceed the normal line length, which can be useful but also slows up editing. The self adhesive transparent key-cap labels provided are unique, and as users of Wordpro will know, very helpful. Considerable care has been taken to make the keys selected as near to mnemonics as possible.

Particularly convenient are the single key access to SCRATCH DSAVE COPY DLOAD CATALOGUE. Unfortunately these commands are available only on the 4000 series version. (Incidentally the numbering of the products is somewhat strange: 3040, 4040, and 8040!)

SCROLL: This is arguably the single most important facility on the chip. It works in almost exactly the same way as the same utility in Command O. You may scroll the program up and down the screen in exactly the same way as you can when using a word processor.

RENUMBER: this is by **four** parameters, enabling subroutines to be made to start at significant line numbers.

MLM jumps into the Monitor without a break and without modifying the Stack Pointer. This is helpful, and also enables you to print Monitor

output to the printer, because you are not cancelling the CMD status as you do with SYS4, OR SYS1024.

INSTANT PHRASES. POWER offers up to 26 phrase printed on the screen when you press a shifted key. Although for each one used, you loose an Instant Keyword of course.

INSTANT SUBROUTINES: up to 26 shifted characters will invoke that many separate programs to coexist peacefully, and use common variables.

Commands like GET and INPUT can be included in Instant Subroutines whereas in direct mode they cannot. It is not necessary to hit the Return key after the shifted character.

The manual shows useful Disk-status-checking and Catalogue-getting routines, and of course numerous magazines contain useful material. Nevertheless, this section of the manual could also be expanded to considerable advantage.

The ability to switch the Instant Keywords, Subroutines, and Phrases off is welcome also.

The unique, and very clever feature of this is that special REMs are used, at the start of the program, so that a BASIC program will continue to run even if Power is not in place, and yet the phrases are available if required.

Both Instant Phrases and Instant Subroutines represent tremendous advances in capability, and I found the manual a little frustrating, because many more examples would be lapped-up by the enthusiastic user. It would be ideal if Jim and Brad could be persuaded to produce some form of Advanced Users Supplement, or to update to the manual, dealing with this and the similar question of the more specialised aspects of using the XEC command to allow the disk unit, or some other peripheral to take over the machine in substitution for the keyboard. It is tantalising to see simple examples. Why not some extremely clever ones too?

The debugging potential of the Power chip is enhanced by virtue of the fact that Power features can be embodied in instant phrases. This has enabled the major design criterion which Brad Templeton had in mind, namely that programs should be transportable and should not be Power-dependent, to be implemented, whilst allowing the programmer full use of this attractive feature.

A major advantage which is available on the 80 column machine only is the command SEL I which gives the ability for an Instant Phrase to be accepted in response to INPUT command. This is really great for debugging, because it implies, provided you hit the right key, that the data you input is identical every time.

WHY: this shows where an error occurs. However, if your line has multiple commands on it, WHY will show exactly where the program stopped.

DUM: dump the variables in the order **in which they were used.**

Unfortunately Power does not dump arrays, due to space constraints. It is true that an instant subroutine could be used to dump specific arrays anyway, but there is a competing German chip which dumps arrays!

TRACE: This is unique in the range of options provided. You can see a listing of the line, and each variable as it is calculated. The Trace can be at the top of the screen or continously down the screen (allowing review of what happened earlier). The stepping options are greater than usual: TAB or = gives a Step, the Space bar gives a continuous Trace and the full-stop runs the program without tracing.

There is a useful display of the result of IF THEN, in that the Boolean value is shown as the variable. This is most helpful in tracing the pathways taken through the program.

Unusually, the manual gives full information on how to trace to printer, although allocating a file number in excess of 127 does not create a line feed if the Ascii printer you are using requires one.

The versatility of the Trace command is enhanced by having special SYS command with parameters enabling you to switch in and out specific types of Traces. This is particularly good in avoiding waste of paper when loops are running, and also makes it easy to by-pass sections of the program which have already been sucessfully debugged. Complex programs can be debugged in stages using a GOSUB to a subroutine for the SYS command switching the Trace on and off.

The manual shows how to avoid the problems of tracing GET commands. FIX will occasionally be very useful for resetting certain essential pointers which may have been clobbered by loading machine code programs into an unusual part of memory. The manual gives a very full explanation of how this command will prevent loss of variables or of the BASIC program.

XEC (execute) this is very powerful and totally unique. Because it is an extension of the technique pioneered by Brad Templeton for merging programs on cassette, there is a danger that it will be seen as merely a rather complicated merging method. This is totally wrong. The sequential file created is read by the computer and treated in every way as if the strings were being typed in at the machine. This implies that they will be acted upon as Direct commands, and their capability is limited only by the fact that the special commands in POWER cannot be included. The manual gives a very tantalizing hint of writing programs which write programs. I would love to see an extended tutorial on this technique added.

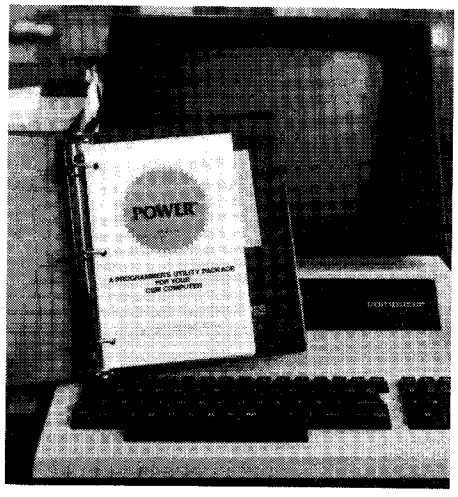
CONCLUSION

Power offers so many facilities, that the user, whether beginner or expert, can grow in capability by using it.

It is worth emphasing the fact that this is no mere run-time improvement to extend BASIC, nor a mixture of such and programmer's aids, but rather a very carefully thought-out set of aids to the serious user of BASIC, designed by a truly expert programmer, to cram as much power as possible into a 4K chip, with as much care taken in choosing what to leave out, as what to include.

The fact that it has been rigorously field-tested, and is a mature product is also relevant.

More power to your elbow



Sorting Techniques

Fulvio Rizzitano

When switching from BASIC to MACHINE-LANGUAGE you might think your time troubles are over. But it isn't so. Not alwasys. Especially when the routine in question is a sort.

I used a M.L. ordinary sort and used to live in happy contentment till the day I had to cope with 5000 elements. Ten minutes are both tiny and a huge amount of time: it all depends on the standpoint.

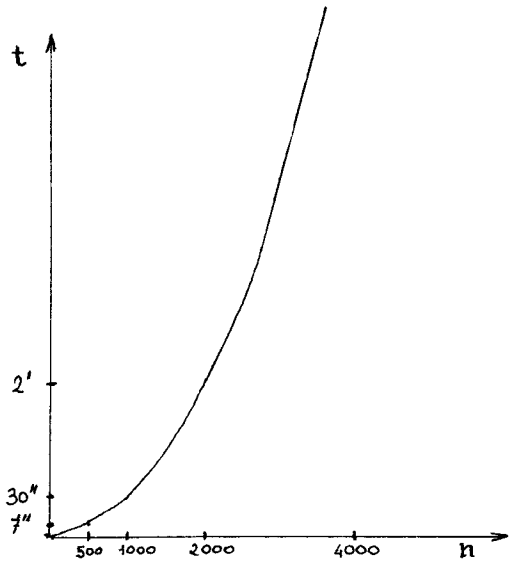
Usual sorts are time consuming because the time needed increases twice as fast as the number of elements..... as you can note on the graph and function figure 1.

To cut down the overall time it is useless to shrink and squeeze only E. A linear trend routine is needed. The SHELL-METZNER for instance whose function is on average: t = E n log 2 n.

By the way, did you ever try to dry-run it? A tough job isn't it? Well, type in the SHELL-DEMO and it will have no secrets any more.

Back to machine-language, what makes the conversion long and quibbling is to tackle the subscripts or pointers (a SHELL-SORT uses up to five pointers) which have to be multiplied by the length of the element and added to the offset each time they are to be used.

But if they are tailored to be absolute addresses in the indirect indexed mode and with a small amount of other modifications (that can be easily seen on the block-diagrams as they are stressed with a darker background) the routine comes out smoothly and loops at vertigo-speed (3000 elements in almost 11 seconds).



ELEMENTS	T I M E
500	7 seconds
1000	30 seconds
2000	2 minutes
4000	8 minutes

$$t = \frac{\xi n^2 + \xi n}{2}$$

t: time
n: number of elements
ξ: amount of time to execute one single routine loop

To understand the coding, remember a floating point number is represented with 5 bytes; the first one being the exponent. If the number is negative then its 2nd byte is greater than 127 - i.e. bit 7 of this second byte is set (bit7 = 1) -

My old routine sorted only the array dimensioned in BASIC before the others, while this new one accepts any array: the VARIABLE-SEARCH SYSTEM-SUBROUTINE is called. - C128 (49451) on BASIC 4.0 - CF6D (53101) on BASIC 2.0 -

If you wish to use it in your programs, just insert the variable name in the BASIC text immediately after the SYS address. You'll get the variable address at locations 68 and 69 on page 0. You may also call the SEARCH-SUB many times on your M.L. Routines. In this case put the names separated by a space after the SYS address of your own routine. Suppose your M.L.R. starts at 32512 and calls the SEARCH-SUB 4 times to get the addresses of A$, B$, U$ and Y$ then the correct syntax to call it from BASIC is:

SYS32512 A$ B$ U$ Y% : REM SPLENDID !!!

When loading integer variables remember the hi-value comes before the lo-value.

To call this sort routine just type: SYS32256 N% A(O) where N% (but it can be any other integer variable) contains the actual number of elements to be sorted *** not the dimension given to A ( ) *** and A(O) (here too, you may use any other floating-point array name) is the 1st element from which our sort begins.

As you can see, you can also sort only one part of your array. Ex.:
1000 N% = 10:SYS32256 N% A(30) : REM SORTS 10 ELEMENTS STARTING FROM THE 30TH.

When loading it remember to lower the TOP OF MEMORY by 512 bytes: the very first BASIC program line should read something like:
10 POKE52,255:POKE53,125

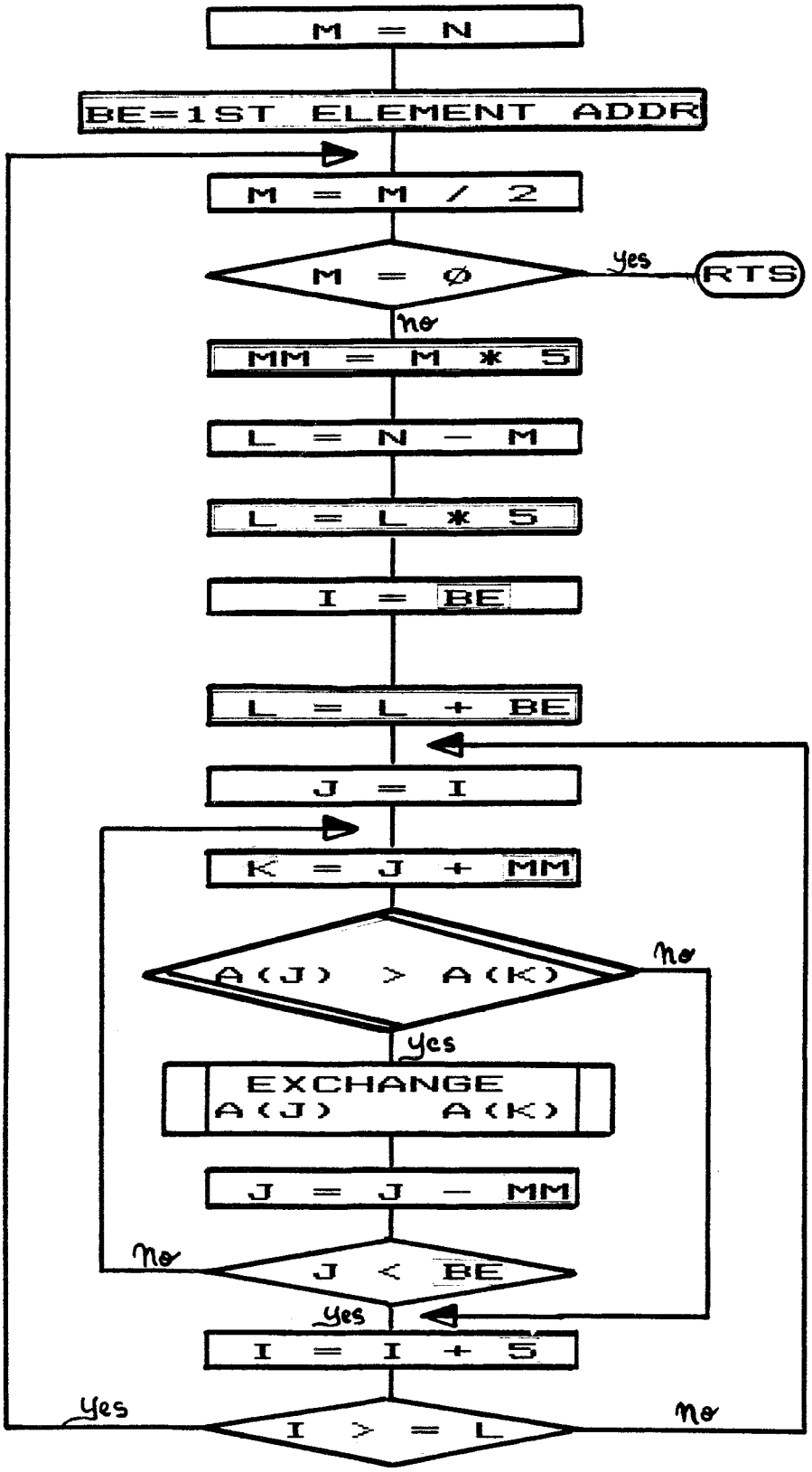
ALPHA-SORT is coming soon.

Fulvio Rizzitano from Italy describes a Shell-Metzner sort in Machine Code Language

NOTE

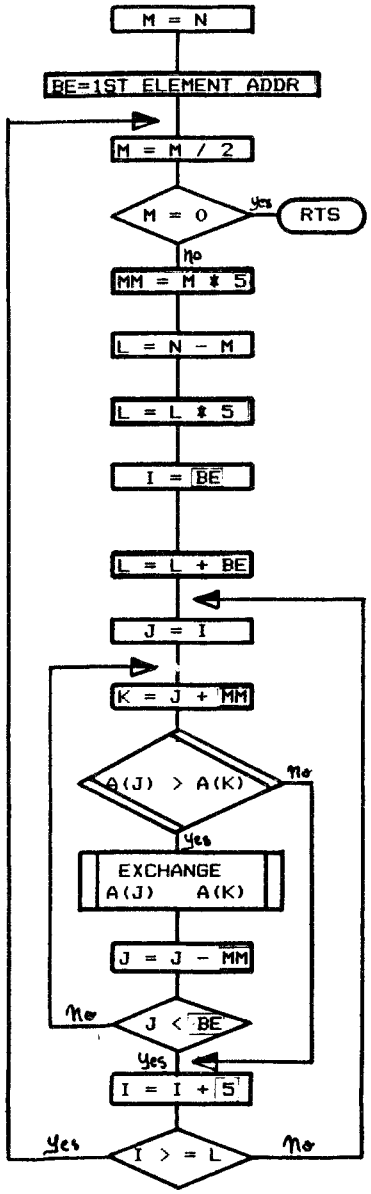
Two different sorting routines are given. The first one for positive numbers only is shorter and runs a bit(!) faster. The second one works with negative elements too.

MODIFIED SHELL-M. SORT TO FIT CBM M.L.



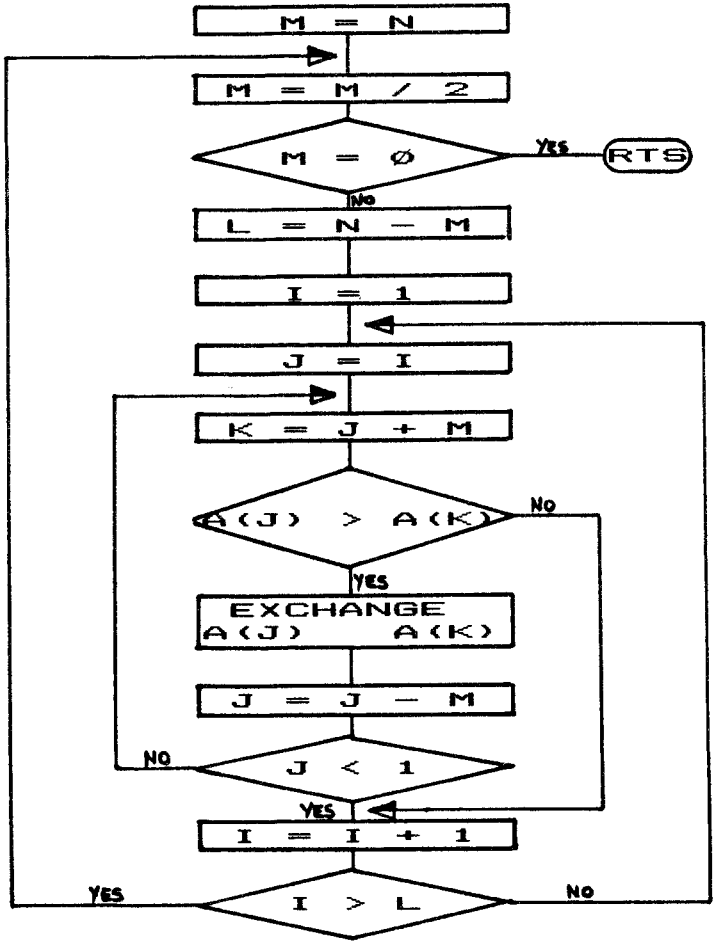
```
1 GOTO10
6 POKE158,0:WAIT158,1:GETX$:POKE158,0:RETURN
10 PRINT"":POKE59468,12:PRINT""
11 PRINT"  S A C E   I N F O R M   S . R . L .": PRINT
12 PRINT"                PRESENTS":PRINT:PRINT
14 PRINT"▀▀▀  ▀   ▀ ▀  ▀  ▀▀   ▀▀▀  ▀▀  ▀▀▀▀ ▀▀▀▀"
15 PRINT"▐▀▀▌ ▐▀▌ ▐▀▌▐▀▌▐▀▌ ▐▀▀▌ ▐▀▀▌ ▐▀▀▌  ▐▀▀▌"
16 PRINT"▐▀▀▌ ▐▀▌ ▐▀▌▐▀▌▐▀▌ ▐▀▀▌ ▐▀▀▌ ▐▀▀▌  ▐▀▀▌"
17 PRINT"▐▀▀▌ ▐▀▌ ▐▀▌▐▀▌▐▀▌ ▐▀▀▌ ▐▀▀▌ ▐▀▀▌  ▐▀▀▌"
18 PRINT"▐▀▀▌ ▐▀▌ ▐▀▌▐▀▌▐▀▌ ▐▀▀▌ ▐▀▀▌ ▐▀▀▌  ▐▀▀▌"
19 PRINT"▐▀▀▌ ▐▀▌ ▐▀▌▐▀▌▐▀▌ ▐▀▀▌ ▐▀▀▌ ▐▀▀▌  ▐▀▀▌":PRINT:PRINT
20 PRINT"▀▀▀▀    DYNAMIC TEACHING SOFTWARE"
25 FORJ=1TO6000:NEXT
30 PRINT"    S  H  E  L  L  -▒D  E▒M  Ó":PRINT:PRINT
32 PRINT"THIS PROGRAM WILL SHOW WHAT HAPPENS WHEN"
34 PRINT"A SHELL-METZNER SORT IS IMPLEMENTED."
36 PRINT"IT WILL PROCEDE STEP BY STEP TO SORT A"
38 PRINT"20 ELEMENTS ARRAY."
40 PRINT"ON THE LEFT, THE CRT WILL DISPLAY THE"
42 PRINT"THE PROGRAM AS WRITTEN IN BASIC."
44 PRINT"AT ANY TIME THE LINE BEING IMPLEMENTED"
46 PRINT"WILL BE REVERSED."
48 PRINT"THE MID-CRT WILL DISPLAY THE CURRENT"
50 PRINT"VALUES OF THE SIMPLE VARIABLES USED."
52 PRINT"ON THE RIGHT, THE CRT WILL DISPLAY"
54 PRINT"THE CURRENT VALUES OF THE ARRAY.":PRINT
56 PRINT"PRESSING ANY KEY WILL TRIGGER OFF THE "
58 PRINT"NEXT STEP."
60 GOSUB6
62 M%=20:N=20:L=0:I=0:J=0:K=0:
64 DIMA(20),A$(20)
66 PRINT"   L I N E S         VARIABLES     ARRAY":PRINT
68 PRINT"1  M%=N        M%: 20    N: 20    A(01):25":A(1)=25
70 A$(2)="2  M%=M%/2 "
72 A$(3)="3  IFM%=0THENEND "
74 A$(4)="4  L=N-M%  "
76 A$(5)="5  I=1     "
78 A$(6)="6  J=I     "
80 A$(7)="7  K=J+M%  "
82 A$(8)="8  IFA(J)<=A(K)THEN12 "
84 A$(9)="9  GOSUB15:REM EXCHANGE"
86 A$(10)="10 J=J-M% "
88 A$(11)="11 IF J >= 1 THEN 7 "
90 A$(12)="12 I=I+1  "
92 A$(13)="13 IF I > L THEN 2  "
94 A$(14)="14 GOTO 6 "
95 FORJ=2TO20:POKE216,J:PRINT
96 PRINTA$(J);TAB(31)"A(";RIGHT$("0"+MID$(STR$(J),2),2)"):";
97 READA(J):PRINTMID$(STR$(A(J)),2):NEXTJ
98 DATA 90,80,70,60,50,25,35,45,55,40,30,20,95,85,75,65,90,15,65,5
99 GOSUB6:POKE216,1:PRINT:PRINT"1  M%=N    "
200 M%=M%/2:POKE216,2:PRINT:PRINT""A$(2)"","  M%:   "M%
210 GOSUB6:POKE216,2:PRINT:PRINTA$(2)
300 POKE216,3:PRINT:PRINT""A$(3)"":POKE216,1:PRINT:PRINTTAB(13)"
310 GOSUB6:POKE216,3:PRINT:PRINTA$(3)
320 IFM%=0THEN3000
400 L=N-M%
410 POKE216,4:PRINT:PRINT""A$(4)"","   L:   "L
420 GOSUB6:POKE216,4:PRINT:PRINTA$(4)
500 I=1:POKE216,12:PRINT:PRINTTAB(20)"          "
510 POKE216,5:PRINT:PRINT""A$(5)"","   I: 1"
520 GOSUB6:POKE216,5:PRINT:PRINTA$(5)
600 J=I:POKE216,10:PRINT:PRINTTAB(20)"          "
610 POKE216,6:PRINT:PRINT""A$(6)"","   J:   "J
620 GOSUB6:POKE216,6:PRINT:PRINTA$(6)
700 K=J+M%:POKE216,7:PRINT:PRINT""A$(7)"","   K:   "K
710 K=J+M%
720 GOSUB6:POKE216,7:PRINT:PRINTA$(7)
800 POKE216,8:PRINT:PRINT""A$(8)""
810 GOSUB2000
820 POKE216,8:PRINT:PRINTA$(8)
830 IFA(J)<=A(K)THEN1200
900 S=A(J):A(J)=A(K):A(K)=S
910 POKE216,9:PRINT:PRINT""A$(9)""
920 GOSUB2000
930 POKE216,9:PRINT:PRINTA$(9)
1000 J=J-M%:POKE216,6:PRINT:PRINTTAB(20)"          "
1010 POKE216,10:PRINT:PRINT""A$(10)"","   J:   "J
1020 GOSUB6:POKE216,10:PRINT:PRINTA$(10)
1100 POKE216,11:PRINT:PRINT""A$(11)""
1110 GOSUB6:POKE216,11:PRINT:PRINTA$(11)
1120 IFJ>=1THEN700
1200 I=I+1:POKE216,5:PRINT:PRINTTAB(20)"          "
1210 POKE216,12:PRINT:PRINT""A$(12)"","   I:   "I
1220 GOSUB6:POKE216,12:PRINT:PRINTA$(12)
1300 POKE216,13:PRINT:PRINT""A$(13)""
1310 GOSUB6:POKE216,13:PRINT:PRINTA$(13)
1320 IFI>LTHEN200
1400 POKE216,14:PRINT:PRINT""A$(14)""
1410 GOSUB6:POKE216,14:PRINT:PRINTA$(14)
1420 GOTO600
2000 POKE216,J:PRINT:PRINTTAB(37)""MID$(STR$(A(J)),2)""
2010 POKE216,K:PRINT:PRINTTAB(37)""MID$(STR$(A(K)),2)""
2020 GOSUB6
2030 POKE216,J:PRINT:PRINTTAB(37)MID$(STR$(A(J)),2)
2040 POKE216,K:PRINT:PRINTTAB(37)MID$(STR$(A(K)),2)
2050 RETURN
3000 POKE216,18:PRINT:PRINT"    E   N   D    "
3010 END
READY.
```

MODIFIED SHELL-METZNER SORT TO FIT CBM MACHINE LANGUAGE



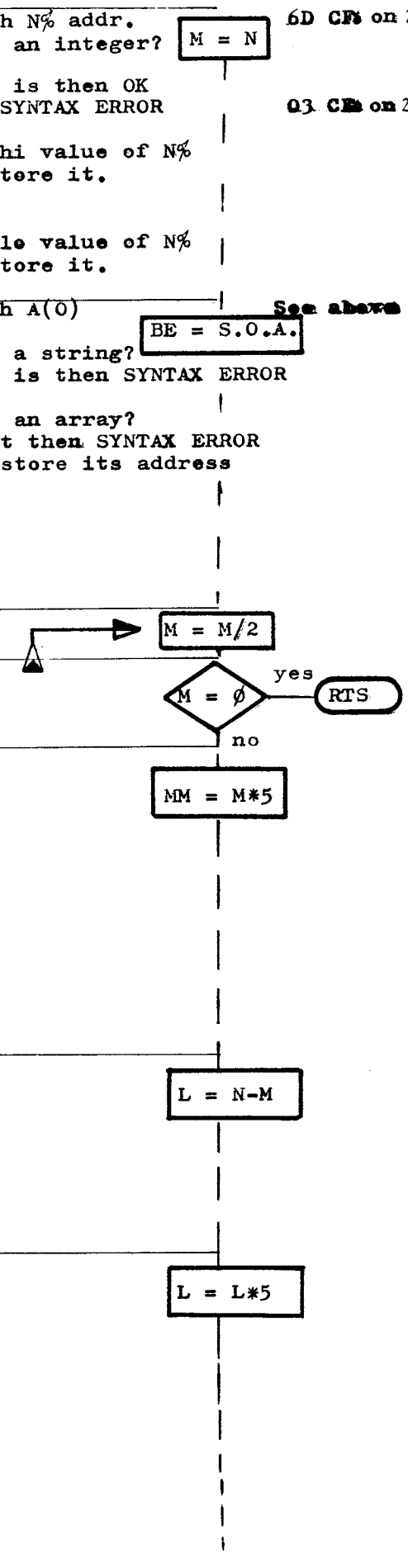
VARIAB.	DESCRIPTION	ADDRESS
N :	NUMBER OF ELEMENTS	00 01
M :	HALVED INDEX	206 207
I :	INCR. ADDRESSING INDEX	96 97
J :	ADDR OF 1ST COMPAR. ELEM.	98 99
K :	ADDR OF 2ND COMPAR. ELEM	102 103
L :	UPPER LIMIT	104 105
MM:	ACTUAL LENGTH OF M	106 107
BE:	ADDR OF 1ST ARRAY ELEMENT	108 109
A()	ARRAY TO BE SORTED	

SHELL-METZNER SORT IN BASIC



BASIC 4.0 SORT FOR POSITIVE ELEMENTS BASIC 2.0

```
7F00  32512   JSR        2B C1     49451 < ; Search N% addr.
7F03  32515   LDA.PO     08 (  8)          ; Is it an integer?
7F05  32517   CMP.IMM    80 (128)
7F07  32519   BEQ        03      > 32524   ; If it is then OK
7F09—32521    JMP        00 BF     48896 < ; else SYNTAX ERROR
7F0C—32524    LDY.IMM    00 (  0)
7F0E  32526   LDA.IND+Y  44 ( 68)          ; Load hi value of N%
7F10  32528   STA.PO     01 (  1)          ; and store it.
7F12  32530   STA.PO     CF (207)
7F14  32532   INY
7F15  32533   LDA.IND+Y  44 ( 68)          ; Load lo value of N%
7F17  32535   STA.PO     00 (  0)          ; and store it.
7F19  32537   STA.PO     CE (206)
7F1B  32539   JSR        2B C1     49451 < ; Search A(0)
7F1E  32542   LDA.PO     07 (  7)
7F20  32544   CMP.IMM    FF (255)          ; Is it a string?
7F22  32546   BEQ        E5      > 32521   ; If it is then SYNTAX ERROR
7F24  32548   LDA.PO     45 ( 69)
7F26  32550   CMP.PO     2D ( 45)          ; Is it an array?
7F28  32552   BCC        DF      > 32521   ; If not then SYNTAX ERROR
7F2A  32554   STA.PO     6D (109)          ; else store its address
7F2C  32556   BNE        06      > 32564
7F2E  32558   LDA.PO     2C ( 44)
7F30  32560   CMP.PO     44 ( 68)
7F32  32562   BCS        D5      > 32521
7F34—32564    LDA.PO     44 ( 68)
7F36  32566   STA.PO     6C (108)
7F38—32568    LSR.PO     CF (207)
7F3A  32570   ROR.PO     CE (206)
7F3C  32572   BNE        05      > 32579
7F3E  32574   LDA.PO     CF (207)
7F40  32576   BNE        01      > 32579
7F42  32578   RTS
7F43—32579    LDA.PO     CF (207)
7F45  32581   STA.PO     6B (107)
7F47  32583   LDA.PO     CE (206)
7F49  32585   STA.PO     6A (106)
7F4B  32587   ASL.PO     6A (106)
7F4D  32589   ROL.PO     6B (107)
7F4F  32591   ASL.PO     6A (106)
7F51  32593   ROL.PO     6B (107)
7F53  32595   CLC
7F54  32596   ADC.PO     6A (106)
7F56  32598   STA.PO     6A (106)
7F58  32600   LDA.PO     CF (207)
7F5A  32602   ADC.PO     6B (107)
7F5C  32604   STA.PO     6B (107)
7F5E  32606   LDA.PO     00 (  0)
7F60  32608   SEC
7F61  32609   SBC.PO     CE (206)
7F63  32611   STA.PO     68 (104)
7F65  32613   TAX
7F66  32614   LDA.PO     01 (  1)
7F68  32616   SBC.PO     CF (207)
7F6A  32618   STA.PO     69 (105)
7F6C  32620   TAY
7F6D  32621   ASL.PO     68 (104)
7F6F  32623   ROL.PO     69 (105)
7F71  32625   ASL.PO     68 (104)
7F73  32627   ROL.PO     69 (105)
7F75  32629   TXA
7F76  32630   CLC
7F77  32631   ADC.PO     68 (104)
7F79  32633   STA.PO     68 (104)
7F7B  32635   TYA
7F7C  32636   ADC.PO     69 (105)
```



Communications Editorial

Judging by the reaction to the communications feature in issue 7 of the newsletter, this is an area of growing awareness in the PET world. Many of you expressed great interest in the article, and were wondering if (or more usually when) we'd be seeing another.

Well, Dr. Barker (of "Using a microcomputer as an interactive terminal" fame - the main communications article in issue 7) has done it again . This time an article entitled "Alogorithms for Intelligent Terminal Operation", in which he describes alogorithms to enable the transmission of files between a microcomputer and a mainframe. He gives illustrative implementations of these algorithms, and also describes some applications of file transfer operations. Written in his usual clear style, complete with sample listings, flowcharts, diagrams etc. the article is extremely easy to follow for both beginner and the expert, and should provide many ideas of the way in which the usefulness of a PET could be expanded.

Privileged Information

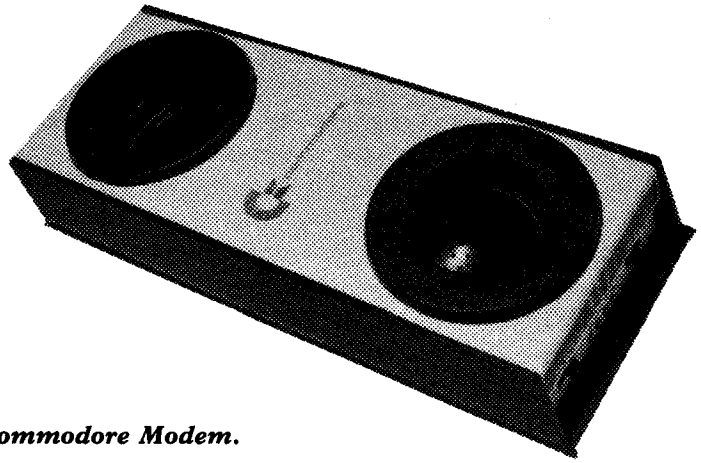
I haven't as yet in the two communications specials reproduced anything from other overseas publications, although I have in the main body of the newsletter. This is due to the excellence of the articles from Dr. Barker, and the fact that I haven't come across anything from overseas to beat them. However, to move away from communications for a while, I'd like to say a few words about this "lifting" of information from other publications.

I'm in the privileged position of receiving information and magazines from many different Commodore magazines and periodicals from all over the world. All of them contain items of interest, and it is tempting to reproduce far more than I do, but to do so would make this particular newsletter something like the size of War and Peace - in large print to boot. On the other hand, if I reproduce all the technical bits (of great interest agreed) the newsletter would be so dry and de-humanised that to read it would be a chore and a bore. Consequently you will find the occasional humourous piece in here as well.

Now some people complain about this. Fine, but to recompense some people also praise material like that, on the grounds that it does break up all the 'heavy' material. I long ago learnt that you can't please all of the people all of the time!

Feedback

Some of you may already subscribe to these overseas publications, and as I said earlier for that I apologise. Perhaps even with people like that I'd be bringing something to your attention that you might previously have overlooked. If any of you have strong feelings about that please write and let me know - feedback is very valuable.



The Commodore Modem.

But back to communications. As well as the Dr. Barker article we have a number of contributions in the main body of the newsletter also concerned with this field. In particular three items on the Commodore 8010 Modem, by well known people in the PET field, which should save a lot of headaches for those of you starting up in that area.

I think you'll see by reading these articles just what sort of thing can be done. As with the newsletter editorial, it might be an interesting idea to take a brief glance at a possible future.

The Future

And what of the future ? Talking about micros in general, and communications in particular, Interface, the American equivalent of the newsletter, published an interview with Jack Nilles, Director of Interdisciplinary Programs at the University of Southern California. Jack had this to say.

"We did a study on a topic I call 'Telecommuting', which is the substitution of telecommunication and computers for the commute to work, working at home or near homes using computer technology. Computers at their present stage of evolution are still pretty dumb beasts. They do well-defined things very quickly. That's mainly what they're about. As long as they are told what you want to have done, step-by-step, and often in agonizing detail, they will do it faithfully. They do dull, routine jobs very fast. Primarily that has been their use in business, science, whatever. Imagine the 19th century job of a clerk in a large store, sitting at a desk all day performing a specific function.

"Now a machine can go through all that information and ZIP!. you have it all together. That's the kind of thing they're good for. In scientific applications, they do tedious calculations that you would never be able to do by hand. And with a calculator you'd wear your fingers out. Combined with telecommunications technology, computers can do these routine jobs independent of their location."

Algorithms for Intelligent Terminal Operation

Introduuction

The use of microcomputers in industry, business, education and the home is increasing at a significant rate. In order to satisfy the wide range of consumer demands, many different kinds of end-user orientated systems are now becoming commercially available. Prices vary considerably both with the hardware/software facilities that are provided and with the type of market at which individual products are aimed. In terms of cost benefit, the attractiveness of a microcomputer system increases substantially if it can also be used as an intelligent terminal device (Ber81.)

Fundamental to this mode of operation are facilities that provide the microcomputer with the capability of being attached to some other larger computer configuration - called a host system. To achieve this type of interconnection suitable modems and interfaces are necessary (Bar81a). Through these, the micromputer will be capable of communicating with

a) a remote or local mainframe/minicomputer,
b) a local network of other intelligent terminals, or,
c) a generalised geographically distributed computer network.

In addition, the microcomputer itself may also be capable of acting as a host to other units that are able to interconnect with it in an appropriate manner.

Once attached to a host system there are many ways in which an intelligent terminal can contribute to and utilise the available resources. Three of the more important of these are

a) the initiation of computational processes within the host system,
b) the support of certain processes delegated in it by the host and,
c) participation in file transfer activity.

As a consquence of these three basic operations many new types of man-machine interaction become possible. For example, by means of an intelligent terminal, a network user is able to construct a program locally, transmit it to a remote processor and then initiate its compilation. Alternatively, such a user may occupy a source program from a remote node to a local terminal, edit it in various ways and then send it to a second remote node for compilation. Subsequently, the final program may then be brought back to the local terminal and cross-loaded into a robot-like device in order to control an automated assembly line.

In a similar way, an intelligent microcomputer based controller might be delegated the responsibility of monitoring and performing local control of some manufacturing process. At particular items in its operating sequence this controller might 'attach' itself to an appropriate host system, transmit data to it and, in exchange receive coded control strategies thereby enabling the manufacturing process to be dynamically modified.

As well as the computational processes involved, each of the above examples depends critically upon the flow of information between two or more computers. Usually, information transfer between an intelligent device and a host system takes place via suitable structured message strings or involves some form of file transfer. In this paper we consider some of the applications and problems of using a microcomputer as an intelligent terminal device capable of participating in file transfer operations with a host computer system.

File transfer - general considerations

Over the last few years there has been growing interest in the development of distributed computer systems. Usually, these consist of a series of processing nodes interconnected by suitable communication links. Nodes in the network community are able to communicate with each other by means of a variety of message passing techniques. A message is essentially a contiguous sequence of symbols. When transmitted between one entity and another messages usually invoke some form of action or response on the part of their recipient. For example, if a terminal user sends the message

LIST JACK

to a remote computer (the recipient), then, provided the object JACK (a file) exists, its contents would be listed - unless there was any form of access control in operation.

LIST JACK

is an example of a message string. The effect of a message will depend upon its information content and the nature of the rules of interpretation built into its recipient. Messages usually have only a transient existence and are fairly short in length.

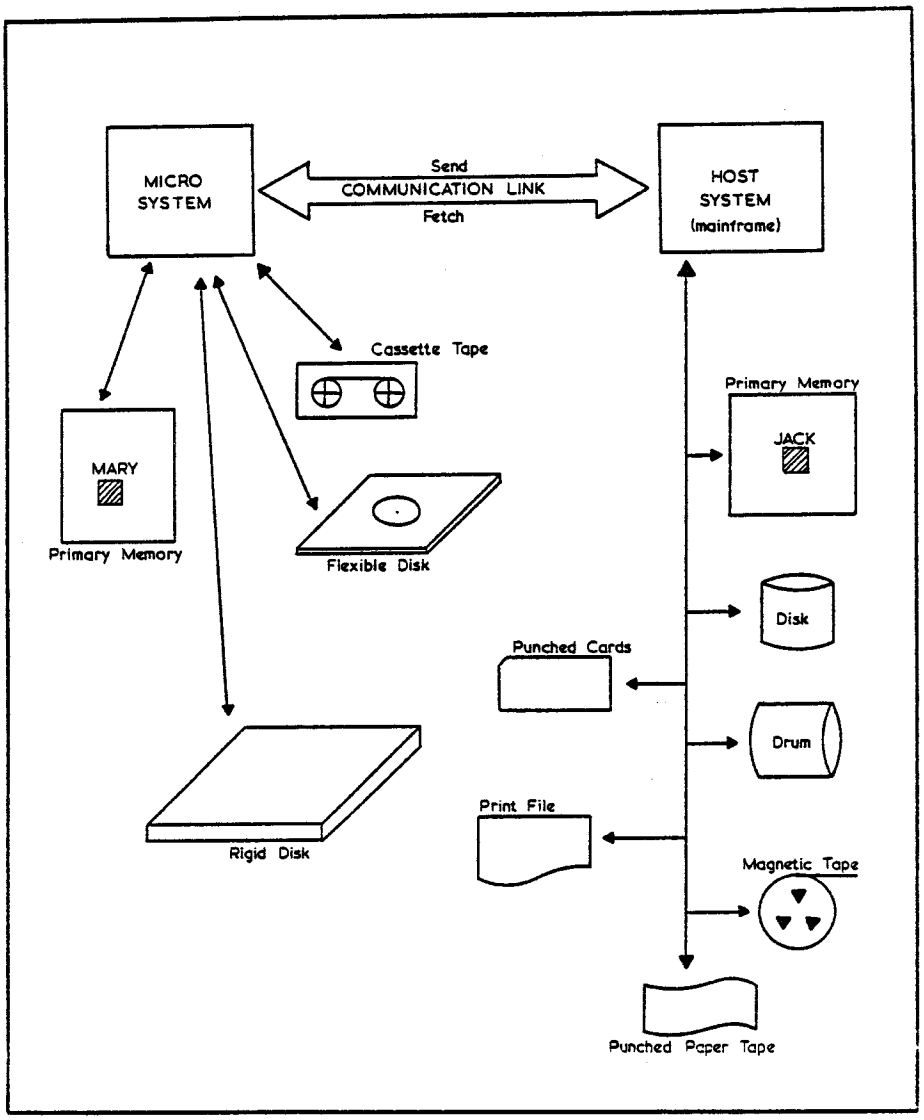
In addition to message transfer most distributed systems permit files of data/information to be transmitted between nodes. Like a message a file may be regarded as a contiguous sequence of characters. However, a file is a much more complex entity than a message. Unlike the latter it is usually more highly structured, has a greater physical volume, contains far more information and has a much longer lifetime. In order to cater for this latter characteristic a variety of storage media exist. These enable files of information to be retained within a computer system for an indefinite period of time. The relationship between some of the different types of storage media and the file transfer processes in which they are likely to beocome involved is depicted in figure 1.

When transferring information according to the strategy suggested in this diagram a number of important factors need to be considered. Some of these are briefely outlined below.

Direction of Transfer

File transfer between an intelligent terminal device and a host system will usually need to be bidirectional. This means that each of the communicating nodes must be able to send or fetch files over the communication link. For the simplest type of application it is unlikely that the link will need to support

FIGURE I FILE TRANSFER PROCESS - MEDIA CONSIDERATIONS



simultaneous transfer in both directions. However, many situations do arise in which this is a necessary requirement.

Media Considerations

A variety of different storage devices will usually be available - both at the intelligent terminal and within the host system. Mechanisms must therefore exist to enable the complete mobility of files across the various support media - for storage purposes if not for processing. A typical transfer operation might involve movement of a tape cassette file from the micro to the host system for storage on an archive tape. Similarly, another useful operation might be that of copying a disk/drum resident file from the host to the intelligent terminal. Here it might be stored either within primary memory or as a floppy disk file. In principle, when file transfer takes place all possible combinations of source and destination devices need to be supported.

Transfer Time

This consideration is important in situations where (a) the communications link is being costed on a line utilisation basis (for example, a dial-up line), and (b) where file transfer is being used to service some form of real-time man-machine dialogue. The amount of time that it takes to transfer a file will depend upon many factors such as a line speed, file size, error rate, processor power and peripheral speed.

Error Control

When messages and files are transferred between network nodes it is important that they do not become perturbed by the transmission channels that are used.

Perturbations usually introduce noise into the information which means that the file/message received at the destination node differs from that dispatched by the source. Adequate software/hardware protection must thus be built into the file transmission system in order to ensure that all errors are detected and, if possible automatically corrected without the need for excessive retransmission of information.

Logical File Structure

As was mentioned earlier, it is extremely likely that when two or more procesing nodes are involved in file transfer each will support different types of physical file structure (media considerations). In addition, each computer system may necessitate the use of a different logical file organisation.

The simplest type of logical file structure is one in which the file is regarded as being a continuous collection of characters similar to that shown in figure a.

In this type of situation, any higher level structure is super-imposed upon the file by the various software items that process it. An alternative arrangement in which some form of record structure is imposed on the file is illustrated in figure b.

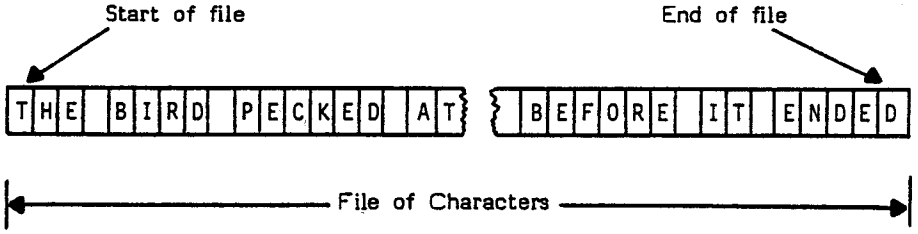


Figure a

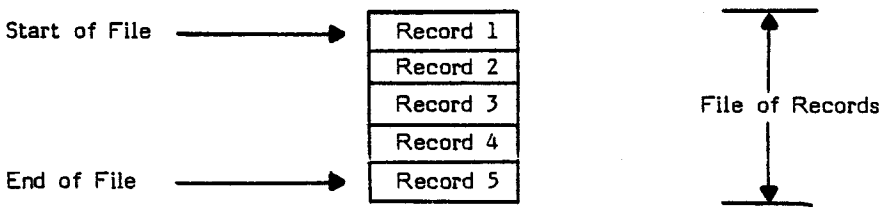


Figure b

Each of the above represent simple approaches to logical file organisation. Many other, more complex, arrangements could be formulated (Wie77).

However, for the following discussion, the models presented above will be sufficient.

Algorithm Design

In designing algorithms for file transfer operations it is imperative to consider the effects of all the above factors. Furthermore, because of the intrinsic difference between each of the processing nodes and between their storage peripherals the algorithms may need to incorporate suitable conversation rules. The complexity of these algorithms may need to incorporate suitable conversation rules. The complexity of these will depend upon the nature of both the intelligent terminal and the host system. Fortunately, in many situations the development of virtual memory computers and machine independent filing systems greatly minimises the amount of effort that needs to be devoted to file conversion problems. However, it is important to bear in mind that as far as file processing activity is concerned some special rules may still need to be observed (Jud73).

File transfer - a case study

In the remaining sections of this paper we consider the process of file transfer in a system in which a mainframe computer, acting as a host, services the file activity associated with an intelligent terminal device. For the purposes of illustration a 32K Commodore PET is used as the intelligent terminal. This communicates with a remote IBM 370/168 over the public switched network. A detailed description of the hardware configuration has been given elsewhere (Bar81).

In general, BASIC is used as the high level language for the implementation of alogorithms in the microcomputer. However, in situations where speed improvement is necessary, machine code could just as easily be used. Before discussing the details of the algorithms a brief description of the file structures used on the mainframe and the microcomputer is a necessary pre-requisite.

File Structure on the Mainframe

Files that are resident on the mainframe may be regarded as collections of records each of which consists of a contiguous sequence of 8-bytes. Individual files may contain either fixed or variable length records. These may be of any non-zero length up to a maximum of 32,767 bytes. Particular records within a file may be uniquely identified by means of their associated record number (R) whose value lies in the range -99999.999 through 99999.999.

File Structure on the Microcomputer

The mainframe file structure may be easily 'modelled' on the microcomputer by means of a BASIC character string array. Essentially, each mainframe record is represented by one or more elements of the array. Storage for a file can thus be allocated by a statement of the form

```
10 DIM L$(100)
```

which reserves memory storgae for a file containing 100 records. There is a limitation on the length of these records since they cannot exceed 255 bytes. Records longer than this would need to be modelled by a two dimensional character array. Thus, a record of L bytes could be segmented into CEIL(L/255) sub-records of maximum length 255. These could then be stored is such a way that one of the subscripts of an array reference would identify a particular record while the other subscript would identify the required segment within that record. For example, L$(2,4) would reference the second 255 byte segment of the fourth record in the file.

Notice that the CEIL function introduced above is defined in such a way that the value of CEIL(A) is equal to A if A is integer, otherwise, it is equal to the smallest integer that is larger than A.

Depending upon the memory size of the microcomputer there would be a limit placed on the number of records that could be accommodated. Based upon the way in which character string arrays are stored in the PET (Don80), it can be shown that, for a one-dimensional array of K elements the memory space required would be

$$M = 7+(K+1)*3 + \sum_{I=1}^{K} LEN(L\$(I))$$

Assuming that all records would be 255 bytes long, the memory space available on a 32K PET would limit the value of K to about 120. However, for many applications the record lengths are not likely to exceed 80 characters. Where this is the case the number of records that could be handled increases to about 370. Files having a greater capacity than this would need to be off-loaded to disk or tape storage.

In the discussion that follows we consider file transfer from mainframe to micro and from micro to mainframe. In all cases, transfer to or from the mainframe takes place via a one-dimensional character string array created by a BASIC program running on the PET.

File transfer from Mainframe to Micro

When transferring data from a file system to a target microcomputer these are two general cases to consider. These differ according to whether the information that is transferred to the micro is,

a) retained in its primary memory area, or,
b) transfered to its secondary storage system.

The first of thes situations arises in cases where the mainframe computer is used either,

a1) to achieve memory images from the micro, or,
a2) to develop programs for subsequent execution in the intelligent terminal device.

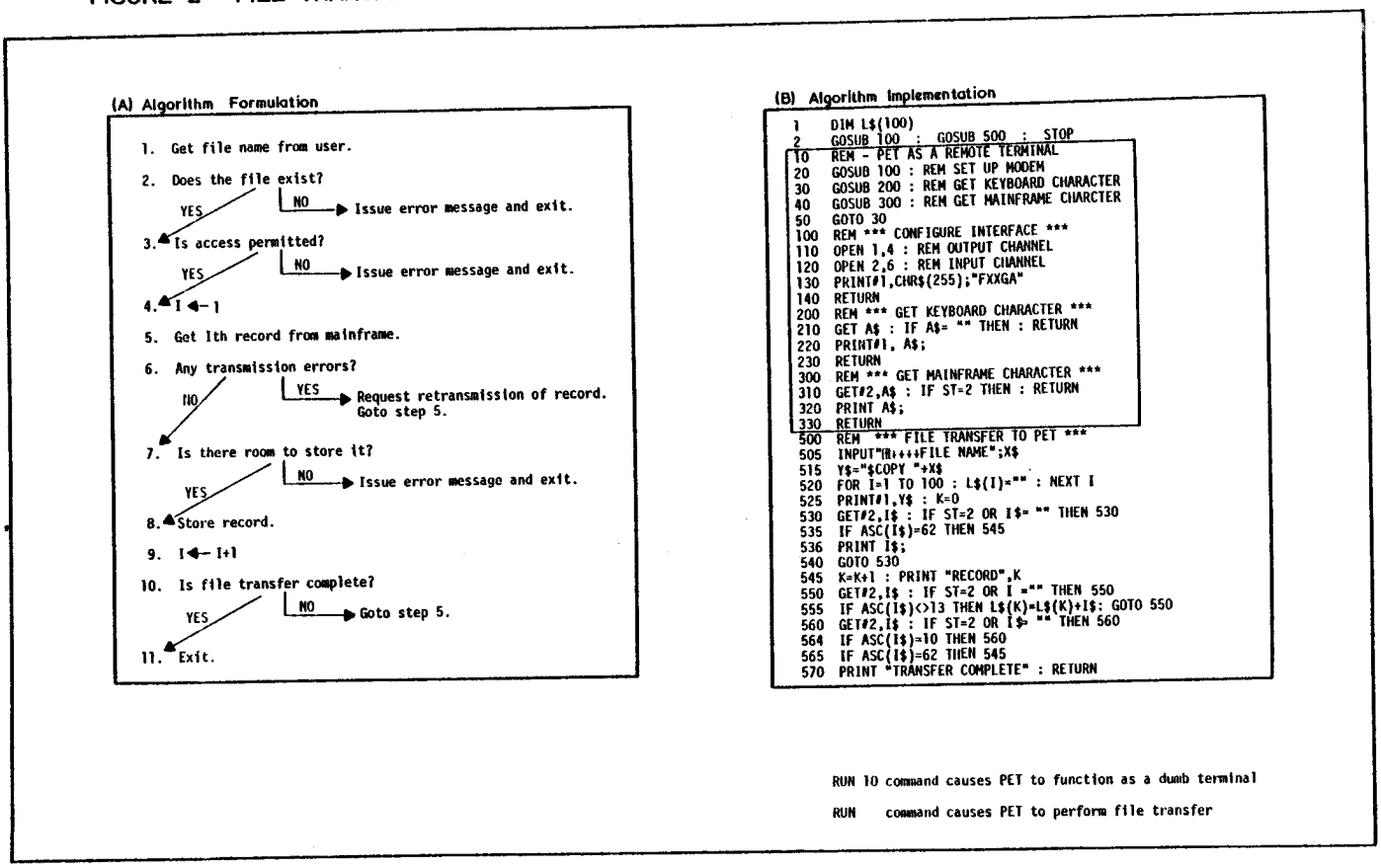
Examples of the second situation, (b), arise when files on the mainframe (which exceed the micro's primary storage capacity) are to be transferred to the intelligent terminal for local processing. Each of these will be described in turn.

Algorithm for File Transfer to Primary Storage

When a file is to be transferred to the micro, the software that it contains must perform three basic operations. First, it must send an appropriate message to the mainframe in order to initiate file transfer. Then, as records are received, it must validate them and request retransmission - if they are found to contain any errors. Finally, each error-free record must be stored in an appropriate position within the memory space. The various steps that are involved are depicted in the algorithm shown in figure 2A

An implementation of the algorithm is presented in figure 2B.

FIGURE 2 FILE TRANSFER FROM MAINFRAME TO MICROCOMPUTER MEMORY



In order to simplify the discussion certain basic assumptions have been made. It has been assumed that the file to be copied exists and that the terminal user has access to it. Because of memory space limitations there are certain restrictions placed upon the size of the file that is to be copied - the file must not contain more than 100 records whose length must not exceed 255 bytes. Finally, for simplicity, it has been assumed that records will be transferred without any perturbation by the communication link.

In the listing presented in figure 2B, lines 10 through 330 are responsible for operating the microcomputer as a terminal device (Bar81a). The subroutine defined in lines 500 through 570 is responsible for the file transfer. The name of the file to be transferred is input at statement 505 and the copy process is initiated by the command message sent to the mainframe via the print statement in line 525. Each record transmitted to the microcomputer is preceeded by a start of record character (ASCII value 62) and terminated by a carriage return/line feed combination (ASCII values of 13 and 10, respecitvely).

When using the file transfer routine shown in figure 2B it is the user's responsibility to ensure that the assumptions listed above are not violated. If this is not possible, appropriate modifications to the program would need to be made.

Algorithm for File Transfer to Secondary Storage

The simplest strategy for transferring a file to secondary storage involves a block by block transfer mechanism. Such a scheme is embodied in the algorithm depicted in figure 3A. The transfer loop involves two basic steps. First, a block of records is transmitted to the local storage device. This process is repeated until the whole file has been passed across to the peripheral being used for its storage. A minimal implementation of the algorithm is presented in figure 3B.

The underlying principle upon which the subroutine depends is the same as that which was employed in the implementation of the previous file transfer process (figure 2B). However, instead of sending a single copy message to the mainframe (to initiate the transfer of the whole file), a sequence of messages of the form

COPY filename(S,F)

is used. Each of these, with the possible exception of the last, copies across a segment of the file containing M records (M=FS+1). In this expression, S and F represent, respectively, the start and finish record numbers within a segment. The values of S and F will depend upon the block size, M, and assume that records in the original file are numbered sequentially starting from unity. The series of values of S and F are thus,

Si = 1, M+1, 2M+1, 3M+1, ...

Fi = M, 2M, 3M, 4M, ...

The code shown in figure 3B performs no error checking - either of transmitted data or of user input from the teminal. These are refinements that could be added in a more detailed implementation. Notice also that the subroutine depends upon the provision of appropriate peripheral support routines to handle the secondary storage device(s) to which a file is being transferred. The first of these (line 455 - GOSUB 1000) is responsible for opening the local file on the external device. The second (line 585 - GOSUB 1100) is delegated the task of writing the data blocks onto the chosen peripheral while the third (line 630 - GOSUB 1200) performs all the housekeeping activities associated with closing the local file when transfer is complete.

The subroutine is figure 3B has been used to transfer mainframe files across to both tape cassette (Don80) and a flexible disk (CBM80). In the latter case a standard Commodore Model 3040 twin floppy disk unit was used. Details of the peripheral sup-

FIGURE 3 FILE TRANSFER TO MICROCOMPUTER SECONDARY STORAGE

(A) Algorithm Formulation

1. Get mainframe file name from user.
2. Get local file name from user.
3. Get block size from user.
4. I ← 1
5. Get Ith block from mainframe.
6. Write Ith block to secondary storage on microcomputer.
7. I ← I+1
8. Is file transfer complete?
   - YES → 9. Close local file.
   - NO → Goto step 5.
9. Close local file.
10. Exit.

(B) Algorithm Implementation

```
1  DIM L$(100)  :  REM BUFFER STORAGE
2  GOSUB 100 :     REM CONFIGURE INTERFACE
3  GOSUB 400 :     REM PERFORM FILE TRANSFER
4  STOP
10 REM  :  PET AS A REMOTE TERMINAL
20 GOSUB 100 :     REM SET UP MODEM
                   o
                   o
                   o
400  REM FILE TRANSFER TO PET WITH
410  REM OUTPUT TO SECONDARY STORAGE
420  INPUT"{4 SPACE}FILE TO BE TRANSFERRED"; X$
430  INPUT"{3 SPACE}LOCAL FILE NAME"; Z$
440  INPUT"{3 SPACE}BLOCKSIZE";M
450  IF M>100 THEN PRINT"+BLOCKSIZE TOO BIG" : GOTO 440
455  GOSUB 1000 : REM OPEN FILE ON SECONDARY STORAGE DEVICE
460  S%=1  : F%=M : N=0
465  K=0
466  FOR I=1 TO M : L$(I)="" : NEXT I
470  S$=MID$(STR$(S%),2)
475  F$=MID$(STR$(F%),2)
480  Y$="$COPY " +X$+ "("+S$+","+F$+")"
490  PRINT#1,Y$
500  GET#2,I$ : IF ST=2 OR I$="" THEN 500
510  IF ASC(I$)=62 THEN 530
520  PRINT I$; : GOTO 500
530  N=N+1 : K=K+1 : PRINT "RECORD",N
540  GET#2,I$ : IF ST=2 OR I$="" THEN 540
550  IF ASC(I$)<>13 THEN L$(K)=L$(K)+I$ : GOTO 540
560  GET#2,I$ : IF ST=2 OR I$="" THEN 560
570  IF ASC(I$)=10 THEN 560
580  IF ASC(I$)=62 THEN 530
585  GOSUB 1100  : REM WRITE BLOCK TO SECONDARY STORE
590  IF K<M THEN 620
600  S%=S%+M : F%=F%+M
610  GOTO 465
620  PRINT "TRANSFER COMPLETE"
630  GOSUB 1200  :  REM CLOSE LOCAL FILE
640  RETURN
1000
1010   Support routines for secondary storage devices
  etc.
```

FIGURE 4 FILE TRANSFER FROM MICRO TO MAINFRAME

A - Algorithm Formulation

B - Algorithm Implementation

A

1. Get file name from user.
2. Does the file exist?
   - YES → 3. Is the file to be over-written?
   - NO → Create it and goto step 6.
3. Is the file to be over-written?
   - NO → 4. Is the file to be extended?
   - YES → Empty it and goto step 6.
4. Is the file to be extended?
   - NO → 5. Assume a new file is required and goto step 1.
   - YES → Goto step 6.
5. Assume a new file is required and goto step 1.
6. I ← 1
7. Send Ith record to mainframe.
8. Wait for answerback prompt from mainframe.
9. I ← I+1
10. All records sent?
    - YES → 11. Close mainframe file.
    - NO → Goto step 7.
11. Close mainframe file.
12. Exit.

B

```
600  REM FILE TRANSFER TO MAINFRAME
605  N=4
610  DIM R$(10)
615  INPUT"{4 SPACE}FILE NAME"; X$
620  FOR I = 1 TO 10 : R$(I)="" : NEXT I
625  Y$="$CREATE "+X$
630  PRINT#1,Y$ : K=1
635  GET#2,I$ : IF ST=2 OR I$="" THEN 635
640  REM PRINT I$;
645  IF I$= "#" AND K=N THEN 660
650  IF ASC(I$)<>13 THEN R$(K)=R$(K)+I$ : GOTO 635
655  K=K+1 : GOTO 635
660  IF MID$(R$(N-1),2,5)="# FIL" THEN 750
665  PRINT "FILE "+X$+" ALREADY EXISTS"
670  PRINT "{DO YOU WANT TO"
675  PRINT"      1. OVERWRITE ITS CONTENTS?"
680  PRINT"      2. CREATE A NEW FILE?"
685  PRINT"      3. EXTEND THE FILE?"
690  PRINT"{ENTER 1, 2 OR 3"
695  GET I$ : IF I$="" THEN 695
700  IF I$="1" OR I$="2" OR I$="3" THEN 710
705  GOTO 695
710  IF I$="1" THEN 725
715  IF I$="3" THEN 755
720  N=3 ; GOTO 615
725  PRINT#1,"$EMPTY "+X$+" OK" : K=0
730  GET#2,I$ : IF ST=2 OR I$= "" THEN 730
735  PRINT I$; : IF I$="#" AND K=2 THEN 755
740  IF ASC(I$)<>13 THEN R$(K)=R$(K)+I$ : GOTO 730
745  K=K+1  :  GOTO 730
750  PRINT"FILE "+X$+"HAS BEEN CREATED"
755  REM NOW TRANSFER THE L$ ARRAY TO THE MAINFRAME
760  PRINT#1, "%ECHO=OFF" : Z$="(LAST+1)"
765  GET#2,I$ : IF ST=2 OR I$="" THEN 765
770  PRINT I$; : IF I$="#" THEN 780
775  GOTO 765
780  PRINT#1,"$COPY *SOURCE* TO "+X$+Z$ ; K=1
785  GET#2,I$ : IF ST=2 OR I$="" THEN 785
790  PRINT I$; : IF I$=">" THEN 800
795  GOTO 785
800  PRINT#1,L$(K)
805  PRINT "RECORD",K : K=K+1
810  IF L$(K)="" THEN 820
815  GOTO 785
820  Y$="$ENDFILE"
825  GET#2,I$ : IF ST=2 OR I$="" THEN 825
830  PRINT I$; : IF I$=">" THEN 840
835  GOTO 825
840  PRINT#1, Y$
845  GET#2,I$ : IF ST=2 OR I$="" THEN 845
850  PRINT I$; : IF I$<>"#" THEN 845
855  PRINT#1,"%ECHO=ON"
860  GET#2,I$ : IF ST=2 OR I$="" THEN 860
865  PRINT I$; : IF I$<>"#" THEN 860
870  PRINT"TRANSFER COMPLETE" : RETURN
```

port routines for these devices are given elsewhere (Bar81b)

File transfer from Micro to Mainframe

The preceding section contained a detailed description of file transfer from a mainframe to a microcomputer system. In principle, the transfer of files from an intelligent terminal might be expected to require similar software - data flow, however, being in the opposite direction. Unfortunately, because the system is not totally symmetrical, the principle of reversibility cannot be fully employed. In view of this, the new algorithms and programs that are developed will need to contain mechanisms that are capable of accommodating any major differences in transmission protocol arising as a result of data flow reversal.

As before, when file transfer takes place, two basic situations must be taken into account:

a) transfer of a section of the memory space of the microsystem to the mainframe, and,
b) transmission of one of the micro's local storage files to the remote computer.

An outline algorithm for file transfer to a remote machine is presented in figure 4A. As this diagram illustrates, the program that implements the algorithm will be assigned the task of creating a file in the filestore of the host computer - if one does not already exist (steps 1 through 5). Successful file creation is followed by a loop that transmits the file data on a record by record basis.

Inspection of the BASIC listing in figure 4B will indicate that the file creation and validation activity accounts for the larger part of the program code (lines 605 through 750). The data to be transferred to the mainframe is held in the memory array L$. Once the mainframe file has been created (or its existence confirmed), data is transferred to it from L$ one element at a time. Each element of L$ corresponds to a record to be stored in the remote file. Records are transmitted over the communication link only when the host computer requests their transmission. It does this by issuing an appropriate prompt character (ASCII value 62) -as is implied by the code contained in lines 790 and 830 of the listing. Once all the non-null elements of L$ have been transferred to the remote computer the local program transmits an end-of-file message which causes the file to be closed.

Inherent in the implementation of the algorithm is the assumption that the transfer loop will be terminated by a null element within L$. If this condition is not met the program is likely to abort with an index error once the upper bound of L$ is exceeded. Should this situation arise the terminal user would then need to close the remote file 'manually'. This limitation could easily be overcome by including some extra statements at line 806:

806 IF K=N-1 THEN PRINT"ARRAY BOUND WARNING" : GOTO 820

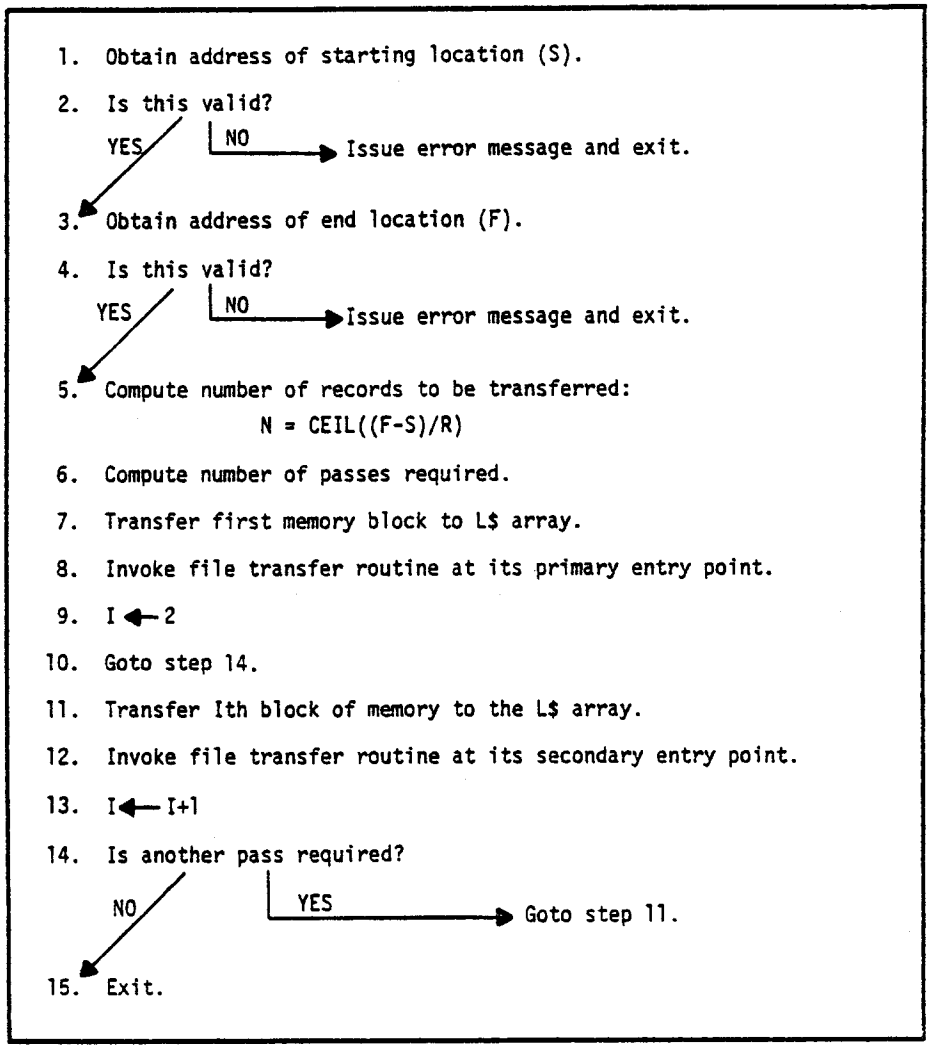
where N represents the upper bound of L$. The calling routine would now have to set the value of N prior to invoking the file transfer subroutine.

Because the data link operates in full-duplex mode (Bar81a), as data is received by the mainframe, it would normally be echoed back to the terminal. To prevent this happening during file transfer the data echoing process needs to be disabled. This is achieved by the code embodied in statements 760 through 775. The argument of the print statement in line 760 is a special message that instructs the remote mainframe not to echo back the data characters that it receives. As soon as file transmission is complete the echo back feature must be reinstated in order to enable the normal terminal mode of operation of the microcomputer. The code contained in lines 855 through 865 of the listing is responsible for this.

Notice that in addition to file creation and over-writing the implementation of the algorithm allows for a file in the mainframe to be extended. This is realised through statement 780 as a result of the value of Z$ being previously set to "(LAST+2)". This ensures that the host operating system always appends the contents of the L$ array to the end of the remote file - commencing at the (LAST+1)th record. As a result of using this approach it now becomes an easy matter to

a) overcome any limitations imposed by the size of L$, and,
b) transfer secondary storage files of any size.

FIGURE 5 ALGORITHM FOR MEMORY TRANSFER TO MAINFRAME



Either of these goals may be achieved by simply applying the transfer subroutine repeatedly (via a secondary entry point such as GOSUB 755 if need be) or by adding modifications to enable the code between lines 780 and 840 to be re-executed within a loop that could be terminated by an 'out of data' condition arising on the local microcomputer.

An illustration of this approach is contained in the skeleton algorithm for primary memory space transfer presented in figure 5. This is based upon multiple invocations of the file transfer routine contained in figure 4. In step 5, the value of R specifies the size of the records that are to be transmitted. This value will depend upon the record structure used and the way in which the information they contain is organised. Invocation of the file transfer routine at the primary entry point is necessary to perform the file creation/checking procedures and then transfer the memory block into the file. Subsequent invocations of the routine would reference its secondary entry point - thereby avoiding the initial file creation steps. An analogous algorithm could be formulated for the transfer of files from the secondary storage space of the microcomputer.

Applications of file transfer

Once it is possible to connect one or more microcomputers to either a distributed computer network or a local/remote mainframe system a variety of new applications of the combined technologies become possible. In the context of file transfer, the greatest impact of these new applications will be felt in the areas of program development, distributed processing and data base management. The ability to easily move files of data from one location to another will introduce many new approaches to programming and the way in which data may be collected, archived and shared. Numerous possibilities exist. A few illustrative examples are discussed in the remaining part of this paper.

1. Cross-loading Machine Code Programs

There are several ways of developing machine code programs for use on a microcomputer system. The most important of these are,

1) manually keying binary, octal or hexadecimal values directly into the computer's memory (CBM79),
2) via an assembly language development system resident on the microcomputer (Bar81), and,
3) using a cross-assembler running on another machine (Bar81a).

Each of these approaches have their merits and are fairly straightforward techniques to use. Details of all of the above methods have been given elsewhere. Unfortunately, method 3 is more difficult to implement than the other two since it requires a means of loading the object code, produced on the host machine, across to the target micro. However, using the file transfer routine outlined earlier, it is a fairly simple matter to cross-load programs to a micro provided that the latter is able to function as an intelligent terminal device. A strategy for transferring object code is depicted in figure c.

Once the file has been transferred to the micro (step 1) each record is checked for transmission errors. This is achieved by performing a checksum consumption and comparing the result with value embedded in the record (steps 3 and 4). If the values do not agree the mainframe is asked to retransmit the record that is in error. Up to N retransmission attempts (N is user defined) will be made. After this, if the checksum values still fail to agree, the program terminates with an appropriate error message. When all the records have been checked for correctness, the machine code program is transferred from its containing memory array into its execution area (step 7) as defined by the addresses contained in the object code.

In order to implement the cross-loading algorithm outlined above, two further support routines are thus necessary. One to perform the checksum calculations and one to perform the loading operation. Each of these will operate on a common data structure which is set up by the file transfer routine. The structure of a typical object program file is shown in figure 6. Essentially, each record consists of a sequence of hexadecimal symbols. When the file is transferred to the micro, each of these will be stored as an element of a character string array called L$.

Suppose the checksum computation is to be performed on record I (that is, array element L$(I) is to be used) and that an indication of the outcome is to be stored in the global

Figure c.

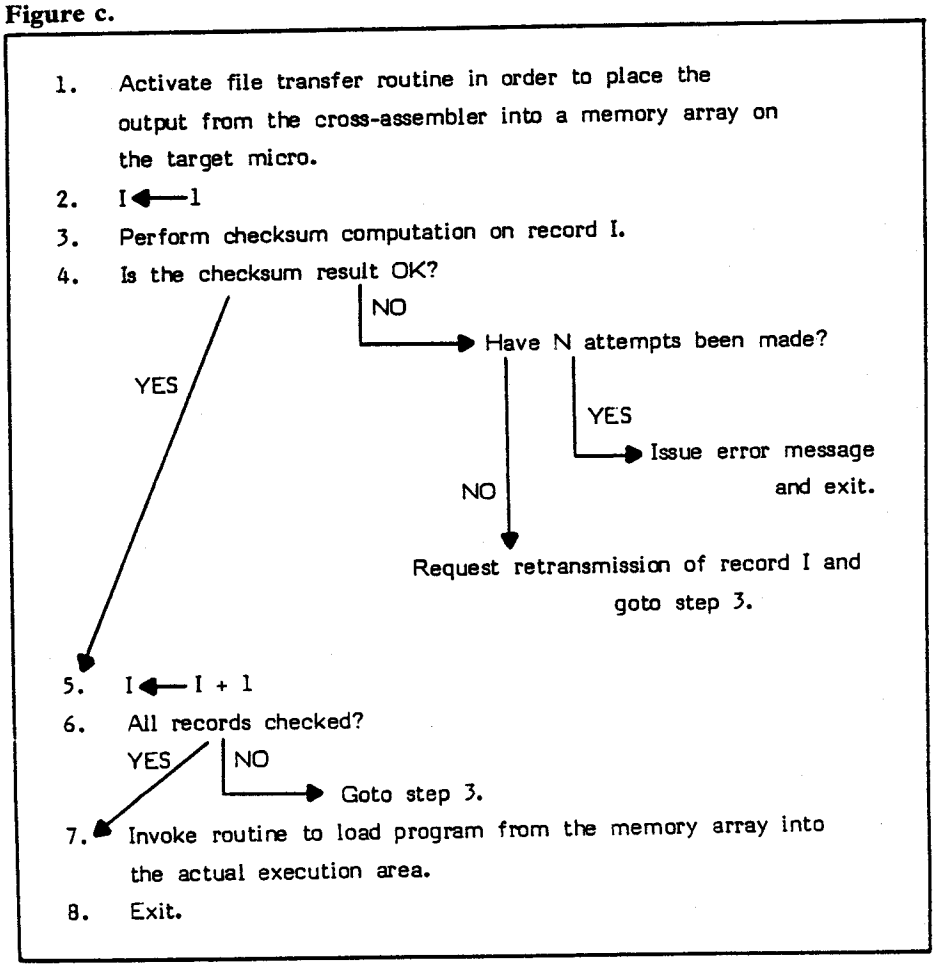
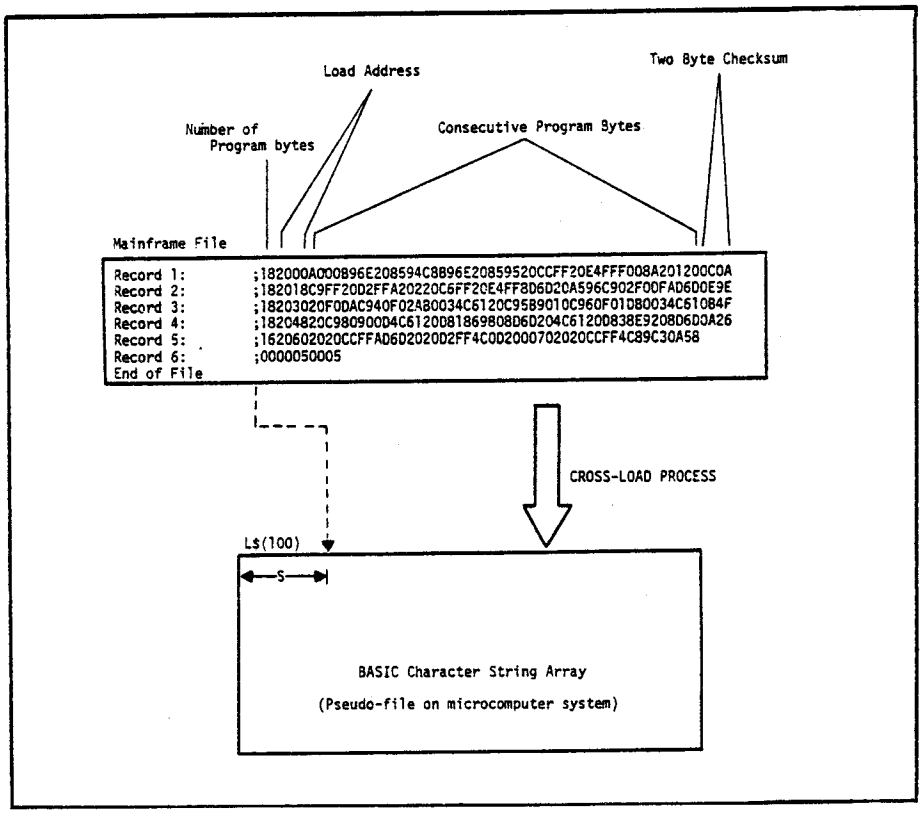


FIGURE 6 FORMAT OF AN OBJECT FILE



variable CSM (O=checksum OK, 1=checksum error). If there are R bytes in the record and the two byte checksum value embedded in it is CSV, then the following simple algorithm may be used to check the correctness of the records, as in figure d.

Suppose now that the contents of the T records of the object file (that is, the elements of L$) are to be loaded into their destination addresses in memory. An algorithm to perform this operation is illustrated in figure e.

Implementations of both the checksum and loader algorithms are illustrated in the listings contained in figure 7. Notice that within each of these routines, before any numeric processing can be undertaken values must be appropriately transformed from character to numeric form. This is easily achieved using the following sequence of instructions,

```
A% = ASC(A$) - 48
IF A% > 9 THEN A% =
A% -7
```

where A$ is a hexadecimal character value and A% is its corresponding (base 10) numeric value.1

Used in conjection with the file transfer routine, the code listed in figure 7 enables machine code programs to be easily cross-loaded to the microcomputer. Although the file

```
Checksum ← 0;
For J ← 1 to R2;
    Checksum ← Checksum + hex-value of Jth byte;
End;
If Checksum = CSV
    then CSM ← 0;
    else CSM ← 1;
Exit;
```

Figure d.

```
For I ← 1 to T;
    N ← number of program bytes in record I;
    M ← load address for byte 1 of record I;
    If invalid(M)
      then issue error message and exit;
    For J ← 1 to N;
        Load Jth program byte
                into memory address M+J-1;
    End;
End;
Exit;
```

Figure e.

FIGURE 7 ROUTINES TO SUPPORT PROGRAM CROSS-LOADING

A

```
600  REM *** CHECKSUM COMPUTATION ROUTINE
605  S1%=0 : S2%=0 : S3%=0 : S4%=0
610  S=8
615  F=LEN(L$(R))-4
620  FOR J=S TO F STEP 2
625  A$=MID$(L$(R),J,1)
630  D2%=ASC(A$)-48
635  IF D2%>9 THEN D2%=D2%-7
640  A$=MID$(L$(R),J+1,1)
645  D1%=ASC(A$)-48
650  IF D1%>9 THEN D1%=D1%-7
655  S1%=S1%+D1% : C%=0
660  IF S1%>15 THEN S1%=S1%-16 : C%=1
665  S2%=S2%+D2%+C% : C%=0
670  IF S2%>15 THEN S2%=S2%-16 : C%=1
675  S3%=S3%+C% : C%=0
680  IF S3%>15 THEN S3%=S3%-16 : C%=1
685  S4%=S4%+C% : C%=0
690  IF S4%>15 THEN S4%=S4%-16
695  NEXT J
700  FOR I = 1 TO 4
705  A$=MID$(L$(R),F+I,1)
710  V%(I)=ASC(A$)-48
715  IF V%(I)>9 THEN V%(I)=V%(I)-7
720  NEXT I
725  IF V%(4)<>S1% THEN C1%=1 : RETURN
730  IF V%(3)<>S2% THEN C1%=1 : RETURN
735  IF V%(2)<>S3% THEN C1%=1 : RETURN
740  IF V%(1)<>S4% THEN C1%=1 : RETURN
745  C1%=0  :  RETURN
```

B

```
800  REM *** LOADER ROUTINE
802  S=8  :  I=0
804  I=I+1  :  REM GET NUMBER OF BYTES
806  A$=MID$(L$(I),S,1)
808  B$=MID$(L$(I),S+1,1)
810  A%=ASC(A$)-48  :  B%=ASC(B$)-48
812  IF A%>9 THEN A%=A%-7
814  IF B%>9 THEN B%=B%-7
816  N=16*A%+B%
818  IF N=0 THEN 866
820  FOR K= 1 TO 4
822  A$=MID$(L$(I),S+1+K,1)
823  V%(K)=ASC(A$)-48
824  IF V%(K)>9 THEN V%(K)=V%(K)-7
826  NEXT K
828  M=0 : REM COMPUTE LOAD ADDRESS
830  FOR K=1 TO 4
832  M=M+V%(K)*(16+(4-K))
834  NEXT K : REM DO MEMORY CHECK
836  L%=256*PEEK(47)+PEEK(46)
838  U%=256*PEEK(49)+PEEK(48)
840  IF M<(L%+1) THEN 868
842  IF M>(U%-N) THEN 868
844  C%=S+6
846  FOR J=0 TO N-1
848  A$=MID$(L$(I),C%,1)
850  B$=MID$(L$(I),C%+1,1)
852  A%=ASC(A$)-48 : B%=ASC(B$)-48
854  IF A%>9 THEN A%=A%-7
856  IF B%>9 THEN B%=B%-7
858  W%=16*A%+B%
860  POKE M+J,W%
862  C%=C%+2
864  NEXT J
865  GOTO 804
866  PRINT "LOAD TERMINATED OK" : RETURN
868  PRINT "LOAD ERROR - MEMORY CONFLICT" : RETURN
```

transfer module is quite general, each of the other two subroutines reflect the structure of the object code records. They are thus 'strongly bound' to the particular cross-assembler being used. However, provided the overall algorithm is implemented using a modular approach, it is a simple matter to recode appropriate checksum and loader modules to suit other cross-assemblers.

Some results of timing measurements for the various steps involved in the cross-loading process (using a 300 baud disk-up link) are shown in figure f (right).

Figure f.

Transfer time	=	3 seconds	per record
Checksum validation	=	3 seconds	
Loading time	=	2 seconds	
Total	=	8 seconds	

A small file (20 records) would thus take about 3 minutes to cross-load. This is an estimate of the minimum time since it is based upon the assumption that there are no re-transmission overheads associated with the transfer process. Thus, for smally programs (about 500 bytes) the transfer times are not too long. However, for larger programs (say, 10K) much longer load times would be experienced. For this size of program a high speed communications link would really be needed. In addition, the software in the intelligent terminal would need to be directly executable machine code rather than interpreted BASIC.

2. Cross-loading of BASIC Programs

Transportability of computer programs has always been a problem that has never been solved in a really elegant way. As a general rule it is widely accepted that programs written in machine orientated languages cannot easily be transported from one computer to another of a different type. In contrast, programs written in high level languages (such as COBOL, FORTRAN, BASIC) are usually much easier to transport between machines. However, difficulties can still arise as a consequence of different languages dialects, differences in machine architecture and the software environments in which a program is likely to be used.

One of the areas where considerable difficulty is likely to be encountered is that in which a mainframe (acting as a back-end data base machine) provides a storage facility for various microcomputer systems. Such an arrangement is illustrated in figure g.

Figure g.

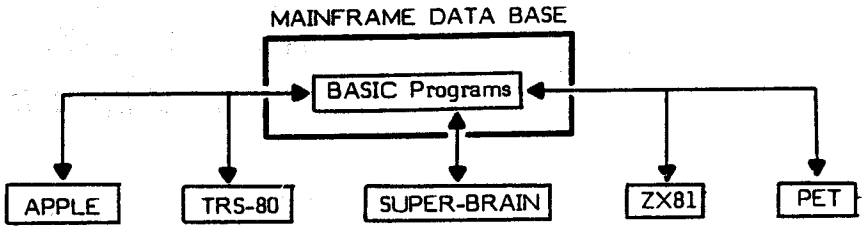


Figure h.

1. Transfer source file from mainframe to local micro.
2. I ← 1
3. Load Ith BASIC source statement into buffer memory.
4. Convert source statement into internal (tokenised) format.
5. Move tokenised statement into its execution location within memory.
6. I ← I + 1
7. All source statements processed?
   YES → 8. Exit.
   NO → Goto step 3.

There are many ways of storing BASIC programs in a back-end data base system. Usually, however, they are all variants of one or other of two general methods. Either the programs are stored

a) as memory space images or,
b) in source text format.

Method a) permits easy and rapid sharing of programs between machines of the same type and enables the use of very simple loading software in the intelligent terminal. Indeed, using this technique, BASIC programs may be treated just like machine code programs. However, there is no portability across machines of different types.

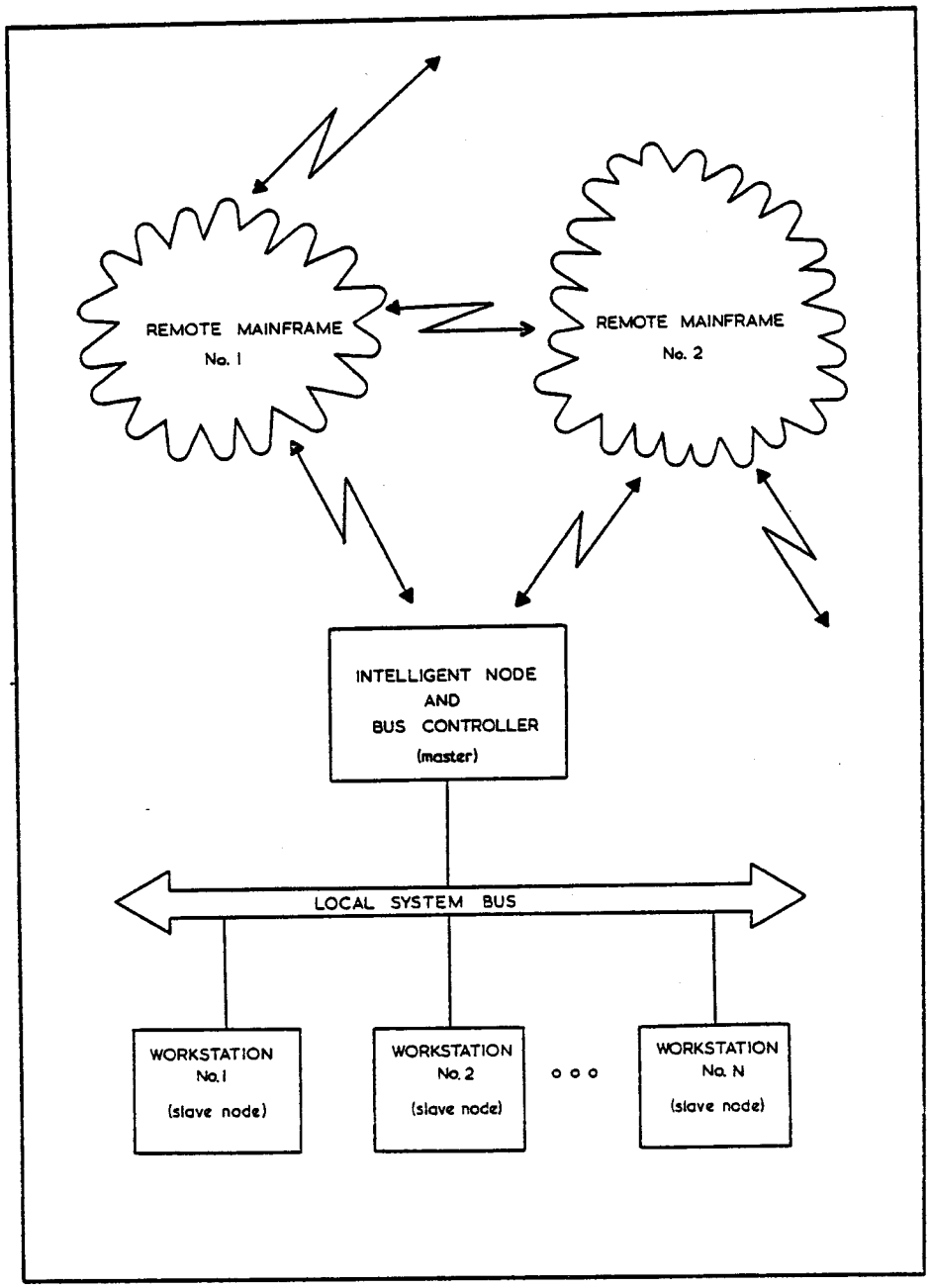
Method b) is more useful from the point of view of sharing software in high level language format - thereby enabling distribution of development effort. Because programs are stored in source code format they can be more easily updated, modified and shared between intelligent terminals of different types. However, when programs are transmitted to local machines there is a need to provide loading software that is much more sophisticated than that required for method a). The complexity of loaders for this type of application arises as a consequence of the need to convert programs from source code format into their internal machine code representation.

Each of the above methods has its uses and particular areas of application. Detailed descriptions of both techniques are given elsewhere (Bar81). Because it introduces some new concepts, an outline of the algorithm for method b) is presented in figure h.

This algorithm has been implemented on a Commodore PET system linked to a remote mainframe. Step 1 is essentially the same as that described in the earlier part of this paper (FROM MAINFRAME TO MICRO - file transfer to secondary storage). Steps 2 through 8 were coded in assembler and located at the high end of memory (locations $7FFF for a 32K PET). The program is quite simple and uses appropriate branches to the PET's BASIC ROM interpreter in order to perform steps 4 and 5. A listing of the assembler program is given in Bar81c.

In order to assess the time required to perform a load operation we wrote some BASIC code to generate a test program containing 1000 statements. The average length of each statement was about 25 characters and its load size was 19 Kbytes. Loading of the source code version of this program from tape took about 16 minutes. Once in memory the test program was SAVEd to tape and then re-LOADed using the appropriate PET commands. Under these conditions loading took only 5 minutes. Loading of source code programs is thus about three times slower than that of the corresponding binary versions. Disk storage media would obviously enable much faster loading operations. We have not directly measured the times involved but would anticipate improvements of up to a factor of 30-40 in load time.

FIGURE 8 A DISTRIBUTED PROCESSING SYSTEM



3. Distributed Processing

A distributed processing system is one in which the resources necessary to achieve a particular functional objective are dispersed over a number of discrete geographical locations (Lor80, Sch81). In order to optimize the use of the available resources the locations at which they reside are usually interconnected by means of some form of communication network. This enables both messages and files to be freely transmitted between appropriate parts of the overall system.

There are many examples of systems based upon this type of architecture. One of the simplest involves the use of a mainframe computer to which are attached a variety of intelligent terminal devices. Normally, these terminals will be located at many different physical sites and each will have its own data storage and information processing capability. In addition to the mainframe links there are likely to be many different interconnections between the individual terminal devices. The basic building blocks thus enable a wide variety of systems to be constructed. These may have either a fixed or dynamically re-configurable architecture. An example of one fairly common structure is illustrated in figure 8.

In this system, two remote mainframe systems provide support facilities (data storage and processing capacity) for an intelligent device that, in turn, services a series of individual workstations. These are autonomous units which have the capacity of communicating with the intelligent device via a shared bus. This enables workstations to communicate one with another and also allows individual units to indirectly send and receive information from the remote mainframe computers. An arrangement similar to this is often used in situations where shared access to stored information is desired.

As an illustration of the application of such a system, consider the 'distributed processing' approach to conducting some form of computer based learning or testing exercise on a group basis. An algorithm that describes one possible approach to the way in which this may be conducted is shown in figure g.

In this example, a variety of file transfer processes are involved: from the mainframe to the micro (steps 1

1. Transmit aptitude/IQ test from central site to remote intelligent node.
2. Distribute test to slave nodes (workstations) at the remote site.
3. Perform tests using workstations at remote site.
4. Gather results of individual tests at remote site.
5. Transmit results data from remote intelligent node to central site(s).
6. Process results using a mainframe statistical package.
7. Archive results in a global data base.
8. Transmit processed results back to remote node as the load data for a local data base.
9. etc.

and 8), from the micro to the mainframe (step 5) and between micro system (steps 2 and 4). Processing takes place both at the remote station (step 3) and at one of the central sites (step 6). Finally, data is archived on the mainframe (step 7) and a subset of it is stored at the intelligent node (step 8).

Based upon the simple analysis that has been outlined above it is easy to see that distributed processing applications are critically dependent upon the ease with which file/message transfer can take place between the various components from which the system is constructed. As the cost of there components decreases there is likely to be a substantial increase in the use opf distributed processing architectures for a wide variety of applications involving the use of intelligent terminals.

Conclusion

The economic merits of using a stand alone microcomputer system as an intelligent terminal device are potentially quite substantial. A variety of applications for such terminals now exist. Many of these involve some form of data transmission - either in the form of control messages or file transfer.

Of the many different possibilities that exist, two of the more useful types of application depend upon the fact that

a) using the software available on a remote node of a computer network can greatly enhance the type of work that can be undertaken, and,
b) the ability to transfer files to/from a local node can greatly facilitate the implementation of distributed processing techniques.

In this paper three applications of intelligent terminals have been briefly outlined: many others obviously exist.

A variety of hardware and software packages are available to enable this mode of microcomputing to be undertaken. The number of these is likely to increase as more 'intelligent' is built into the interface that the intelligent terminal presents to the 'outside world' - auto diallers, error checking devices, encryption facilities, message/file compaction units, and so on. In the future it is likely that the most difficult problem to overcome will be that of choosing the terminal device that is most appropriate to any particular application.

References

Ber81 Bernstein, G.B. and Kashar, A.S., **Intelligent Terminals: Functions, Specificaitions and Applications,** QED Information Sciences, ISBN: 0-89435-021-8-320, 1981

Bar81 Barker, P.G., A Comparison of Sort Times Using a Microcomputer, Interactive Systems Group Working Paper, February 1981.

Wie81 Wiederhold, G., **Database Design,** McGraw-Hill, ISBN: 0-07-070130-X, 1977.

Jud73 Judd, D.R., **Use of Files,** MacDonald/American Elsevier Computer Monographs, ISBN: 0-444-19568-8, 1973

CBM79 Commodore Business Machiens, Inc., **CBM 2001-19, -32, 3016, 3032 Pesonal Computer User's Handbook,** Publication No. 320856-3, June 1979.

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

Don80 Donahue, C.S. and Enger, J.K., **PET/CBM Personal Computer Guide,** Osborne/McGraw-Hill, ISBN: 0-931988-30-6, 1980.

CBM80 Commodore Business Machines, Inc., **User's Manual for CBM Dual Drive Floppies** (Models: 2040, 3040, 4040, 8050), Part No. 320899, October 1980.

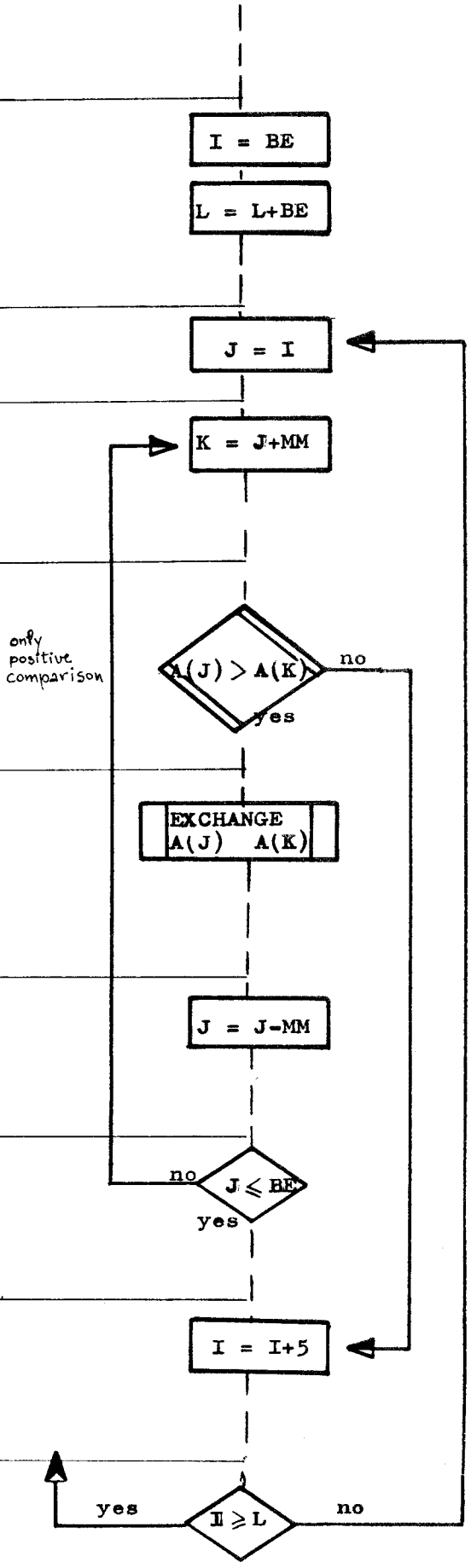
Bar81b Barker, P.G., **Support Routines for File Transfer to Microcomputer Secondary Storage Devices,** Interactive System Group Working Paper, June 1981.

Bar81c Barker, P.G., **Use of the Public Switched Network for Program Transfer to Micros,** Interactive Systems Group Working Paper, July 1981.

Lor80 Lorin, H., **Aspects of Distributed Computer Systems,** John Wiley & Sons, ISBN: 0-471-08114-0, 1980.

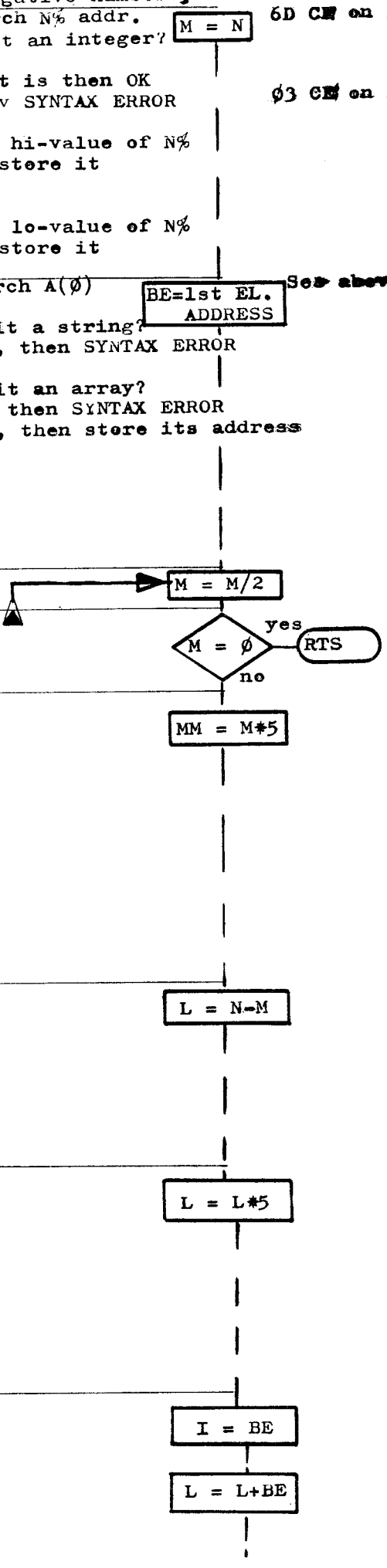
Sch81 Schneider, H.J., **Distributed Versus Central Systems? - Distributed Data Base Systems a Necessity for Distributed Organisations?,** Paper Presented at the NATO Advanced Study Institute on Data Base Management and Applications, Estoril, Portugal, June 1-14, 1981.

```
7F7E  32638    STA.PO      69 (105)
7F80  32640    LDA.PO      6C (108)
7F82  32642    STA.PO      60 ( 96)
7F84  32644    CLC
7F85  32645    ADC.PO      68 (104)
7F87  32647    STA.PO      68 (104)
7F89  32649    LDA.PO      6D (109)
7F8B  32651    STA.PO      61 ( 97)
7F8D  32653    ADC.PO      69 (105)
7F8F  32655    STA.PO      69 (105)
7F91--32657    LDA.PO      60 ( 96)
7F93  32659    STA.PO      62 ( 98)
7F95  32661    LDA.PO      61 ( 97)
7F97  32663    STA.PO      63 ( 99)
7F99--32665    LDA.PO      62 ( 98)
7F9B  32667    CLC
7F9C  32668    ADC.PO      6A (106)
7F9E  32670    STA.PO      66 (102)
7FA0  32672    LDA.PO      63 ( 99)
7FA2  32674    ADC.PO      6B (107)
7FA4  32676    STA.PO      67 (103)
7FA6  32678    LDY.IMM     00 (  0)
7FA8--32680    LDA.IND+Y   66 (102)
7FAA  32682    CMP.IND+Y   62 ( 98)
7FAC  32684    BCC         09      > 32695
7FAE  32686    BNE         31      > 32737
7FB0  32688    INY
7FB1  32689    CPY.IMM     04 (  4)
7FB3  32691    BMI         F3      > 32680
7FB5  32693    BPL         2A      > 32737
7FB7--32695    LDY.IMM     04 (  4)
7FB9  32697    LDA.IND+Y   66 (102)
7FBB  32699    TAX
7FBC  32700    LDA.IND+Y   62 ( 98)
7FBE  32702    STA.IND+Y   66 (102)
7FC0  32704    TXA
7FC1  32705    STA.IND+Y   62 ( 98)
7FC3  32707    DEY
7FC4  32708    BPL         F3      > 32697
7FC6  32710    LDA.PO      62 ( 98)
7FC8  32712    SEC
7FC9  32713    SBC.PO      6A (106)
7FCB  32715    STA.PO      62 ( 98)
7FCD  32717    LDA.PO      63 ( 99)
7FCF  32719    SBC.PO      6B (107)
7FD1  32721    STA.PO      63 ( 99)
7FD3  32723    BCC         0C      > 32737
7FD5  32725    CMP.PO      6D (109)
7FD7  32727    BCC         08      > 32737
7FD9  32729    BNE         BE      > 32665
7FDB  32731    LDA.PO      62 ( 98)
7FDD  32733    CMP.PO      6C (108)
7FDF  32735    BCS         B8      > 32665
7FE1--32737    LDA.PO      60 ( 96)
7FE3  32739    CLC
7FE4  32740    ADC.IMM     05 (  5)
7FE6  32742    STA.PO      60 ( 96)
7FE8  32744    LDA.PO      61 ( 97)
7FEA  32746    ADC.IMM     00 (  0)
7FEC  32748    STA.PO      61 ( 97)
7FEE  32750    CMP.PO      69 (105)
7FF0  32752    BCC         9F      > 32657
7FF2  32754    BNE         06      > 32762
7FF4  32756    LDA.PO      60 ( 96)
7FF6  32758    CMP.PO      68 (104)
7FF8  32760    BCC         97      > 32657
7FFA--32762    JMP         38 7F   32568 <  ; Care when relocating
```

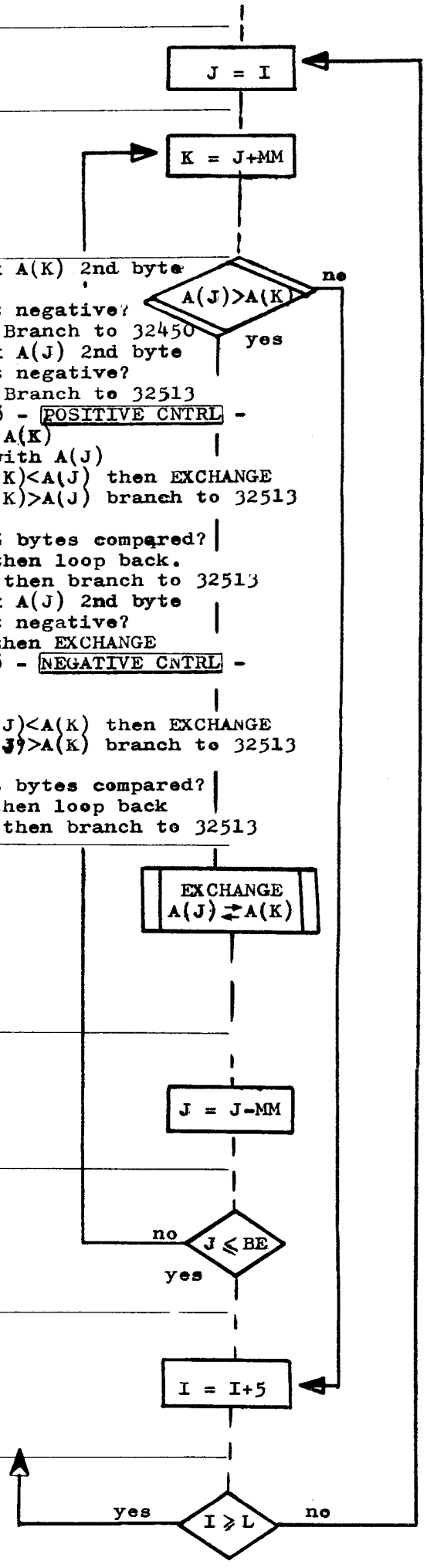


BASIC 4.Ø S O R T for positive and negative numbers BASIC 2.Ø

```
7E00  32256  JSR        2B C1    49451 < ; Search N% addr.
7E03  32259  LDA.PO     08 (  8)          ; Is it an integer?
7E05  32261  CMP.IMM    80 (128)
7E07  32263  BEQ        03       > 32268  ; If it is then OK
7E09  32265  JMP        00 BE    48896 < ; elsev SYNTAX ERROR
7E0C  32268  LDY.IMM    00 (  0)
7E0E  32270  LDA.IND+Y  44 ( 68)          ; Load hi-value of N%
7E10  32272  STA.PO     01 (  1)          ; and store it
7E12  32274  STA.PO     CF (207)
7E14  32276  INY
7E15  32277  LDA.IND+Y  44 ( 68)          ; Load lo-value of N%
7E17  32279  STA.PO     00 (  0)          ; and store it
7E19  32281  STA.PO     CE (206)
7E1B  32283  JSR        2B C1    49451 < ; Search A(Ø)
7E1E  32286  LDA.PO     07 (  7)
7E20  32288  CMP.IMM    FF (255)          ; Is it a string?
7E22  32290  BEQ        E5       > 32265  ; Yes, then SYNTAX ERROR
7E24  32292  LDA.PO     45 ( 69)
7E26  32294  CMP.PO     2D ( 45)          ; Is it an array?
7E28  32296  BCC        DF       > 32265  ; No, then SYNTAX ERROR
7E2A  32298  STA.PO     6D (109)          ; Yes, then store its address
7E2C  32300  BNE        06       > 32308
7E2E  32302  LDA.PO     2C ( 44)
7E30  32304  CMP.PO     44 ( 68)
7E32  32306  BCS        D5       > 32265
7E34  32308  LDA.PO     44 ( 68)
7E36  32310  STA.PO     6C (108)
7E38  32312  LSR.PO     CF (207)
7E3A  32314  ROR.PO     CE (206)
7E3C  32316  BNE        05       > 32323
7E3E  32318  LDA.PO     CF (207)
7E40  32320  BNE        01       > 32323
7E42  32322  RTS
7E43  32323  LDA.PO     CF (207)
7E45  32325  STA.PO     6B (107)
7E47  32327  LDA.PO     CE (206)
7E49  32329  STA.PO     6A (106)
7E4B  32331  ASL.PO     6A (106)
7E4D  32333  ROL.PO     6B (107)
7E4F  32335  ASL.PO     6A (106)
7E51  32337  ROL.PO     6B (107)
7E53  32339  CLC
7E54  32340  ADC.PO     6A (106)
7E56  32342  STA.PO     6A (106)
7E58  32344  LDA.PO     CF (207)
7E5A  32346  ADC.PO     6B (107)
7E5C  32348  STA.PO     6B (107)
7E5E  32350  LDA.PO     00 (  0)
7E60  32352  SEC
7E61  32353  SBC.PO     CE (206)
7E63  32355  STA.PO     68 (104)
7E65  32357  TAX
7E66  32358  LDA.PO     01 (  1)
7E68  32360  SBC.PO     CF (207)
7E6A  32362  STA.PO     69 (105)
7E6C  32364  TAY
7E6D  32365  ASL.PO     68 (104)
7E6F  32367  ROL.PO     69 (105)
7E71  32369  ASL.PO     68 (104)
7E73  32371  ROL.PO     69 (105)
7E75  32373  TXA
7E76  32374  CLC
7E77  32375  ADC.PO     68 (104)
7E79  32377  STA.PO     68 (104)
7E7B  32379  TYA
7E7C  32380  ADC.PO     69 (105)
7E7E  32382  STA.PO     69 (105)
7E80  32384  LDA.PO     6C (108)
7E82  32386  STA.PO     60 ( 96)
7E84  32388  CLC
7E85  32389  ADC.PO     68 (104)
7E87  32391  STA.PO     68 (104)
7E89  32393  LDA.PO     6D (109)
7E8B  32395  STA.PO     61 ( 97)
7E8D  32397  ADC.PO     69 (105)
```



```
7E8F  32399  STA.PO     69 (105)
7E91  32401  LDA.PO     60 ( 96)
7E93  32403  STA.PO     62 ( 98)
7E95  32405  LDA.PO     61 ( 97)
7E97  32407  STA.PO     63 ( 99)
7E99  32409  LDA.PO     62 ( 98)
7E9B  32411  CLC
7E9C  32412  ADC.PO     6A (106)
7E9E  32414  STA.PO     66 (102)
7EA0  32416  LDA.PO     63 ( 99)
7EA2  32418  ADC.PO     6B (107)
7EA4  32420  STA.PO     67 (103)
7EA6  32422  LDY.IMM    01 (  1)       ; Check A(K) 2nd byte
7EA8  32424  LDA.IND+Y  66 (102)
7EAA  32426  ROL.A                     ; Is it negative?
7EAB  32427  BCS        15     > 32450 ; Yes, Branch to 32450
7EAD  32429  LDA.IND+Y  62 ( 98)       ; Check A(J) 2nd byte
7EAF  32431  ROL.A                     ; Is it negative?
7EB0  32432  BCS        4F     > 32513 ; Yes, Branch to 32513
7EB2  32434  DEY                      ; Y = ø - POSITIVE CNTRL -
7EB3  32435  LDA.IND+Y  66 (102)       ; Load A(K)
7EB5  32437  CMP.IND+Y  62 ( 98)       ; CMP with A(J)
7EB7  32439  BCC        1E     > 32471 ; IF A(K)<A(J) then EXCHANGE
7EB9  32441  BNE        46     > 32513 ; IF A(K)>A(J) branch to 32513
7EBB  32443  INY
7EBC  32444  CPY.IMM    04 (  4)       ; All 5 bytes compared?
7EBE  32446  BMI        F3     > 32435 ; NO  then loop back.
7EC0  32448  BPL        3F     > 32513 ; YES  then branch to 32513
7EC2  32450  LDA.IND+Y  62 ( 98)       ; Check A(J) 2nd byte
7EC4  32452  ROL.A                     ; Is it negative?
7EC5  32453  BCC        10     > 32471 ; NO  then EXCHANGE
7EC7  32455  DEY                      ; Y = ø - NEGATIVE CNTRL -
7EC8  32456  LDA.IND+Y  62 ( 98)
7ECA  32458  CMP.IND+Y  66 (102)
7ECC  32460  BCC        09     > 32471 ; If A(J)<A(K) then EXCHANGE
7ECE  32462  BNE        31     > 32513 ; If A(J)>A(K) branch to 32513
7ED0  32464  INY
7ED1  32465  CPY.IMM    04 (  4)       ; All 5 bytes compared?
7ED3  32467  BMI        F3     > 32456 ; NO  then loop back
7ED5  32469  BPL        2A     > 32513 ; YES  then branch to 32513
7ED7  32471  LDY.IMM    04 (  4)
7ED9  32473  LDA.IND+Y  66 (102)
7EDB  32475  TAX
7EDC  32476  LDA.IND+Y  62 ( 98)
7EDE  32478  STA.IND+Y  66 (102)
7EE0  32480  TXA
7EE1  32481  STA.IND+Y  62 ( 98)
7EE3  32483  DEY
7EE4  32484  BPL        F3     > 32473
7EE6  32486  LDA.PO     62 ( 98)
7EE8  32488  SEC
7EE9  32489  SBC.PO     6A (106)
7EEB  32491  STA.PO     62 ( 98)
7EED  32493  LDA.PO     63 ( 99)
7EEF  32495  SBC.PO     6B (107)
7EF1  32497  STA.PO     63 ( 99)
7EF3  32499  BCC        0C     > 32513
7EF5  32501  CMP.PO     6D (109)
7EF7  32503  BCC        08     > 32513
7EF9  32505  BNE        9E     > 32409
7EFB  32507  LDA.PO     62 ( 98)
7EFD  32509  CMP.PO     6C (108)
7EFF  32511  BCS        98     > 32409
7F01  32513  LDA.PO     60 ( 96)
7F03  32515  CLC
7F04  32516  ADC.IMM    05 (  5)
7F06  32518  STA.PO     60 ( 96)
7F08  32520  LDA.PO     61 ( 97)
7F0A  32522  ADC.IMM    00 (  0)
7F0C  32524  STA.PO     61 ( 97)
7F0E  32526  CMP.PO     69 (105)
7F10  32528  BCC        0B     > 32541
7F12  32530  BNE        06     > 32538
7F14  32532  LDA.PO     60 ( 96)
7F16  32534  CMP.PO     68 (104)
7F18  32536  BCC        03     > 32541
7F1A  32538  JMP        38 7E    32312 < ; Care when relocating
7F1D  32541  JMP        91 7E    32401 < ; Care when relocating
```



Input subroutine

Bradford Metropolitan Council owns 85 Pets at present (July 1981) most of which are situated in schools, though 9 departments within the Authority have their own Pets, some with printers and discs.

One problem we have found with all users when inputting information from the keyboard is that they have a district tendency to watch the keys rather than the screen. This leads to problems when using the INPUT statement as very often the return key is pressed before information has been entered. The user then continues typing data followed by return and so can remove a large portion of the program.

Our input subroutine was originally written just to overcome null entries, but was gradually extended to enable the user to enter only the correct type of input and also to restrict the number of characters input at any time. In other words, the input subroutine performs a great deal of validation on data entered from the keyboard.

Input types

The subroutine gets a character at a time from the keyboard and builds up an input string.

The subroutine is designed to accept certain input types - these can easily be changed for programs requiring different types. The standard subroutine used by Bradford as a starting point for most programs contains four different input types. These are indicated by the variable IT where

IT = 1	is digits only
IT = 2	is digits and point
IT = 3	is Y or N only
IT = 4	is any keyboard character except quotes

The only cursor control allowed is 'delete': other cursor movements are rejected.

A shifted return is treated in exactly the same manner as return. We have found this to be necessary when the machine is used in lower case, when users may require names and addresses completely in upper case, so they use the 'shiftlock' key throughout.

The flashing cursor is replaced by a non-flashing underline as a prompt.

A maximum number of acceptable characters is specified before calling the subroutine. Any or all of these can be deleted back to a null string, but no further.

Variables set

Before entering the input subroutine two variables must be set, IT to specify the type of input and N to specify the maximum length of input.

Example 1

```
.....
730 .....
720 PRINT "ENTER NAME";
730 IT = 4 : N = 20 : GOSUB 120
740.....
```

The input type is set to 4, i.e. any keyboard character and the maximum length is set to 20 characters.

4 Variables used

The following variables are used within the input subroutine.

L	- current length of input string
E$	- single character obtained from keyboard
AS	- ASCII code for character from keyboard
WS$	- current input string

5 Explanation of coding

120	Sets current length of string zero Sets current string to null Print underline as prompt
130	Gets one character from keyboard
140	Tests for return or shifted return
150	Tests for delete or shifted delete
160-170	Tests for invalid keys -e.g. cursor movements
200-270	Branches and tests for various input types
400	Tests for maximum length of string
410	Prints character from keyboard Prints next prompt Increases input string and length by one
450	tests for null string before deleting character
460	Deletes one character Prints new prompt Decreases input string and length by one
470	Test for null string when 'return' input
480	Clears prompt and returns to main program

The input string is returned to the main program WS$. This can easily be converted to a numeric value if required using the VAL function.

With the line numbers given here, ample space is left between lines 270 and 400 for additional input types if required.

The subroutine runs very much faster if placed at the beginning of the program, as it contains a number of 'GOTO' statements, each of which is searched for from the start of BASIC.

Whilst further validation is necessary in some cases, this input subroutine is very useful not only in selecting required types of input, thus avoiding confusing messages of "REDO FROM START", but also in restricting the input length, an important feature if using disc files or formatted printing.

Since the subroutine is in Basic it is fairly slow, though not excessively so. In order to speed up the procedure, it is presently being rewritten in machine code.

Anita Cornwell
Computer Division
Bradford Metropolitan Council
Britannia House
Hall Ings
Bradford
West Yorkshire BD1 1HX

```
INPUT SUBROUTINE

100 POKE59468,14:PRINT"□":GOTO500
110 REM INPUT SUBROUTINE
120 L=0:WS$="":PRINT"_■";
130 GETE$:IFE$=""GOTO130
140 AS=ASC(E$):IFAS=13ORAS=141GOTO470
150 IFAS=20ORAS=148GOTO450
160 IFAS<13OR(AS>13ANDAS<32)OR(AS=34)OR(AS>127ANDAS<141)GOTO130
170 IFAS>141ANDAS<161GOTO130
200 ONITGOTO210,230,250,270
210 IFAS>47ANDAS<58GOTO400:                    REM DIGITS ONLY
220 GOTO130
230 IF(AS>47ANDAS<58)ORAS=46GOTO400:           REM DIGITS AND POINT
240 GOTO130
250 IFAS=78ORAS=89GOTO400:                     REM Y OR N ONLY
260 GOTO130
270 IFAS>31ANDAS<255GOTO400:                   REM ANYTHING EXCEPT QUOTES
400 IFL=NGOTO130
410 PRINTCHR$(AS)"_■";:L=L+1:WS$=WS$+E$:GOTO130
450 IFL=0GOTO130
460 PRINTCHR$(20)"_■";:L=L-1:WS$=LEFT$(WS$,LEN(WS$)-1):GOTO130
470 IFL=0GOTO130
480 PRINT" ":RETURN
500 REM MAIN PROGRAM

READY.
```

Driving the User Port in BASIC

The PET employs the IEEE-488 bus for general purpose interfacing of external devices, however, for "quick and dirty" interfacing problems, it may be simpler and cheaper to use the 8-bit parallel I/O port. This port is capable of handling many common peripherals including an ASCII keyboard, a printer or a paper tape reader, but ony one device can be connected to the port at a time without some external switching logic.

The 8-bit port is actually part of an MOS Technology MCS 6522 Versatile Interface Adapter (VIA). Most of the VIA's features apparently are used for the PET itself, leaving only an 8-bit port and two handshake lines, which are really quite simple to use. This discussion will limit itself to input through the 8-bit port, but essential information for output through the port is included.

On the 8-bit port edge connector: pins A and N are grounded, pin B is CA1, the input handshake line, pin M is CB2, the output handshake line, and pins C through L are the 8 data lines, with C being the high order (leftmost) and L the low order bit. When the PET is turned on, the 8 data bits are programmed to act as inputs and CA1 is programmed to recognize a negative transition (from 1 to 0). If the handshake or data strobe line on your peripheral device produces a positive transition, you can reprogram CA1 with the BASIC statement:

POKE 59468,PEEK(59468) OR 1

which changes the CA1 control bit in the VIA's Peripheral Control Register (address 59468) from 0 to 1.

When a transition occurs on CA1, meaning that data is ready to be read from the data lines, the next to low order bit in the VIA's Interrupt Flag Register (the CA1 flag bit) will be set. You can test for this with the BASIC statement:

WAIT 59469,2

which takes the contents of the Interrupt Flag Register, AND's it with 2 or binary 00000010, and tests the result, repeating the test until the result is non-zero. (Note that execution of the WAIT statement cannot be interrupted with the RUN/STOP key, so you should have a way of manually creating a transition on CA1 when you are testing the interface).

After execution of the WAIT statement the data present at the 8-bit port is ready to be read with the BASIC statement:

D=PEEK(59457)

which reads the VIA's Port A and stores the data in the BASIC variable D as an unsigned integer between 0 and 255. A side effect of the PEEK is to reset the CA1 flag bit in the Interrupt Flag Register, thereby setting things up for execution of the next WAIT statement.

Thus, to read a whole line of ASCII characters ending with a carriage return (binary 00001101 = 13) into a string variable, you could use the following program segment:

```
10 A$=""
20 FORI=1T072
30 WAIT 59469,2
40 D=PEEK(59457) AND 127
50 IF D=13 THEN80
60 A$=A$+CHR$(D)
70 NEXT I
80 PRINT A$
```

Here statement 20 simply limits the number of characters read to 72; statement 40 masks the data read to 7 bits to eliminate any parity bit; and statement 60 uses string concatenation to convert the data into a single string. Although the PET's internal character code is essentially ASCII, some character code translation will be needed in most practical applications. This can easily be done with an array in BASIC.

To use the 8 bit port for output, you must first program the data lines to act as outputs with the BASIC statement: POKE 59459,255

which sets all bits of the VIA's Data Direction Register A to 1s. Handshaking is considerably less convenient, since only the CB2 line is brought to the edge connector. You can force CB2 to a logic 1 with the BASIC statement:

POKE 59468,PEEK(59468) OR 224

and reset it to 0 with: POKE 59468, PEEK(49468) AND 31 OR 192

Reproduced by kind permission of Ron Geere.

Multidimensional Arrays

From page 51 of our CBM Users Manual we all know that the COMMODORE allows three-dimensional arrays. But I did not realize, until I received a tape from my brother in South Africa who has been playing with his new computer, that it allows more than THREE dimensions in an array. So I tried a four and a five-dimensional array in my 32K CBM and it worked. Don't put too many elements into the array or you will run out of memory. After all, a five-dimensional array with 5 elements per dimension, contains 6↑6 = 46656 elements (don't forget the zero position elements!).

Even the excellent "Pet/CBM Personal Computer Guide", only hints at more than three-dimensional arrays on pages 70 and 126.

Here is a short program that fills a five-dimensional array with four elements with consecutive numbers. I did not use the 0 position here, or it would in reality have been a five-dimensional array with 5 elements.

Line 10 dimensions the integer array A%. Line 15 to 70 fill the array positions with consecutive numbers.

Line 80 and 90 ask you to nominate an array position and lines 95 to 105 print the number contained in the nominated position.

From the print-out it is clear that position A%(1,1,1,1,1,) contains: 1 and the last position A%(4,4,4,4,4) contains 1024, which is 4↑5.

Apart from people interested in higher mathematics, multiple dimension arrays might be useful for small businesses. For instance, the first 3 dimensions of e.g. a 5-dimensional 4 element array could be used as code numbers for 125 customers (also using the 0 elements). That would leave another 5 x 5 = 25 array positions to store 25 items of information about the customer. If e.g. all the postcode numbers are stored in position e.g. a%(4,4,3,0,0) and e.g. a%(4,2,1,0,0), it would seem quicker to sort and print e.g. all the postcodes by printing all array positions whose 4th and 5th dimensions are 0.

I wonder what the actual limit of dimensions allowed in the COMMODORE is; has anybody tried it? Has anybody used multi-dimensional arrays to store business data?

```
5  REM         DIMENSIONAL ARRAY PROGRAM
10 DIMA%(4,4,4,4,4)
12 PRINT "[cls] FILLING 5 DIMENSIONAL ARRAY"
15 Z=1
20 FORI=1TO4
30 FORJ=1TO4
40 FORK=1TO4
50 FORL=1TO4
55 FORM=1TO4
60 A%(I,J,K,L,M)=Z
65 Z=Z+1
70 NEXT:NEXT:NEXT:NEXT:NEXT
80 PRINT"WHAT POSITION?"
90 INPUT"ENTER I,J,K,L,M";I,J,K,L,M
95 OPEN1,4
100 PRINT#1,"POSITION A%(";I;",";J;",";K;",";L;",";M;")=";A%(I,J,K,L,M)
105 CLOSE1
110 GOTO80
READY.


POSITION A%( 1, 1, 1, 1, 1 ) = 1
POSITION A%( 1, 1, 1, 1, 2 ) = 2
POSITION A%( 1, 1, 1, 1, 3 ) = 3
POSITION A%( 1, 1, 1, 1, 4 ) = 4
POSITION A%( 1, 1, 2, 2, 1 ) = 5
POSITION A%( 1, 1, 1, 2, 2 ) = 6
POSITION A%( 1, 1, 1, 3, 2 ) = 10
POSITION A%( 3, 3, 3, 3, 3 ) = 683
POSITION A%( 3, 3, 3, 3, 4 ) = 684
POSITION A%( 3, 3, 3, 4, 4 ) = 688
POSITION A%( 4, 3, 3, 3, 4 ) = 940
POSITION A%( 4, 4, 4, 4, 4 ) = 1024
POSITION A%( 4, 4, 4, 4, 3 ) = 1023
POSITION A%( 4, 4, 4, 4, 1 ) = 1021
```

Plotting Multiple Functions

1. INTRODUCTION

The following program is a way to print a number of graphs on the same set of axes, using the Commodore 3022 printer. This method is quicker than some I've seen which uses carriage returns to make multiple passes along the same print line.

The basic idea is to hold, in the computer, an image of the print line. This is created using a one dimensional array with one element of each position along the print line.

To start off with, the array contains 79 spaces. As each function is evaluated it's graphic plotting symbol is placed in the appropriate element. After the last function is evaluated the array is output, element by element, as one line on the printer. This enables us to plot say twelve. After each line has been printed the array is again filled with 79 spaces.

2. EXPLANATION OF PROGRAM LISTING

Block 10 contains 4 "REM" statements identifying the program.

Block 100 alters the vertical print spacing so that each line

```
10 REM COMBINED GRAPHS
20 REM NEIL TWEATS, CBM TRAINING
30 REM 28.7.81
99 REM"

100 REM SET UP
110 DIM PL$(78)
120 OPEN 1,4
130 OPEN4,4,6:PRINT#4,CHR$(18):CLOSE4
199 REM "

300 REM CALC VALUES FOR GIVEN X
310 FOR X = 0.40 TO -0.40 STEP -0.05
311 REM "

312 REM CLEAR 'PRINT BUFFER'
314 FOR I = 0 TO 78
316 PL$(I) = " " : NEXT I
318 PL$(39)="+" : REM DRAW X AXIS
319 REM "

330 REM DRAW Y AXIS
332 IF ABS(X)>0.0001 THEN 350
334 FOR I=0 TO 78
336 PL$(I)="+":NEXT I
339 REM"

350 REM HYPERBOLA
352 Y = INT(1/X+12)
354 GOSUB 1000 : PL$(Y+39) = "*"
359 REM"

360 REM PARABOLA
362 Y=INT (100*X↑2-4)
364 GOSUB 1000 : PL$(Y+39) = "+"
369 REM"

370 REM STRAIGHT LINE
372 Y = INT ((-100*X+12)/3)
374 GOSUB 1000 : PL$(Y+39) = "●"
379 REM "

380 REM OUTPUT 1 LINE, COMBINED GRAPH
382 FOR K = 0 TO 78
384 PRINT#1, PL$(K);
386 NEXT K
388 PRINT#1
390 NEXT X
399 REM"

400 REM PRINT KEY
410 OPEN 4,4,6: PRINT#4, CHR$(24) : CLOSE 4
420 PRINT#1, "Y =    1/X    + 12....*
430 PRINT#1, "Y =  100*X↑2 -  4....+
440 PRINT#1, "Y = -100*X    + 12....●
450 PRINT#1
460 PRINT#1, "THE Y AXIS IS HORIZONTAL"
499 REM "

500 REM CLOSE DOWN
510 CLOSE 1 : END
599 REM "

1000 REM COMPRESS LIMIT
```

```
1010 IF ABS(Y)>39 THEN Y=39*SGN(Y)
1030 RETURN
1099 REM"

60000 REM SPUD
60010 SAVE"@0:GR1",8
60020 VERIFY"0:GR1",8
READY.
```

touches the previous one. In this way, we can print the vertical X axis as one continual line, instead of a series of separate characters. We also open up a normal print channel to the printer. Finally we set up an array of 79 elements: one for each position along the print line.

Block 300 sets up the range of values that "X" will have: in the this case +0.4 to -0.4. It also sets up the number of different co-ordinates that are to be used. This can be altered by changing the "STEP".

Block 312 is used to load the "PRINT BUFFER" with 79 spaces. In the 39th element is put the character for the "X" axis.

Block 330 is there to print the "Y" axis. This only happens if the value of "X"is equal to zero, but due to the nine digit accuracy of the PET an exact value of zero may never be reached. So ABS is used to check if X is nearly zero.

Blocks 350, 360 and 370, set up the co-ordinates for three example function usng the following equations:-

a. Y=1/ X+12
b. Y=100X↑2-4
c. Y=(−100x+12)/3

: FOR A HYPERBOLA
: FOR A PARABOLA
: FOR A STRAIGHT LINE

Block 380 is used to print out whatever is in the print image array, in this case four different characters which represent one symbol of each function and one part of the "X" axis. Should two or more functions plot into the same space only the last will be printed.

Block 400 prints out the "KEY", relating graph plotting symbols to their functions.

Block 500 closes down the program properly.

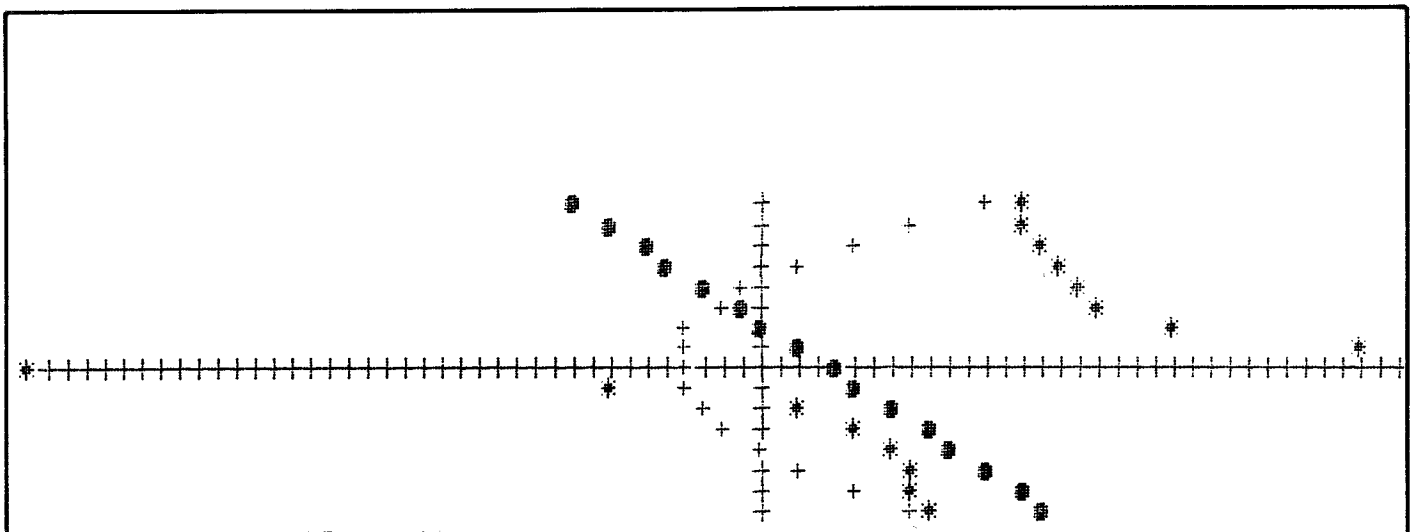
Block 1000 is a subroutine which is designed to check that all the values used are within the limits allowed by the printer.

3. SMOOTHER CURVES

This program used full print positons for each point plotted and the curves are not smooth. It is possible to get smoother curves by using the 3022 programable character to print one matrix dot per point. A two dimensional array is required, 7 deep and 480 wide. First one fills the array in the manner described above for 7 consecutive values of X. In this case just a one for a point and a zero for no point is required. The print routine then takes 'chunks' of the array 6 wide and 7 deep and prints each as a programmable character.

Why not give it a try and let me know how you get on.

Neil Tweats.
Commodore Training
Department.



Peripheral Spot

This month we turn our attention to the Commodore 8010 modem. The area of communications is growing in importance all the time, and so it is vital to publish as much information as possible on this subject as the number of communications users continues to increase. One of the articles, from Paul Higginbottom of Commodore, concerns early work that he carried out on the 8010 and the discoveries he made during that work.

The other program is from Jim Butterfield, and is designed to get you on the air using an 8010. Type in the Basic program first, and then the machine code part (enter the monitor with SYS4 as usual, and just type it all in), and then save it all in the normal way. As with all programs involving machine code I strongly recommend saving first, as it is notoriously easy to crash these sort of programs by one incorrectly entered character. Once you're happy, away you go!

```
07E8 00 A7 A9 00 85 B1 A9 00
07F0 85 B1 A9 0E 8D 4C E8 A5
07F8 9B C9 EF D0 01 60 A2 00
0800 A9 00 85 B1 A5 9B C9 EF
0808 D0 01 60 A2 00 86 A7 A2
0810 05 20 C6 FF 20 58 09 A6
0818 96 D0 72 A6 D1 F0 0D 48
0820 A2 08 20 C9 FF 20 D2 FF
0828 20 CC FF 68 A6 B2 F0 57
0830 29 7F C9 7F F0 CE C9 1F
0838 B0 3B C9 0D D0 0B E6 A7
0840 A9 20 20 D2 FF A9 0D D0
0848 3E C9 08 D0 04 A9 14 D0
0850 36 C9 0C D0 04 A9 93 D0
0858 2E C9 13 D0 08 A9 00 85
0860 B4 85 D1 F0 9F C5 B7 F0
0868 9B C9 11 D0 97 A5 B3 85
0870 B4 10 91 30 8F C9 61 90
0878 04 29 5F D0 0A C9 41 90
0880 06 C9 5B B0 02 09 80 20
0888 D2 FF 4C 04 08 A2 05 20
0890 C9 FF 2C 40 E8 08 20 CC
0898 FF 28 50 EE A6 B9 F0 0C
08A0 A9 0A E4 B8 F0 02 A9 7F
08A8 C6 B9 10 24 20 E4 FF D0
08B0 22 A6 B4 F0 B2 A2 07 20
08B8 C6 FF 20 E4 FF 48 20 CC
08C0 FF 68 A6 96 F0 06 A2 00
08C8 86 B3 86 B4 A6 B6 D0 30
08D0 4C 38 09 C9 9D D0 06 A5
08D8 B3 85 B4 10 94 C9 91 D0
08E0 06 A5 D0 85 D1 10 8A C9
0910 10 26 C9 12 D0 05 E6 B1
0918 4C 04 08 C9 0D F0 19 C9
0920 41 90 06 C9 5B B0 02 09
0928 20 29 7F C9 20 B0 09 C9
0930 14 F0 03 4C 87 08 A9 08
0938 A2 05 20 C9 FF 85 B7 20
0940 D2 FF C9 0D D0 0C A6 B5
0948 F0 06 A6 B8 86 B9 10 02
0950 86 B4 20 CC FF 4C 04 08
0958 A9 34 8D 21 E8 AD 40 E8
0960 09 02 8D 40 E8 A9 28 8D
0968 45 E8 2C 4D E8 70 1F 2C
0970 40 E8 30 F6 AD 40 E8 29
0978 FD 8D 40 E8 AD 20 E8 49
0980 FF 48 A9 3C 8D 21 E8 2C
0988 40 E8 10 FB 30 14 AD 40
0990 E8 29 FD 8D 40 E8 2C 40
0998 E8 10 E1 A5 96 09 01 85
09A0 96 48 20 CC FF 68 60 AA
08E8 1D D0 06 A9 00 85 B4 F0
08F0 C2 C9 11 D0 06 A9 00 85
08F8 D1 F0 F4 C9 94 D0 01 60
0900 4C 03 09 A6 B2 F0 31 A6
0908 B1 F0 07 A8 29 1F 46 B1
```

```
        CBM NOTES - JIM BUTTERFIELD


100 PRINT "[CLR] MODEM I/0      JIM BUTTERFIELD[CD]"
110 FORJ=178TO182:POKEJ,0:NEXTJ:POKE181,1:T$="PRG,"
120 INPUT"LINE FORMAT: ASCII OR PET  A[3CL]";Z$:IFASC(Z$)=80GOTO160
130 POKE59468,14
140 POKE178,1:T$="":INPUT"LINE FEEDS  N[3CL]";Z$:IFASC(Z$)=78GOTO160
150 F9=ASC(RIGHT$(Z$,1))-48:IFF9<0ORF9>9THENF9=1
160 PRINT"SEND FROM DISK FILE (";T$;"SEQ,NO)  N[3CL]";:INPUTZ$
170 Z$=LEFT$(Z$,1):IFZ$="N"GOTO240
180 IFZ$<>"P"ANDZ$<>"S"GOTO160
190 INPUT "INPUT FILE NAME";I$
200 I=1:POKE179,7:OPEN7,8,7,I$+","+Z$
210 IFT$<>""GOTO240
220 INPUT"FILE FORMAT:  ASCII OR PET  P[3CL]";Z$:IFASC(Z$)<>65THENPOKE182,1
230 IFF9=0THENINPUT"LINE-AT-A-TIME  Y[3CL]";Z$:IFASC(Z$)=89THENPOKE181,0
240 PRINT"OUTPUT TO DISK (";T$;"SEQ,NO)  N[3CL]";:INPUTZ$
250 Z$=LEFT$(Z$,1):IFZ$="N"GOTO310
260 IFZ$<>"S"ANDZ$<>"P"GOTO240
270 F=8:INPUT"DISK FILE NAME";N$
280 OPEN8,8,8,"0:"+N$+","+Z$+",W"
290 PRINT "[CD] PRESS CURSOR-UP TO ENGAGE DISK LOG;"
300 PRINT " PRESS CURSOR-DOWN TO DISENGAGE"
310 IFI=0GOTO340
320 PRINT"[CD]PRESS CURSOR-LEFT TO START TRANSMITTING"
330 PRINT"FROM TAPE."
340 PRINT "[CD]PRESS INST TO QUIT."
350 OPEN5,5:PRINT"[CD] YOU'RE ON THE AIR![CD]"
360 POKE208,F:POKE209,0:POKE184,F9:SYS2048
370 IFI=1THENCLOSE7
380 IFF=8THENPRINT#8,"***":CLOSE8
390 CLOSE5
READY.
```

The manual that is supplied with the 8010 modem is fairly good, but I found I still had to read between the lines to really get it right.

My objective was to be able to have two PETs connected by modem, that could freely talk with each other. I intended to implement this by having a program in each PET which sends characters typed in to the modem, and echoes them to the screen while displaying any received characters in reverse to distinguish them, from the operators text.

The first program that I wrote was based on what I had read in the manual, that is if ST (the status variable) is zero, a character is pending and so I wrote a program like this:-

```
100 OPEN 5,5
110 GET#5,A$:IF ST<>0
    THEN
    PRINT"(RVS)"A$"(OF-
    FRVS)";
120 GETA$:IF A$<>""THEN
    PRINT#5,A$;:PRINT A$:
130 GOTO 110,0:
```

This program was totally unacceptable, because it dropped 50% of all transmitted characters. However I had read about the SRQ method of using the modem. SRQ is the main reason why the PET IEEE is not a real IEEE-488 bus. It is a real line on the bus in the PET, but is not used. SRQ stands for service request, and all it does is to allow a peripheral to

tell a controller that it requires servicing. So the second program:-

```
100 OPEN 5,5,
110 IF PEEK(59427)AND128
    THEN GET#5,A$
    :POKE
    59426,0:PRINT"(RVS)-
    "A$"(OFFRVS)";
120 GETA$:IF A$<>"" THEN
    PRINT#5,A$;:PRINT A$;
130 GOTO 110
```

This worked a treat. Line 110 (which is the important one), says "IF the modem needs servicing then service it : tell it, it has been serviced : display character from modem ".

Once I had discovered how easy it was to communicate via the 8010, I decided to add a little style. And so the third program evolved from my fingertips 20 minutes later. This program uses the window facility of the 8032, and so it will not work on 40 column machines I'm afraid. The top window displays characters typed locally and the bottom window shows remote activity. I plan to add peripheral transmit and receive file capability, whereby any device can be toggled on or off. This will be published in a future magazine.

```
8032 Dual Window Modem Communicator


1000 OPEN 5,5:PRINT"[HM HM CLR]"
1010 PRINT"[CLR]"
1020 FOR I=0 TO 79:POKE 33728+I,64:NEXT
1030 CD$="[DN DN DN DN DN DN DN DN DN DN DN DN]"
1040 X1=0:Y1=0:X2=0:Y2=0
1050 SI=59427:MS=128:IO=59426:ZE=0
1060 TL=224:ML=13:CP=198:CL=216:BL=225:HF=232
1070 LL=80:S1=32768:S2=33808:US=127:BP=11
1080 POKE S1,160:POKE S2,160:ES$=CHR$(27)
2000 IF NOT PEEK(SI) AND MS THEN 3000
2010 GET#5,A$:POKE IO,ZE:PRINT"[HM HM]";:POKE HF,ZE:POKE TL,ML
2020 P=S2+X2+Y2*LL:POKE P,PEEK(P)ANDUS
2030 PRINT"[HM]"LEFT$(CD$,Y2)SPC(X2):POKE HF,ZE:PRINTA$ES$;
2040 X2=PEEK(CP):Y2=PEEK(CL)-ML
2050 P=S2+X2+Y2*LL:POKE P,PEEK(P)ORMS
3000 GETA$:IF A$="" THEN 2000
3010 PRINT"[HM HM]";:POKE BL,BP:POKE HF,ZE
3020 P=S1+X1+Y1*LL:POKE P,PEEK(P)ANDUS
3030 PRINT"[HM]"LEFT$(CD$,Y1)SPC(X1):POKE HF,ZE:PRINTA$ES$;
3040 X1=PEEK(CP):Y1=PEEK(CL)
3050 P=S1+X1+Y1*LL:POKE P,PEEK(P)ORMS
3060 PRINT#5,A$;:GOTO 2000
```

Machine Code Programming

```
                STRING FLIP
                OLD ROM PET'S WITH CBM PRINTERS

036C            BTMSTR  *       $0082   BOTTOM OF STRING SPACE
                ($0030 WITH 3.0 ROM'S)
036C            HIMEM   *       $0086   TOP OF RAM MEMORY
                ($0034 WITH 3.0 ROM'S)
036C            POINTR  *       $0001   INDIRECT POINTER

033A                    ORG     $033A   SECOND CASSETTE BUFFER

033A A0 00              LDY     #$00    CLEAR INDEX
033C A5 82              LDA     BTMSTR  COPY BOTTOM OF
033E 85 01              STA     POINTR  STRINGS ADDRESS
0340 A5 83              LDA     BTMSTR  +01 INTO POINTER
0342 85 02              STA     POINTR  +01 LOCATION.

0344 A5 02      DONE    LDA     POINTR  +01 CHECK WHETHER
0346 C5 87              CMP     HIMEM   +01 THE POINTER HAS
0348 30 07              BMI     NOTDON  REACHED THE
034A A5 01              LDA     POINTR  TOP OF RAM
034C C5 86              CMP     HIMEM   MEMORY YET.
034E D0 01              BNE     NOTDON  IF SO,
0350 60                 RTS             RETURN TO BASIC.

0351 B1 01      NOTDON  LDA     (POINTR),Y LOOK AT THE
0353 AA                 TAX             NEXT STRING CHARACTER.
0354 29 7F              AND     #$7F    IGNORE ITS CASE FOR NOW.
0356 C9 41              CMP     #$41    IS IT A LETTER?
0358 90 09              BCC     UPPNTR
035A C9 5B              CMP     #$5B    IF NOT,
035C B0 05              BCS     UPPNTR  DON'T CHANGE IT.
035E 8A                 TXA             REMEMBER THE CASE.
035F 49 80              EOR     #$80    FLIP CASE, & PUT BYTE
0361 91 01              STA     (POINTR),Y BACK IN STRING AREA.

0363 E6 01      UPPNTR  INC     POINTR  UP THE POINTER
0365 D0 DD              BNE     DONE    & LOOP BACK.
0367 E6 02              INC     POINTR  +01
0369 D0 D9              BNE     DONE
036B 00                 BRK             CAN'T GET HERE!
```

PET String Flip

This routine solves the problem of upper and lower case inversion when using the CBM 3022 printer. The method is to invert the characters in the string area of RAM.

PET owners with a Commodore CBM 3022 printer may have a problem. The printers do a fine job with lowercase and uppercase printing. However, what appears on the screen as uppercase comes out on the printer in lowercase, and vice versa. This is due to the non-standard ASCII codes used by the PET.

STRING FLIP offers a solution with this short program, just 50 bytes long. It is entirely relocatable. It inverts the bit of each byte that indicates uppercase or lowercase. It does this throughout the part of memory known to hold string data. However, it only changes those

values within the range of the alphabet. This allows the user to invert most data before sending it to the printer. After printing, the routine may be called again to flip the data back to the normal screen form.

I use STRING FLIP this way: load STRING FLIP, alone or without a program as data; define a variable, flip, to remember we've flipped, then before each printer routine, have a line such as

```
300 IF FLIP<>1THENSYS(826):
FLIP=1: REM INVERT CASE
```

Then on return to screen mode, have a similar line:

```
100 IF FLIP<>0THENSYS(826 :
FLIP=0: REM NORMAL CASE
```

This routine should correct most of your printouts. If you find a string that still prints incorrectly, most likely it is defined within the BASIC program, rather than in the string area. You can correct this by redefining it, as in this example:

```
200a$="Sorry, Try Again":a$=
a$+""
```

Enjoy STRING FLIP in all your word-processing and data-handling programs.

Restricted Access

The program below can be used to improve the security of any m/c program that doesn't make use of the 2nd cassette buffer. Most m/c programs are entered by a 'BASIC' one line statement e.g. 10 SYS1029 and by altering this to SYS826 and (having loaded in the new program) changing locations 03F5 & 03F6 with the low and high byte values to return to the original program (at decimal 1029 in our example) the password program is patched in. To 'SAVE' both programs one should jump into monitor (SYS1024) and .S "TEST",01,33A, XXXX where 'XXXX' is the top of the original program in hex. The program, then, has the following feature;

a) A 5-character password ('INPUG') in the listing.

b) All characters are checked after being input giving over 34 Billion combinations!

c) There are only 3 attempts allowed at entry before the screen is cleared and the PET 'hang up'.

d) An 'escape' key is provided for the above event (an '*') to recover control after an unauthorised access -the password must be re-entered in full though.

e) Characters input are not echoed to the screen. To alter the password, change locations $03D4-$03D8 to the relevant hex-encoded ASCII values.

Similarly, the 'escape' character can be changed by altering the value of $03A8; and the no. of tries at entry permissable by modifying location $039C. The following listing is for NEW ROM PETs.

```
033A A9 00 8D CE 03 A9 FF 8D
0342 F7 03 EA 20 C4 03 EA EA
034A EA EA EA EA EA EA EA EA
0352 A0 03 A9 D9 20 1C CA 20
035A E4 FF F0 FB EE F7 03 AE
0362 F7 03 9D CF 03 8A C9 04
036A D0 ED A2 FF E8 BD CF 03
0372 DD D4 03 D0 0B 8A C9 04
037A D0 F2 20 C4 03 4C F4 03
0382 EA EA EA 20 E2 C9 20 E2
038A C9 A0 03 A9 E3 20 1C CA
0392 20 AE 03 EE CE 03 AD CE
039A 03 C9 03 D0 52 20 C4 03
03A2 20 E4 FF F0 FB C9 2A D0
03AA F7 4C 3A 03 EE CB 03 AD
03B2 CB 03 C9 00 D0 F6 EE CC
03BA 03 AD CC 03 C9 00 D0 EC
03C2 EA 60 20 46 E2 20 46 E2
03CA 60 00 00 00 01 49 50 55
03D2 47 2E 49 50 55 47 2E 50
03DA 41 53 53 57 4F 52 44 3F
03E2 00 49 4C 4C 45 47 41 4C
03EA 20 45 4E 54 52 59 00 4C
03F2 3F 03 4C 89 C3 04 00 00
```

Disk Use for Beginners

David J. Pocock

Hello again, welcome to part X (I can't keep count!). This month we turn our attention to DIRECT ACCESS (come back, it's not that bad). I make no apologies for the fact that I have drawn most of my information from an article by Mike Gross-Niklaus in a much earlier newsletter. I feel that this is justified by the fact that it is one of the best articles on the subject, and that also I was involved with the original production of the article for the Training Department (which I am very glad to see is back in operation).

I will give details of Direct Access using the "Block" commands this month. Next month RELative files for 4040 and 8050 users only (I've got to do it sometime). In issue eleven (I could go on for a very long time) I will provide you with some interesting things to play around with and think about.

As with last time I will explain the various sections of a Direct Access system, and at the end of this article you will find a program listing (which I hope is easier to understand than Random 1.0 from your disk manual!).

NOTE 1 : In last month's newsletter the article "Dossing around with Basic", section 7, contained a report on Block Allocate on the 8050. As we are covering Block Allocate this month I had to test a fault mentioned in it (I hadn't heard of this one). During my own tests I could not generate the reported problem. This does not mean however that there is no problem. More information about this when the author (David Middleton) returns from America (possible this month).

REMEMBER, always SAVE the program before running it.

What is Direct Access

So, what is Direct Access? Direct Access (D-A) is a method of accessing the data stored on disk in such a way that you go DIRECTLY to where the data is stored. With sequential you have to search through the file until the information is found.

If, for example, you had a list of stock items on a sequential file, and you wanted to look at item 565 you would have to read through 564 sets of data before you came to the information you required. If you then wanted to look at item 564 you would have to go back to the start. If however you were using a D-A file you could move directly to item 565 and then directly to item 564.

Don't get me wrong, sequential files are very useful for items which will always be accessed in the same order. For example, Personnel information for payroll processing, general mailing lists (like the list of subscribers to this newsletter) etcetera.

How to use it

Okay, so now you know what D-A is for, how do you go about using it ?

There are several ways of managing a D-A system, but the method I intend to show lets the Disk Operating System keep track of where information is stored and where you can put more information. All you have to keep track of is an index of what information is where on the disk. The DOS will tell you this. We will only be putting one record per block.

First of all, a few things used within this article, and other places, which will help.

DOS - Disk Operating System, the ROM within the disk unit.

FIELD - A single item of information, e.g. a name or telephone number.

RECORD - A group of related fields, e.g. name, address, telephone, age etc.

BLOCK - A section on the disk, also called a Sector.

BUFFER - An area of memory within the disk unit, not on the diskette.

BAM - Block Availbility Map, which shows the DOS which sections of a diskette are in use.

CHANNEL - A route between the PET and disk unit's RAM, which does not refer to any particular diskette.

NF - Is the file number.

CH - Is the channel number.

DR - Is the drive number (0 or 1).

The D-A file

Commodore 3040 disk units do not have direct access capability, unlike 4040s and 8050s which have relative files (a similar concept). All Commodore disk units (3040,4040,8050) support "Block" commands which enable you to write data to any specified section of the disk.

To create or use a D-A file you must have the command channel open, so you must use OPEN 15,8,15 somewhere near the start of the program.

You will also need to open a channel to the disk unit. Channels are specified by the secondary address and can range from 0 to 15. However, 0 and 1 are already in use for Load and Save, 15 is the command channel and 14 does not exist on DOS 2.X. I personally tend to use channels 2 and 3 for D-A work, and so to open a channel I use OPEN 2,8,2, "#". Note that the file number and the channel number (Secondary Address) need not be the same, although they are here. For example, OPEN5,8,2, "#" would also be valid. You can also have two files open to the same channel, although I do not advise this. The "#" in place of a file name tells the DOS that you wish to allocate a buffer for use with this file.

NOTE 2 : With D-A both the file number and the channel (secondary address) numbers are used for different things. I find it much easier (there are less numbers to remember!) if both the file and channel numbers are the same. I will tell you which is used where, but respectfully suggest that you adopt the easier method of keeping the numbers the same.

I will assume that the string array F$( ) contains the fields for the record (see program). Because of the way the relative file system works I tend to compile a single string before sending it to the file, this makes for more standardised coding. So :-

```
FOR A = 0 TO F : S$ = S$ + F$(F)
+ CR$ : NEXT
```

wher S$ is null to start, and the complete record at the end.

F is the number of fields (numbered 0 upwards).

CR$ = CHR$(13), a RETURN character.

Then, to send S$ to the file :-
PRINT#NF,S$; (remember we have put the necessary carrige returns within S$) : in this case NF=2.

At this stage the data is not written onto the diskette but only into the buffer within the disk unit, and in actual fact this is not the first thing to do : it is only first here because you've seen it before.

Before we can put the information onto the diskette we must first find some empty space on it, and to do this we use a short routine which attempts to allocate space on the disk.

```
11010   DR=1 : T=1 : S=0
11020   PRINT#15,"B-A";DR;T;S
11030   GOSUB 10000
11040   IF EN = 0 THEN
        RETURN
11050   T = ET : S = ES :
        GOTO 11020
```

(11010) sets pointers to track (T) 1 and sector (S) 0.
(11020) attempts to allocate this block on drive number dr.
(11030) uses our standard error checking routine ; we will also check for disk full etc.
(11040) if the error is 0 (ok) we have found a useable block; it has now been allocated and we can go back to the main program.
(11050) the error was not 0, the error track and error sector now contain the first vacant track and sector. The routine then goes back to 11020 to allocate this block.

Now that we (or the program at least) know where there is some free space, if any, we can go about writing to it. Firstly, set the buffer position to position 1 (this may not always be 1 : it depends on how you're using the system). Then write the record into the buffer (PRINT#) and finally write the buffer onto the disk. And so, to write a record to the disk :-

```
2120    GOSUM 11000 : REM
        FIND A FREE BLOCK
2130    PRINT#15,"B-P";CH;1 :
        REM SET BUFFER
        POINTER TO POSI-
        TION 1
2140    PRINT#NF,S$; : REM
        WRITE DATA TO BUF-
        FER
2150    PRINT#15,"U2";CH;
        DR;T;S : REM WRITE
        BUFFER TO
        DISKETTE
```

(NF=file number (2) and CH=channel number(2))

The data is now on the disk. However, on 3040 and 4040 drives the BAM is only updated when a disk file is closed. Unless you are writing a lot of information in quick succession it is a good idea to close the file after each write operation. If you are in an interactive system where data is coming from the keyboard, the delay for OPEN and CLOSE is not important (it is still faster than you are).

What data has gone where ? We need to keep track of what information has been put into which block on the disk. To do this we need some form of index : in this example the index will be an array in memory, which is written to the disk as a sequential file. If the index array is a two dimensional string array I$(,), then add :-

```
2170 I$(X,0) = F$(0) : I$(0) : I$(X,1)
= STR$(T) : I$(X,2) = STR$(S)
```

Where X is the number of the record, and assuming that F$(0) is to be the key for that record.

And so, for record X, I$(X,0), I$(X,1), I$(X,2) will contain :-
Key, Track, Sector,
This index must be written to disk in some form or other, if it isn't your index will be lost when you close the system down. It is possible, but not easy, to recreate an index from the data on a file, provided that the key is stored as part of the record. I will not be showing you how to do this (not this month anyway).

Now that we have added the key, track and sector to the index we must increment the index pointer (X) so that the next key does not overwrite this one. So :-

```
2180 X=X+1
```

Reading the data back from the disk is just as straightforward. To read back a record you must first search the index, and find out the track and sector of the required record. Assuming track and sector are now in the variables T and S respectively :-

```
3120    PRINT#15,"U1";CH;
        DR;T;S
3130    PRINT#15,"B-P";CH;1
3140    FOR A = 0 TO F : IN-
        PUT#NF,F$(A) : NEXT
```

(3120) this command reads track T and sector S from drive DR into the buffer associated with channel CH
(3130) this moves the buffer pointer for channel CH to position 1, which is the first character of the first field.
(3140) this reads the information from file number NF (or rather its associated buffer CH) into BASIC variables F$( ). It will read F+1 fields, the same number as were sent to the disk.

Note that the same limitations apply when reading D-A as with sequential, namely no null fields, and no fields may be more than 80 characters long. There is an added limitation that a record may not exceed 255 bytes (1 block = 256 bytes, position 0 is used by the DOS with some of the block commands). It is possible to access all 256 bytes, and this will be covered in a later article.

In this article I have "U1" to read a block and "U2" to write a block. These command are alternatives to "B-R" (Block Read) and "B-W" (Block Write). "B-R" and "B-W" operate in a slightly different manner to "U1" and "U2" in that they use position 0 (zero) of a Block to tell the DOS how many characters are in the block. (More about that in issue eleven).

NOTE : that "B-W" does not work on 4040's and so you must use the "U2" alternative instead.

And so with that, good bye for now, and overleaf is the program.

```
D-A EXAMPLE PROG        PAGE 1

  10 REM D-A EXAMPLE PROGRAM
  20 REM BY DAVID J. POCOCK
 100 REM  DR = DRIVE NUMBER
 110 REM NF = FILE NUMBER
 120 REM CH = CHANNEL NUMBER
 130 REM T,S = TRACK, SECTOR
 140 REM X = INDEX POINTER
 150 REM F = NUMBER OF FIELDS (ADD 1)
 160 REM D$ = CURSOR DOWNS
 200 REM SETUP ARRAY AND START VARIABLES
 210 T=1:S=0:DR=1:NF=2:CH=2:X=0:F=3:CR$=CHR$(13)
 220 D$="[CRD][CRD][CRD][CRD][CRD][CRD][CRD][CRD][CRD][CRD][CRD][CRD][CRD
     ][CRD][CRD][CRD][CRD][CRD][CRD][CRD]"
 230 DIM I$(20,2)   REM MAX 21 RECORDS IN AN INDEX
 240 DIM F$(F)   REM MAX F FIELDS PER RECORD
 250 OPEN15,8,15   REM OPEN COMMAND CHANNEL TO DISK

1000 REM MENU
1010 PRINT"[HMS][HMS][CLS]DIRECT ACCESS EXAMPLE"
1020 PRINT"[CRD][CRD]1.  ADD A RECORD"
1030 PRINT"[CRD][CRD]2.  LOOK AT A RECORD"
1040 PRINT"[CRD][CRD]3.  WRITE INDEX TO DISK"
1050 PRINT"[CRD][CRD]4.  READ INDEX FROM DISK"
1060 PRINT"[CRD][CRD]5.  LIST INDEX ON SCREEN"
1070 PRINT"[CRD][CRD]6.  AMEND A RECORD"
1080 PRINT"[CRD][CRD]7.  QUIT"

1100 REM SELECT OPTION
1110 GETA$:IFA$=""THEN1110
1120 IFA$<"1"ORA$>"7"THEN1110
1130 IFA$="7"THEN1200
1140 ON VAL(A$) GOSUB 2000,3000,4000,5000,6000,7000
1150 GOTO1000

1200 REM CLOSE DOWN SYSTEM
1210 CLOSE15:END

2000 REM ADD A D-A RECORD
2010 REM GET INFORMATION AND COMPILE STRING FOR DISK
2020 IFX>20THEN2200
2030 PRINT"[HMS][HMS][CLS]ADD A D-A RECORD"
2040 FOR A=0 TO F:PRINT"[CRD]FIELD";A;
2050 INPUT"CONTENTS ";F$(A)
2060 NEXT A
2070 S$="":FOR A=0 TO F:S$=S$+F$(A)+CR$:NEXT A

2100 REM OPEN CHANNEL D-A CHANNEL AND WRITE RECORD TO DISK
2110 DR=1:NF=2:CH=2:OPEN NF,8,CH,"#": REM OPEN D-A CHANNEL
2120 GOSUB11000 : REM FIND SPACE ON DISK
2130 PRINT#15,"B-P";CH;1
2140 PRINT#NF,S$;
2150 PRINT#15,"U2";CH;DR;T;S
2160 CLOSE NF
2170 I$(X,0)=F$(0):I$(X,1)=STR$(T):I$(X,2)=STR$(S)
2180 X=X+1:RETURN

2200 REM INDEX FULL
2210 PRINT"[CRD][CRD][RVS]INDEX IS FULL"
2220 FORA=0TO2000:NEXT:RETURN

D-A EXAMPLE PROG        PAGE 2

3000 REM READ A D-A RECORD
3010 REM GET KEY AND SEARCH
3020 IFX=0THEN4100
3030 PRINT"[HMS][HMS][CLS]READ A RECORD":R=-1
3040 INPUT"[CRD][CRD]KEY (FIELD 0) ";KY$
3050 FORA=0TOX-1:IFKY$=I$(A,0)THENR=A:A=X
3060 NEXT A:IF R<0 THEN 3200
3070 T=VAL(I$(R,1)):S=VAL(I$(R,2)):REM GET TRACK AND SECTOR FROM INDEX

3100 REM READ AND DISPLAY RECORD
3110 DR=1:NF=3:CH=3:OPEN NF,8,CH,"#"
3120 PRINT#15,"U1";CH;DR;T;S
3130 PRINT#15,"B-P";CH;1
3140 FOR A=0 TO F
3150 INPUT#NF,F$(A):PRINT"[CRD]FIELD";A;F$(A)
3160 NEXT A: CLOSE NF
3170 GOSUB12000:RETURN

3200 REM KEY NOT FOUND
3210 PRINT"[HMS][CRD][CRD][CRD][CRD][CRD][RVS]KEY NOT IN INDEX"
3220 FORA=0TO2000:NEXT:RETURN

4000 REM WRITE INDEX TO DISK
4010 IFX=0THEN4100
4020 INPUT"[HMS][HMS][CLS][HMS][HMS]NAME OF INDEX TO WRITE ";IN$:IN$=LEFT
     $(IN$,16)
4030 PRINT"[HMS][HMS][CLS][CRD][CRD]WRITING INDEX TO DISK"
4040 DR=0:NF=4:CH=4:OPEN NF,8,CH,"0:"+IN$+",S,W"
4050 GOSUB10000:IFEN<>0THEN4000
4060 PRINT#NF,X;CR$;
4070 FORA=0TOX-1:FORB=0TO2
4080 PRINT#NF,I$(A,B);CR$;
4090 NEXTB,A:CLOSE NF:GOSUB10000:RETURN

4100 REM NO INDEX
4110 PRINT"[HMS][HMS][CLS][CRD][CRD][RVS]NO INDEX IN MEMORY"
4120 FORA=0TO2000:NEXT:RETURN

5000 REM READ INDEX FROM DISK
5010 INPUT"[HMS][HMS][CLS][CRD][CRD]NAME OF INDEX TO READ ";IN$:IN$=LEFT$
     (IN$,16)
5020 PRINT"[HMS][HMS][CLS][CRD][CRD]READING INDEX FROM DISK"
5030 DR=0:NF=5:CH=5:OPEN NF,8,CH,"0:"+IN$+",S,R"
5040 GOSUB10000:IFEN<>0THENRETURN
5050 INPUT#NF,X
5060 FORA=0TOX-1:FORB=0TO2
5070 INPUT#NF,I$(A,B)
5080 NEXTB,A
5090 CLOSE NF:GOSUB10000:RETURN

6000 REM LIST INDEX TO SCREEN
6010 PRINT"[HMS][HMS][CLS]"
6020 FORA=0TOX-1:PRINTI$(A,0),I$(A,1),I$(A,2):NEXT
6030 GOSUB12000:RETURN

7000 REM AMEND RECORD
7010 REM GET KEY AND SEARCH
7020 IFX=0THEN4100
7030 PRINT"[HMS][HMS][CLS]AMEND A RECORD":R=-1
7040 INPUT"[CRD][CRD]KEY (FIELD 0) ";KY$
7050 FORA=0TOX-1:IFKY$=I$(A,0)THENR=A:A=X

D-A EXAMPLE PROG        PAGE 3

7060 NEXT A:IF R<0 THEN 7400
7070 T=VAL(I$(R,1)):S=VAL(I$(R,2)): REM GET TRACK AND SECTOR FROM INDEX

7080 REM READ RECORD
7090 DR=1:NF=6:CH=6:OPEN NF,8,CH,"#"
7100 PRINT#15,"U1";CH;DR;T;S
7110 PRINT#15,"B-P";CH;1
7120 FOR A=0 TO F
7130 INPUT#NF,F$(A)
7140 NEXT A

7200 REM DISPLAY OLD RECORD AND UPDATE
7210 FOR A=0 TO F
7220 PRINT"[HMS][CRD][CRD][CRD][CRD][CRD]";LEFT$(D$,2*A);A;F$(A)
7230 PRINT"[HMS][CRD][CRD][CRD][CRD][CRD]";LEFT$(D$,2*A);"[CRR][CRR][CRR]
     ";:INPUT"ALTER";F$(A)
7240 NEXT A
7250 S$="":FOR A=0 TO F:S$=S$+F$(A)+CR$:NEXT A

7300 REM WRITE RECORD TO DISK
7310 PRINT#15,"B-P";CH;1
7320 PRINT#NF,S$;
7330 PRINT#15,"U2";CH;DR;T;S
7340 CLOSE NF
7350 I$(R,0)=F$(0)
7360 RETURN

7400 REM KEY NOT FOUND
7410 PRINT"[HMS][CRD][CRD][CRD][CRD][CRD][RVS]KEY NOT IN INDEX"
7420 FORA=0TO2000:NEXT:RETURN

10000 REM CHECK ERROR CHANNEL
10010 INPUT#15,EN,EM$,ET,ES:IFEN=0THENRETURN
10020 IF EN=65 THEN 10200
10030 IF EN=72 THEN 10100
10040 IF EN=62 OR EN=63 THEN 10300

10100 REM FATAL DISK ERROR
10110 PRINT"[CLS][CRD][CRD][RVS]DISK ERROR";EN;"[RVF]"EM$
10120 CLOSENF:CLOSE15
10130 END

10200 REM ERROR 65 NO BLOCK
10210 IFET>0THENRETURN
10220 PRINT"[HMS][HMS][CLS][CRD][CRD][RVS]D-A DISK IS FULL"
10230 PRINT"[CRD][CRD]I WILL ATTEMPT TO WRITE THE INDEX":CLOSE NF
10240 PRINT"[CRD]TO DRIVE 0"
10250 GOSUB12000:GOSUB4000:CLOSE15:END

10300 REM FILE EXISTS/DOES NOT EXIST
10310 PRINT"[RVS]";IN$;"   ";EM$
10320 CLOSE NF:GOSUB12000:RETURN

11000 REM FIND SPACE ON DISK
11010 DR=1:T=1:S=0:REM DRIVE 1, TRACK 1, SECTOR 0
11020 PRINT#15,"B-A";DR;T;S
11030 GOSUB10000
11040 IFEN=0THENRETURN
11050 T=ET:S=ES:GOTO11020

12000 REM WAIT FOR SHIFT KEY

D-A EXAMPLE PROG        PAGE 4

12010 PRINT"[CRD][CRD]PRESS [RVS]SHIFT[RVF] KEY TO GO ON"
12020 WAIT152,1:WAIT152,1,1:RETURN
```

Basic Compilers

John Stout

This article is the first of a proposed series which describes how to build a BASIC compiler for the PET. The compiler is to be compatible with the PET's resident BASIC interpreter so that, as with the DTL BASIC compiler currently available, the interpreter can be used to develop a program and the compiler then used to produce a machine code version of the program for speed. Unlike the DTL compiler the final form of the program will be true machine code, rather than a code (B-code?) which is interpreted by a machine code interpreter. This may mean that the compiled program is larger than with the interpreted approach but it should be faster.

The first version of the compiler will attempt no optimisation of the code produced, although this could possibly be added later, and will use a number of 'tricks' to help speed up its development. Firstly it will take as its input a SAVEd version of the program, in the same way as the DTL compiler does. This means that the compiler's scanner, which scans the input for keywords, variable names, operators and constants, need not be as complicated as it might otherwise be, and hence will operate faster. These improvements come about because the BASIC interpreter will already have converted keywords such as THEN, PRINT into their BASIC tokens, 167 for THEN, 153 for PRINT. The second trick will involve the compiler not producing machine code directly but rather an assembly language program file which will be used as input to the PET Assembler and Loader to produce the final executable program. This has the advantages that the compiler need only make one pass through the SAVEd BASIC program (i.e. only read it once), the code can be assembled easily anywhere in memory (or even in separate sections of memory) and the assembly language file can be examined by anyone with sufficient knowledge in order to optimise the code. There are of course two more stages to go through in order to produce the working program, but since the compilation should not be needed very often for any one program this need not be a serious problem.

A number of commands will not be implemented in the first version of the compiler: LIST, LOAD, VERIFY, CONT, CLR, NEW, RUN and STOP. Most of these make no sence for a compiled program, although LOAD (to permit chaining) and CLR might be candidates for later inclusion.

Further details

The compiler is to be written in BASIC so that one of its first tasks will be to compile itself. It would be nice to write it in Pascal or COMAL, but BASIC seems likely to reach a wider audience than Pascal and COMAL might run into problems of both space and implementation, e.g. can COMAL read a program file?

The output assembly language file will contain at its start code to make the program start at location 1025 (just as for a BASIC program), then generate a BASIC line of the form:

```
10 SYS (1039)
```

then at locations 1039 onward would be code to initialise stacks, reset the output and input devices, set variables to zero and so on, then jump to the start of the actual BASIC compiled program, e.g.:

```
JSR INIT
JMP START
(run time library and code for INIT
                follow here)
```

Since we are using the assembler to sort out addresses we can use labels without knowing (or indeed caring) where the final code is to go. After the JMP START instruction will come all the code for the routines that the compiled program will need, e.g. to handle strings, evaluate expressions, index arrays and so on. Since all the code generated so far will not change from compilation to compilation we can leave it in a separate assembly language file called BASICLIB and include it in the assembled version by using the assembler directive:

```
.LIB BASICLIB
```

The compiler will clear its internal arrays used for symbols, line numbers and so on, then open the file which is to be compiled. As it reads each line number it will generate one or more labels, the first of which will have the form:

```
Lhhhh
```

wher hhhh is the 4 digit hexadecimal representation of the line number, e.g. for line 10 the label generated would be:

```
LOOOA
```

Thus a GOTO 10 instruction later on in the program (or even before the line is defined) can be assembled directly to:

```
JMP LOOOA
```

Also at the start of each line will be output a call to a subroutine LINNUM, which is followed by the two bytes of the line number, e.g. again for line 10:

```
LOOOA  JSR LINNUM
       .BYTE 18, 8
```

LINNUM's a job is to set a two byte variable to the line number of the current line being executed. As with the TCL Pascal compiler the BASIC compiler will allow the user to switch this feature off in order to save both time and space.

If the line before the one being currently compiled contained an IF statement then a label of the form:

```
Ihhhh
```

will be generated, where hhhh is the 4 digit hexadecimal representation of a counter which is incremented by 1 at the end of each line. Thus each IF statement has a unique label that it can jump to if the condition in it is false. Thus combining the 2 types of labels so far the BASIC statement in the first line of program:

```
IF X  0 THEN 100
```

would result in the following assembly language code being generated:

```
(code to generate the result of
X  0 in floating point ac-
cumulator 1)
LDA     FP1EXP
BNE     +$03
JMP     +IOOOO
JMP     LOO64
; 0 if false, otherwise non-zero.
;GOTO 100
```

Note that in this case some optimisation could be performed since wecould change the BNE to BEQ and remove the jump to 10000.

In addition to creating the labels Lhhhh the compiler will keep a record of line numbers generated and referenced so that undefined line numbers can be reported at the end of compilation rather than at the assembly stage.

As it reads in each line of BASIC the compiler will list it to the assembly language output file preceded by a semi-colon so that it will be seen as a comment by the assembler. Thus the machine code generated by the complier/assembler combination can be easily matched with the source BASIC line, possibly for hand optimisation.

Variables will be referred to by their name (restricted to two characters in the ouput although any length will be allowed on input) suffixed by I, R or S depending on whether they represent an Integer, Real or String quantity. Space will be generated for the variables at the end of the assembly language file, e.g. with variables A%, BR%, B, CD, F$,

```
AI    .BYTE 0,0
BRI   .BYTE0,0
BR    .BYTE 0,0,0,0,0
CDR   .BYTE 0,0,0,0,0
FS    .BYTE 0,0,0
SMS   .BYTE 0,0,0
```

Strings and their handling can be implemented in a number of ways, but the proposed compiler will probably use the following scheme (or one very like it).

1) String variables will consist of 3 bytes, a length byte and an address pointing to the start of the string. A length byte of 0 means the string has not been defined.

2) A string of length n will occupy n+2 bytes, the first two bytes being a link to the next string of length n, or zero if there are no more strings of that length.

3) Since the strings will occupy memory space in the range $0400-$7FFF the most significant bit of the address/link can be used for other information, and will be set to 1 if the space is not pointed to by a variable.

4) Two 256 byte tables will be maintained, labelled STRL and STRH where the n'th entry in the two tables defines the address (low/high) of the first string space of size n (and will be zero if none exist).

5) If a string has been created (and needs to be stored) of length L then the compiled program need only search the list of spaces of length L for one with its address/link header bit 15 set to 1. This string space can then be filled with the generated string, its used/unused bit set to zero, and the address of the string variables set to point to the space (also its length set to L).

6) If a defined space cannot be found which is big enough then the bottom of string space pointer can be decremented by L+2 bytes and the space so generated linked into the list of strings with length L (with its used/unused bit set to 1). Process 5 can then be executed again.

7) If step 6 generates an out of memory error then the next available space of length L + 1 can be searched for and used, or failing that the next available space of length L + 2 and so on. This means that some one or more bytes will be wasted but the program will continue to run. The space allocated (say of length L+E) will be linked on to the list of spaces of length L.

Although this scheme requires more storage than the one used in the PET it has the benefit of being simple to program and reasonably fast to execute.

Arrays will be implemented by using an array header which will be of length 4+2*N, where N is the number of dimensions in the array. In the 4 bytes the first will be a flag byte, with its most significant bit set to 1 if the array has been defined and the rest of the byte set to 1, 2 or 3 depending on whether the array is of integers, strings or reals. The next byte will contain the number of dimensions and the two after that a pointer to the first array element. The 2*N bytes in the rest of the header will hold the maximum value of the N dimensions in integer form. By labelling these N integer 'variables' in the following form:

```
AR00  .BYTE 0,0
AR01  .BYTE 0,0
```

for an array A(M,N), where the name for the J'th dimension maximum is generated by taking the array name (restricted to two characters), following it by I R or S (for Integer, Real or String arrays) and then adding the two digit hexadecimal representation of the dimension number, the dimensions of the array can be set up dynamically by generating code to evaluate the expression for a dimension, convert it to integer, then store it at address AR00, AR01 etc.

The key routine for referencing arrays, to be called INDEX, will be followed by the address of the array header and will expect on the 'evaluation stack' (used for evaluating expressions) the N values of the subscripts for this particular array reference. It will then place on the 'address stack' (possibly the same as the evaluation stack) the computed address, removing the N subscript values as it does so. This can then be used either to assign the result of an expression to an array element or to retrieve the value of an array element for use in an expression evaluation.

Summary

The design of a proposed compiler has been sketched in sufficient detail to demonstrate that it is practical to attempt the project. The compiler will require a 32K PET and disk (and assembler) and will be presented in stages, a possible outline being:

1) Run time library (routines needed by the compiled program, program initialisation, error routine etc).
2) String handling (concatenation, LEFT$, LEN etc)
3) Array handling (DIM statements, INDEX routine etc).
4) Variable allocation (both simple and arrays)
5) Program flow statements (FOR...NEXT, IF...THEN, GOTO, GOSUB, ON...GOTO etc).
6) Expression evaluation (possibly the key routines of the compiler, DATA evaluation etc).
7) The compiler's main routines.

Continued on page 7.