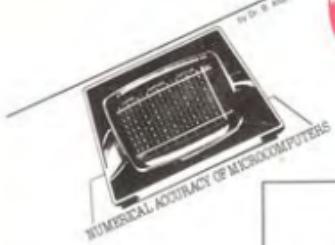


January 1980

Second Edition 50p

LIVERPOOL SOFTWARE GAZETTE



Take your first bite at computers with Apple



® Starter System only £750*
Typical Business System £2500*

WHY APPLE?

APPLE II Plus will change the way you think about computers. It's been designed to handle the day-to-day activities of business, financial planning, scientific calculation, education, and even entertainment. It makes learning to use computers enjoyable and creative, by bringing to the user a new level of simplicity.

Apple Computer has produced a total system based upon the incomparable APPLE II Plus Computer, which has an unequalled range of accessories with superbly produced documentation.

APPLE FEATURES

The basic APPLE II Plus can be used on its own or as the basis of a more comprehensive business computing system by adding such items as floppy disc drives and printers. Professionally written programs are available for a wide variety of tasks.

APPLE II Plus is easily programmed in BASIC but now has available for the first time - PASCAL.

APPLE II Plus also has some futuristic accessories available today such as - programmed speech output - speech recognition - a superb music synthesizer that even displays the musical stave as

it plays - a graphics input tablet—all this and high and low resolution colour graphics too!

Apple brings professional standards to personal computing. It gives you the features, appearance and "feel" for ease of use. The Apple name is your guarantee of satisfaction.

APPLE RELIABILITY

Apple II Plus comes with a solid warranty and the Microsense Computers nationwide team of Dealers who can advise and help you in choosing the System and accessories to suit your particular needs whether for Business, Education or at the Home.

APPLE IN BUSINESS

Apple is ideal for the small company run by its owner-managers. The Business Computer System can, for example, help you run the company, Farnell or handle the Stock Control for a Retail Store. Specialist applications include those for managing an Estate Agents records.

APPLE IN EDUCATION

Computer literacy is rapidly becoming an essential part of the world in which we live. Real "hands-on" experience with Apple computers is believed to be more effective in preparing them studies for business, commerce and the professions where computers will soon be as common as typewriters.

APPLE IN THE HOME

The computer can help you give your children a head start in understanding this modern business environment. It can help you manage your home learning and increase computer awareness. For the household there are the advantages of easily handling home finance.

ARE YOU TEMPTED?

If you wish to purchase the Apple Computer system, contact your nearest Microsense Dealer, or write to Microsense Computers Limited, who will put you in touch with them.

*Please telephone 01744 870000 and request an information pack.

SOLE U.K. DISTRIBUTOR

microsense computers

Finney Road, Hemel Hempstead, Herts HP2 7PS
Hemel Hempstead (0442) 41191 (3 lines)
and 48151 (3 lines); 24 hour answering service
Telex: 625554 DATEFF G



© Apple is a trade mark of Apple Computer Inc., Cupertino, CA, USA.

MICRODIGITAL BOOKS

Automatic Portable & Recordable

23.95 Basic - a cost for Secondary Schools

£18.95 Schools

£18.75 Basic Programming

£18.75 Basic Data Processing

£18.49 Basic Data Structures

Programs

£18.00 Advanced C Compiler

£15.95 Advanced Data Structures

£15.95 Artificial Intelligence

£15.95 Art & Computer

£15.95 Artistic Computer Programming

£15.95 Artistic Games

£15.95 Artistic Games & Recordable

</div

LIVERPOOL SOFTWARE GAZETTE



Publishers Letter

Dear Reader,

MY first two months as a publisher has produced a strange mixture of experiences. Help and support has come from the most unexpected of sources, for which I am extremely grateful, yet some people and organisations who are benefitted by the magazine have shown a strange apathy.

United Kingdom distribution is being handled by Computer Bookshop of 43-45 Temple Street, Birmingham, to whom all enquiries for bulk purchase should be directed. Through their efforts Liverpool Software Gazette should be available at every computer store in this country.

United States distribution is being handled by Bits Inc. of Peterborough, New Hampshire, and I am currently in the process of arranging European distribution.

This issue of the magazine has more pages than the first and more careful proof reading plus less undue haste should have resulted in less errors. For this improvement to continue we need more advertising as this is the life-blood of any magazine, anyone interested should look at the advertising details elsewhere within these covers.

To finish may I once again exhort you to communicate with us your likes, dislikes,

grievances, contributions etc., as these too will improve the magazine.

B. Everiss

BRUCE EVERISS

CONTENTS



Dungeons & Dragons Revisited	4
Numerical Accuracy of Microcomputers	8
Ceil—an introduction	12
Acorn and the Kim	15
Z-80 Processor Profile	18
Revax & Zeap	20
Application Software for Microcomputers	25
Pets Corner	27
Nobles	28
Chess for the Acorn	37
Exetera	43
Surround	44
Apple Pips	48
A Hen on You	54
Programming Practices and Techniques	57
Trekking by 'JTK'	60
Nascom Notes	61
Errata	61
Byting more off your Disk	62
Stop Press	64



LIVERPOOL SOFTWARE GAZETTE

EDITORIAL

PUBLICATION of the first edition of Liverpool Software Gazette produced as wide a range of comments, suggestions, criticisms and appreciation as we had feared! As a result there are a number of changes in the presentation and format of this issue. As far as possible all listings, however small, are reproduced directly from the respective machines listing—based on the well known premise that if the machine typed it, it must have accepted it in the first place!

Additionally, many people wanted to adapt a particular program to run on their machine—in future we will try and publish conversion hints along with the article. I believe this month's edition shows many improvements over its predecessor. Contributions, letters, comments are always welcome.

From the third edition onwards Nikki Devereux will be handling our advertising. If you are even just vaguely interested in taking space please contact her on 051-227 2535.

ADVERTISING:

Full page (17.5 cm x 24 cm)	£180.00
Half page (Upright 8.5 cm x 24 cm)	£ 95.00
Half page (Landscape 17.5 cm x 11.75 cm)	£ 95.00
Quarter page (8.5 cm x 11.75 cm)	£ 52.00
Agency discount 10%	

Please contact Nikki Devereux on 051-227 2535 if you would like further details.

REPRINTS: Articles that are explicitly marked as having restricted reproduction rights may not be copied or reprinted without written permission from Microdigital. All other articles may be reprinted for any non-commercial purpose provided a credit line is included stating that said material was reprinted from the Liverpool Software Gazette, 14 Castle Street, Liverpool, L2 0TA. Please send copies of any reprints to Liverpool Software Gazette, attention of Carl Phillips.

DISCLAIMER: All the information in the magazine has been thoroughly debugged and tested. However, no guarantees are made as to its truth or validity.

TRADE DISTRIBUTION: Computer Bookshop, 43-45 Temple Street, Birmingham, 021-643 4577.

U.S. DISTRIBUTION: Bits Inc., P.O. Box 428, 25 Route 101 West, Peterborough, NH 03458.

(C) MICRODIGITAL 1980

Publisher: Bruce Everiss.

Editor: Carl Phillips.

Editorial Assistant: Marie Beard.

Contributing Editors: John Stout, Dr. Martin Beer, Dave Straker.

Advertising: Nikki Devereux.

Subscriptions: Christine Crofton.

Artwork: Peter Croft.

THE LIVERPOOL SOFTWARE GAZETTE is published bi-monthly by Microdigital, 14 Castle Street, Liverpool, L2 0TA.

Subscriptions: Within Great Britain, £6.00 for 12 issues. Individual copies, by post 60p. Please tell us the issue you would like to start a subscription with!

CAMBRIDGE LEARNING ENTERPRISES

Self Instruction Courses

Microcomputers are coming - ride the wave! Learn to program. Millions of jobs are threatened but millions more will be created. Learn BASIC—the language of the small computer and the most easy-to-learn computer language in widespread use. Teach yourself with a course which takes you from complete ignorance step-by-step to real proficiency with a unique style of graded hints. In 60 straightforward lessons you will learn the five essentials of programming: problem definition, flowcharting, coding the program, debugging, clear documentation.



Book 1 Computers and what they do well; READ, DATA, PRINT, powers, brackets, variable names; LET; errors; coding simple programs.

Book 2 High and low level languages; flowcharting; functions; REM and documentation; INPUT, IF...THEN, GO TO; limitations of computers; problem definition.

Book 3 Compilers and interpreters; loops, FOR...NEXT, RESTORE; debugging; arrays; bubble sorting; TAB.

Book 4 Advanced BASIC; subroutines; string variables; files; complex programming; examples; glossary.

Understand Digital Electronics

Written for the student or enthusiast, this course is packed with information, diagrams and questions designed to lead you step-by-step through number systems and Boolean algebra to memories, counters and simple arithmetic circuits and finally to an understanding of the design and operation of calculators and computers.

Book 1 Octal, hexadecimal and binary number systems; conversion between number systems; representation of negative numbers; complementary systems.

Book 2 OR and AND functions; logic gates; NOT, exclusive OR, NAND, NOR and exclusive NOR functions; multiple input gates; truth tables; De Morgan's Laws; combinational logic; logic function mapping; three state and weird logic.

Book 3 Half adder and full adder; subtraction; serial and parallel adders; processors and ALU's; multiplication and division systems.

Book 4 Flip flops; shift registers; asynchronous and synchronous counters; ring, Johnson and exclusive OR feedback counters; ROMS and RAMS.

Book 5 Structure of calculators; keyboard encoding; decoding display data; register systems; control; program ROM; address decoding.

Book 6 CPU; memory organisation; character representation; program storage; address modes; input output systems; program interrupts; interrupt priorities; programming, assemblers; computers; executive programs; operating systems.

GUARANTEE - No risk to you

If you are not completely satisfied your money will be refunded on return of the books in good condition.

Please send me—

....Computer Programming in BASIC (4 books) @ £7.50

....Design of Digital Systems (6 books) @ £11.50

All prices include worldwide surface mailing costs (airmail extra)

IF YOUR ORDER EXCEEDS £15, DEDUCT £2

I enclose a cheque/PO payable to Cambridge Learning Enterprises for £.....

or please charge my Access/Barclaycard

account no.

Telephone orders from credit card holders accepted on 0480-67446 (Ansafone). Overseas customers (inc. Eire) send a bank draft in sterling drawn on a London bank, or quote credit card and number.

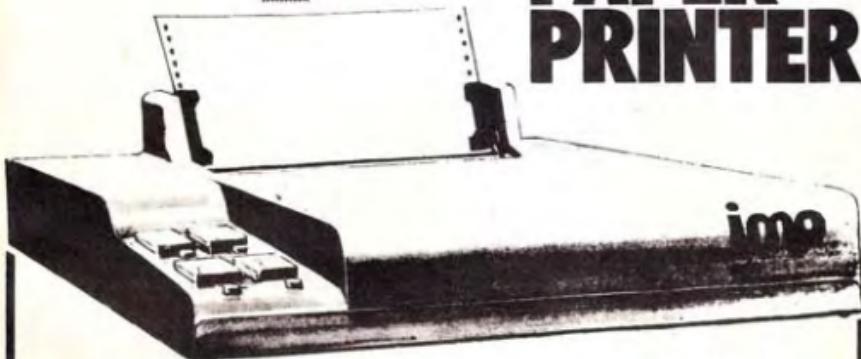
Name
.....

Address
.....

Cambridge Learning Enterprises, Unit 46, Rivermill Site, FREEPOST, St. Ives, Huntingdon, Cambs PE17 4BR England.

nascom imp

PLAIN PAPER PRINTER



BOXED AND BUILT FOR ONLY £325

PLUS VAT

FEATURES

- Serial RS232 interface
- 80 characters wide
- Bidirectional printing
- 60 lines per minute
- 10 line print buffer
- 96 character ASCII set (includes upper/lower case, \$ # £)
- Automatic CR/LF
- 8½" paper
- Optional tractor feed
- Baud rate from 110 to 9600
- External signal for optional synchronisation of baud rate

The Nascom IMP plugs straight into a Nascom 1/2 but is usable with all other micro systems. Parallel option will be available shortly.

TO NASCOM MICROCOMPUTERS LTD

92 BROAD STREET
CHESHAM
BUCKS

Tel: 02405 75155

Nascom Microcomputers
nm
NM LSG

Please send me Nascom IMPs at £325
each plus VAT plus £2.50 p&p

NAME _____

ADDRESS _____

ACCESS/
BARCLAYCARD NO.

Dungeons & Dragons Revisited

by D. Straker

IN the beginning was man, and man made war. Soon man found that war was expensive both in money and men, so in between wars, whilst saving up for the next one, he invented the wargame, in order to satisfy his lust for conflict.

As man evolved, so did his games. Some became fixed with simplified rules and playing area, such that the notion of human conflict became an abstract conflict, such as chess or draughts. Others, ever trying to simulate the real thing, became large and complex, with dimly defined borders, such as the modern wargame.

The modern trend towards pacifism, coupled with the enormously popular novels (?) of J.R.R. Tolkein (notably *The Lord of the Rings*) spawned the current drift away from warring and towards more fantastic forms of gaming. The forerunner of most of these is a game called Dungeons and Dragons (D & D to initiates) which emerged in 1973 from Gary Gygax and Dave Arneson in America.

The players now, instead of taking the parts of soldiers, play as hobbits, magicians, etc. exploring unknown caverns in search of treasure and adventure.

Conflict still arises, such as when you meet a hostile Troll, but it is less significant, and relates less to human activities. These games require one experienced player to take the part of referee and games-master. He plans the caves and decides upon the action, often depending upon the roll of a die.

Inevitably, perhaps the role of games-master was taken over by the computer. The first of these was Wil Crowther's 'Adventure', which now occupies many hours of processor time on PDP 11's across the globe. A successor is Zork, written in 1977 at MIT, which can completely fill a DEC 10 (much larger than PDP 11). These games are all based upon the question and answer, e.g.

Program: Suddenly a big goblin jumps out of the door on the left. He has a big sword and a mean look.



- Player: Hit goblin.
- Program: With what?
- Player: My fist.
- Program: The goblin jumps aside, and takes a swipe at you. His sword narrowly misses, and rings off the stone to your right.
- Player: Light flare.
- Program: The goblin is dazzled, he flees leaving behind a small brown package.
- Player: Open package.
- Program: Inside is a silver snuff box with a curious inscription on the top.
- Player: Read inscription.
- Program: SHALACZ MARAB!! You are transported to a massive room.
- Etc.

The complexity of such games depends upon the ingenuity of the programmer and the amount of recallable space in the computer. Much information is held in tables. These tables may be held in memory, but if a disk system is available files of this information may be used to expand the game almost infinitely.

This article is not written to completely describe a program, but as a stimulus, a provider of basic ideas and a simple systems analysis of the problem.

The program(s?) must describe the player, the other creatures and monsters, the environment and cater for as many interactions between them as possible.

A goal to aim at is essential, whether it be to find the lost crown of Albaroc, rescue Alesia the Beautiful from the goblin hordes, or simply to acquire as much experience and treasure as possible.

Chronology Log

If possible a log should be kept of all activities, so that past experiences can be referenced. The following other tables may be needed:—

Player Table

1. Basic Characteristics—a fixed number of levels for each, e.g. 16 (4 bits?). Set at initialisation, either randomly or by player. They may be affected by play, e.g. strength is decreased by a wound, and charisma is increased by charming monsters, e.g.

strength	— physical strength
wisdom	— ability to understand creatures
dexterity	— physical co-ordination
intelligence	— ability to understand problems
charisma	— personality appeal
constitution	— health and physical stamina

2. Possessions—

Limits that player may carry, e.g. 100 kg (may depend upon strength) plus all items carried, including weapons, armour, heavy treasure, magic items, food, etc., taken from the carried items tables.

3. Others—

including name, race and any other special attributes a player may have, such as mobility, spell-maker, etc.

Creature and Monster Table

Each creature or monster has an entry, describing: species, strength range, friendliness, special weaknesses (e.g. fire), value if defeated (e.g. strength + 1, wisdom + 1), etc.

Carried Items Tables

These may be divided into:

1. Armoury Items

Weapons for physical attack and defence, such as swords, shields etc. Each has attack and defence value, cost etc.

2. Survival Items

Such as lights, rations etc., each having a cost and duration of use.

3. Magical Items

Each with name, requirement for use, (e.g. wisdom of 10) value, and special attributes (e.g. wards off spectres) etc.

4. Treasure Items

Such as gold and jewels, with corresponding value—all carried items have weight (it may be zero)

The above tables may be shortened, lengthened, subdivided to suit the game, but once established, may be used for many separate scenarios.

Environment Table

Each room, tunnel and cavern must be described. The contents of each, the position of items in the room described, so that when the room is entered, a description may be given to the player. A room may contain treasure, artifacts or monsters. Special events may occur upon certain actions, e.g. if a chest is opened, 3 goblins appear 3 turns later. Events may occur randomly at specific probabilities and triggered by other events. One

environment table per scenario is required.

Current Environment Chart

A chart of the present environment needs to be kept. The states of all doors, traps and objects are kept, so that the player cannot strike an imaginary orc with a non-existent sword, when the room is exited that chart can be saved in the chronology log, such that if one room is re-entered, it will be as left.

Descriptions Table

This describes every object in the game. It may be very long (even filling a small disk!) If the player asks to DESCRIBE something, this table is searched, and the description displayed. (Alternatively a printed handbook may be kept)

In order to receive and understand input from the player, the program must apply some simple lexical analysis. This needs tables of recognised words:

Item Table

All items that may be referenced, such as weapons, treasure, creatures etc. (the original tables may be used).

Action Table

All possible actions, plus expected accompanying words or word types (e.g. where do you run? What, and with what do you hit?).

Direction Table

For movements up, down, north, onwards etc.. To expand the vocabulary, several words may be grouped for one action (e.g. strike, hit etc.)

Responses Tables

Contain phrases and sentences to build up response to action command. The specific effect is coded for program use.

Great care must be taken as many words have several meanings.

The above suggestions for table break down are by no means a rigid or complete, but they indicate how tables may be used to create a total base for a fantasy game.

The program itself needs a beginning, a middle and an end. To start, the character and environment must be initialised, either by the program or the player. If the program is starting, a good description at the beginning of the player's surroundings and objectives is essential to give the player an initial injection of enthusiasm. e.g.

"You have just arrived high in the Andean Alps, in the ancient city of Ankarapatal. Here there is an old legend of fabulous wealth hidden deep in the catacombs. You have 3,000 Dengas with which to buy supplies from the local traders before you enter where no man has ever returned from. Even as you arrived, the sound of anguished howls, far away, meet your ears. You must tread warily. For unknown things guard the greatest

treasure ever assembled through the long eons of time. Make your moves with care and short, active phrases—good luck, for you will need all your life can find..."

A player should be able to terminate a game at any time (e.g. by saying a magic word) and possibly re-enter the game at the same point at a later date—i.e. by restarting from the Chronology Log.

The adventure continues in a command for action and response until either the player terminates the session, achieves his goal(s) by a set degree, or is killed, captured etc.

The program may be divided into 3 parts:

- (a) Analysis of command
- (b) Formulation of response.
- (c) File handling and searching—This is called by both (a) and (b)

Each part will now be considered in outline.

(a) Analysis of the input command includes:

- (i) Finding the action by searching the action table with command words
- (ii) If the action requires other words for clarification, searching the command for these, using the item/direction tables. Command types may be:
 1. action.
 2. action direction.
 3. action item.
 4. action item (preposition) item.

Examples of these are:

1. shout.
2. run north.
3. drop sword.
4. throw crystal ball at door.

Commands may be stacked using 'and':

drop sword and run north

If a command is incomplete, the remainder must be asked for e.g.:—

Player: run

Program: run where?

Special commands must be catered for, such as status request and termination requests.

If the command cannot be recognised, then the default response:

Nothing happens.

is given, and the return to command request made.

(b) Once the command has been recognised, the appropriate response is made using the Response Tables, coupled with the current Environment Table.

The feasibility of the command must first be checked. If the player, for example, tries to use an item that he does not possess, then he may be prompted or penalised,

similar options remain if the player makes an obviously foolhardy action (i.e. one that is not catered for). Thus the command:

Hit troll with feather
may either receive the reply:

Troll laughs

or

Troll becomes annoyed, and cleaves you in two.
The response, may be of several types:

- (i) Relocation of player in a new environment, when a suitable command is given, spell is read etc.
- (ii) Resolution of encounter with another creature. In an encounter, a player may:
 1. Approach in a friendly manner.
 2. Approach in a hostile manner (attack).
 3. Approach in an indifferent manner (ignore).
 4. retreat.

Each of these will trigger specified reactions as set in the Environment/Creature tables, modified by the player's basic characteristics, so a player with a high charisma will more probably persuade the creature to help.

If a combat situation ensues, the accuracy of blows, dodges etc., again depends upon the Creature and Environment Tables, and the player's characteristics. A combat turn is resolved first, by deciding whether a blow is made, and then how much damage is done.

(iii) Resolution of encounter with object, such as a booby-trapped treasure chest.

(iv) The allocation of rewards/penalties in the form of treasure, modifications to basic characteristics, etc., and such as the increase in wisdom when a trap is avoided or decrease in strength when wounded.

Factors which can affect a response must be considered, they include:—

1. time—an event may be triggered only within a given timespan.
2. certainty—an event than cannot be avoided.
3. interference—if certain precautions are not taken then an event will occur.
4. Warning—impending events may give warning signs, examples are:—
 - (i) A lamp burning out.
 - (ii) A roof collapse.
 - (iii) If sign is not read then the player is mugged (and treasure lost).
 - (iv) The sound of distant laughter, growing louder.

(c) Tables may be held in arrays or files. Arrays are memory based, and therefore limited. Files, held on disk provide much wider possibilities, and are more easily changed. Each table is made of records, and each record consists of a search element, plus fields relevant to that entry. e.g. a record in the Creature Table may be:

C12, Black Bat, 4, 20, 50, TSM

This is a Black Bat (referenced by C12), of strength 4 units. If it is approached, a random number from 1 to 100 is generated. If it is 20, or below the bat is friendly. If it is 21 to 50 then the bat is indifferent. If it is 51 to 100

then the bat attacks! A friendly bat will leave item T5 behind, as will a defeated bat, if mercy is shown.

What language should be used? For an extensive system using disk files, a language with a good record structure and file handling capabilities must rate highly. If you are lucky enough to have Pascal at your disposal, then it must be highly recommended. This does not exclude BASIC, in which many excellent games can, and have, been written. For packing most into a small space, and gaining faster running speed, if you have the time and inclination, an assembler program is always (potentially) the most efficient.

This article has been an attempt at a broad outline of a workable fantasy game. It still requires much more work on it, but I believe it shows the possibilities that exist for worlds to escape into when the Klingons have all been driven away. Science fantasy is a long way from dead—a space game equivalent to this, with exploring strange new worlds and boldly going where no man has boldly gone since last week is potentially infinitely more complex than this—our game could be a small subsection of the ultimate universe game.

If you are interested in this type of game, get a copy of Dungeons and Dragons basic rules. There are many clubs that play it (often along with wargames) across the country, so try a game run by an organic computer!

Science fantasy fans might prefer the complexities of 'Traveller' a similar variety of a space theme.

Above all when you are programming games, remember, a good game leaves a player satisfied, but always ready for more escape into that realistic world that is a marriage of imagination and program generated fantasy.



STRATHAND



Apple II comes to Scotland

Why not call and see the fantastic Apple II the finest micro currently available. Demonstration without obligation.

16K	£750
Disc drive with controller	£398
16K add on (Max 2 giving 48K total)	£89
High-speed serial I/F	£110
Parallel I/F	£110
Comms card	£132
Applesoft firmware card	£110
Centronics card with cable	£132
Hitachi 9 in. monitor	£127.00
Hitachi 12 in. monitor	£167.00
APPLE CLOCK BOARD — Real time clock with battery back up. 388 days by 1ms intervals	£140

Pascal Language System now available for Apple II

Editor, Compiler, Relocatable Assembler, System Utilities, etc. Price including Language Discs, 16K Memory Card, Documentation £298.

Note: Integer Basic, Floating Point Basic, and Pascal all on Discs supplied with package.

STOP PRESS

SUPERCOLOUR FOR APPLE II. At last — top quality colour for your Apple. Brand new Supercolour board. Gives red, green, blue and sync, as totally independent TTL signals, thus eliminating all previous colour problems. The quality of colour using this method which drives the 3 colour guns of the CRT independently is fantastic. Colour of text, low res. graphics and high res. graphics can be switched separately by the user, e.g. green text. Complete with 14" Sony monitor and boards for Apple II, £440.

WALTERS DOLPHIN HIGH-SPEED PRINTER£595
Intertube 2 VDU now in stock.

Software packages prepared by arrangement. For further details please write, phone or telex.

All prices exclusive of VAT.

STRATHAND

44 ST. ANDREW'S SQUARE,
GLASGOW G1 5PL
041-552 6731

Tel. order welcome with Access and Barclaycard

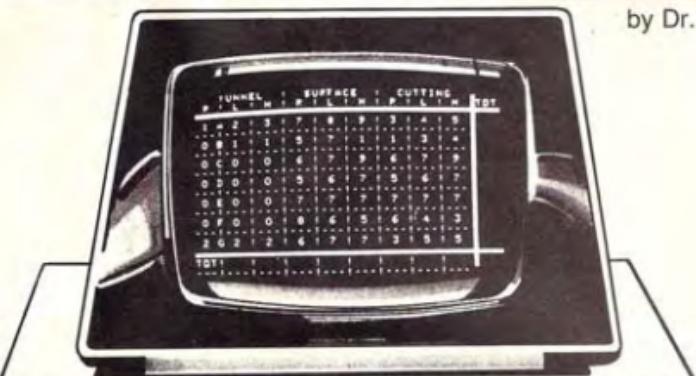


Callers welcome



Now on Telex 777268, 24 Hours Service

by Dr. B. Allan



NUMERICAL ACCURACY OF MICROCOMPUTERS

INTRODUCTION

IF one studies the application made of microcomputers, these applications are in areas where frequently an Integer Basic is all that is needed (e.g. games, simulations or stock control). Applications for which we need to be concerned about figures accurate to seven, eight or more significant digits are few. As one of the principal concerns of main frame computer ('macrocomputer') users is with what might be termed 'statistical computing' software (e.g. SPSS, GLIM, GENSTAT and other acronyms), I have made an analysis of whether it is possible to perform some of these mystical rites on a microcomputer. My evaluation had to be concerned with two main questions:

- Was the storage necessary for multivariate statistical methods available with microcomputers?
- Was the normal microcomputer accurate enough to be able to cope with a large amount of numerical calculations?

The answer to the first question was 'yes': storage was more than adequate in RAM and on disk for many microcomputers. The answer to (b) was a qualified 'yes': pretty accurate; not as accurate as hand-coded BASIC on a typical macrocomputer; more accurate than macrocomputer BASIC using built-in functions; and compares well to an earlier generation of macrocomputers.

WORD LENGTHS AND ERROR ANALYSIS

If one takes a typical calculator, and enters a series of '1's, one soon finds that only a set number of '1's can be entered. In the case of the calculator I am using, the number in the display is '11111111', and I can key in as many other numbers as I care, but only these eight significant digits are recognized by the calculator. When the square root key is pressed the digits '3333.3333' are displayed and if this number is then squared the result is '11111110.'. In an exact world the square of

the square root of a number is the same as the original number (e.g.: the square root of '4' is '2', and the square of '2' is '4'); in most computation the exact world does not fit. We have a 'rounding error'; that is, the square root of '11111111' is never ending series of digits of which the first eight are '333.333', and there is never any rounding-up. Technically, my calculator works with 'chopped' arithmetic (the trailing digits are chopped off).

Present macrocomputers do not usually work with chopped arithmetic, but work with what is termed 'rounded' arithmetic—the final significant digit depends upon the value of the following digit. Both forms of arithmetic produce errors, but with rounded arithmetic there is no consistent bias up or down, as there is with chopped arithmetic. For the ICI 1900 series (I used a 1902T in the comparisons below) each real number is held in two 24-bit words (48 bits in all). Of these 48 bits: 37 bits constitute the fixed point part (or mantissa) giving a relative accuracy of about eleven decimal digits (2^{-37} is about 7×10^{-12}); 9 bits constitute the exponent part, so that numbers can vary between about 10^{78} to 10^{-76} in absolute size; and 2 bits are used for the signs of the two parts. (See Goult et al (1974: Ch 2) for more information; the key book is Forsythe & Moler (1967); and for definitions see Chandor (1977) but note that neither 'RAM', 'ROM' nor 'Microcomputer' appears in this latter publication.)

The 6502 processor (used, inter alia, by APPLE II, PET and ACORN) has a byte-based system of 8 bits per byte. Most of the development work for the comparisons below was performed in APPLESOFT BASIC (though they have been replicated on a PET), in which real numbers are held in five bytes (40 bits in all). Of these 40 bits: 30 bits hold the fixed-point part, giving a relative accuracy of about nine decimal digits (2^{-30} is about 9×10^{-10}); the exponent part is 8 bits long, so that numbers can vary from about 10^{58} to 10^{-58} ; and the 2 remaining bits hold the signs of the two parts. This would indicate

that APPLESOFT BASIC has a high level of accuracy, but the relative accuracy of APPLESOFT BASIC compared to the SOBS BASIC on the 1902T is of the order of 1:128. Is this important?

TESTS OF ACCURACY—INVERSE HILBERT MATRICES

The impetus behind this examination is a multivariate statistical system being written for microcomputers (the Factor Analysis program is to be released in the early part of 1980): in multivariate analysis the most common mathematical procedure one uses, once one has a matrix of measures of association, is the inversion of a positive semi-definite matrix (those looking for a simple introduction to matrix algebra should try Branfield and Bell (1970), then chapter V from Noble (1966), and the more advanced reader can do no better than Forsythe and Moler (1967)).

For this reason I have developed a highly numerically accurate matrix elimination and decomposition set of routines, using Cholesky's method (to be discussed in more detail in the second paper in this series). The Cholesky routine was applied to test matrices with the coding in APPLESOFT BASIC (Apple II) and in SOBS BASIC (1902T)—in addition the built-in SOBS BASIC matrix routine 'MAT INV' routine seemed less stable.

Most of the matrices tested were well-behaved without an exactly known inverse, and for this reason I decided to use inverse Hilbert matrices as test matrices (Forsythe & Moler, 1967: Ch 19). Hilbert matrices are

not well-behaved (the technical term is 'ill-conditioned') in that small rounding errors have their effects maximized, but for reasons explained in Forsythe & Moler (1967: Ch 19) both the Hilbert matrix and its inverse are known exactly. Hilbert matrices can be of any order from 2×2 upwards: the Cholesky routines in APPLESOFT BASIC and SOBS BASIC were tested for Hilbert matrices from 2×2 to 10×10 , together with the SOBS 'MAT INV' routine, (PETS BASIC gave identical results to APPLESOFT BASIC). (The Cholesky routine is a 'pseudo-inverse' routine, in which zero pivots are accommodated.)

To give a feel for what is meant by an 'ill-conditioned' system consider this pair of simultaneous equations:

$$100X + 99y = 199$$

$$99X + 98y = 197$$

where the exact solution is $X = 1$ and $y = 1$. Suppose that for some reason (e.g. it was an initial estimate) X was estimated to be 3, which led to the estimate $y = -1.020202$ on a calculator with 8 significant (chopped) digits. This produces

$$100X + 99y = 199.00001$$

$$99X + 98y = 197.02021$$

Differences of such magnitude (about 2 units) in X and Y only produce a relative error of .01025% in the second equation; in the first equation the relative error $.00001/199 \times 100\%$ is identically equal to zero on my calculator. This is an 'ill-conditioned' system and the 'condition number' is approximately 3.96×10^4 to reflect this.

LARGEST ERRORS IN HILBERT MATRICES

Order	Condition Number	SOBS Error	APPLESOFT Error
2×2	1.20×10^1	0 (1)	0
3×3	1.92×10^2	0	0
4×4	6.48×10^3	0	0
6×6	1.79×10^5	0	2.5×10^{-5}
6×6	4.41×10^6	1.0×10^{-4}	5.8×10^{-4}
7×7	1.33×10^8	9.0×10^{-6}	1.6×10^{-2}
8×8	4.25×10^{10}	5.1×10^{-8}	"(2)
9×9	1.22×10^{11}	4.7×10^{-10}	*
10×10	3.48×10^{12}	2.0×10^{-12}	*

Notes: (1) The SOBS BASIC routine 'MAT INV' would not even invert the 2×2 inverse Hilbert matrix;
 (2) At this order and greater the routine detected zero pivots and thus did not give an inverse, but rather a pseudo inverse.

Table 1 shows the largest error in calculating the Hilbert matrix of different orders for SOBS BASIC and APPLESOFT BASIC versions of the Cholesky routine. The condition number is shown for each matrix, and it can be seen that beyond a 4×4 Hilbert matrix we have systems which are even more unstable than our example.

From an examination of this table one can see: the APPLESOFT BASIC Cholesky routine is remarkably accurate upto and including the 6×6 matrix (this matrix has a very large condition number); the SOBS BASIC

Cholesky routine is highly accurate up to and including the 8×8 matrix; and the poorness of the supplied SOBS BASIC inversion routine ('MAT INV') is indicated by the fact that it could not invert the 2×2 matrix.

Incidentally, a 6×6 Hilbert Matrix is given in full in Forsythe & Moler (1967:82) and their largest error is 1.9×10^{-4} calculated on an earlier macrocomputer, the IBM 7090 this is of the same order of magnitude as the APPLESOFT BASIC error.

CONCLUSIONS

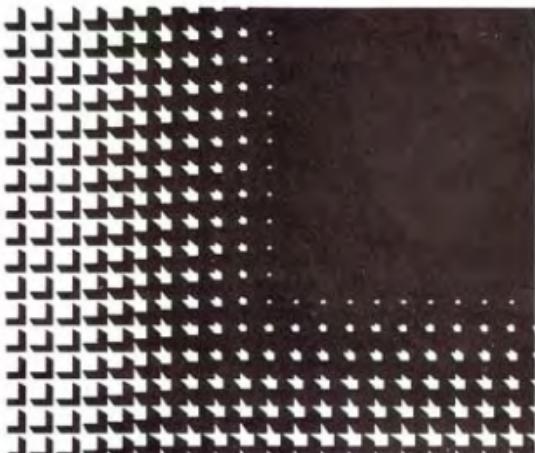
It would seem that, if we are careful with the computational efficiency of our numerical routines, we are able to guarantee a fair degree of numerical accuracy in our results on at least one microcomputer, APPLE II. What is equally interesting (yet not as important) is that present microcomputers are about as accurate as many of the

60s and early 70s macrocomputers. (Though the implementation of 'SQR' in APPLESOFT BASIC may not be as efficient as it might be).

The next article in this series will discuss matrix inversion routines in BASIC for microcomputers.

REFERENCES

- Branfields J.R. and Bell A.W.
1970 **Matrices and their Applications**
London: Macmillan.
- Chandor A. (with Graham J. and Williamson R.)
1977 **The Penguin Dictionary of Computers.**
(Second Edition)
Harmondsworth: Penguin Books.
- Forsythe G. and Moler C.B.
1967 **Computer Solution of Linear Algebraic Systems.**
Englewood Cliffs: Prentice-Hall.
- Goutt R.G., Hoskins R.F., Milner J.A. and Pratt M.J.
1974 **Computational Methods in Linear Algebra.**
London: Stanley Thornes.
- Noble B.
1964 **Numerical Methods: 1 Iteration, Programming and Algebraic Equations.**
Edinburgh: Oliver and Boyd.



Reach the people who matter with the **LIVERPOOL SOFTWARE GAZETTE**

The microcomputer magazine that is



An informal poll of our readership (3 samples) showed that they were well educated, normal, responsible people who buy things.

It therefore obviously pays to advertise ...

Ridiculously reasonable rates high quality editorial and production standards make us unbeatable media in which to advertise your product, however mythical it may be. Advertise that Z-8000 board here, the £325 1200 CPS impact Printer, or Hard Disk sub-system for SC/MP.

Notes

Full page	(17.5 cm x 24 cm)	£180.00
Half Page	(Upright 8.5 cm x 24 cm)	£95.00
	(Landscape 17.5 cm x 11.75 cm)....	£95.00
Quarter Page	(8.5 cm x 11.75 cm)	£52.00
Agency discount	10%	

Copy Date for the March Issue is February 15.



MICRODIGITAL

25 Brunswick Street, Liverpool L2 0PJ. Tel: 051-227 2535/6/7/8



CESIL an introduction

D. Straker

CESIL is a very simple language devised by ICL as an introduction to low level programming for school children, hence the name: Computer Education in Schools Instructional Language. As it is uncomplicated by hexadecimal, index registers two's complement etc. it can also serve admirably as an introduction for anyone curious about the vagaries of assembler Languages.

It also provides an excellent example for a programmer who wants to write a Compiler/Interpreter. This article is thus written for as wide a readership as possible, so complex terminology has not been used, and many concepts have been considerably simplified. Stack pointers, program counters and addressing modes are all ignored—the only register (or 'special' memory location) considered is the *accumulator*, with respect to which, all operations are carried out. All operations are on signed integer data—text handling is not included in CESIL. Each memory location holds one signed integer.

The CESIL instruction line is separated into 3 distinct parts, each written in a separate column:

LABEL	INSTRUCTION	OPERAND
-------	-------------	---------

- (a) LABEL—the label is optional, and is, as it says, a label, used by other instructions to refer to this line (like the line number in BASIC)
- (b) INSTRUCTION—the instruction indicates what this line is to perform, e.g. ADD, STORE etc.
- (c) OPERAND—the operand indicates which memory locations or integers the instruction is to act upon

The label and memory location names are restricted to six characters or less. The first must be alphabetic, the others alphanumeric, so FRED and A1 are legal and 1D and JIM* are illegal.

Input and output is always to and from the accumulator. Input is from a data list, given at the end of the program listing, into the accumulator, so

IN

will read the next number in the data list into the accumulator (like READ and DATA in BASIC). Conversely, the instruction

OUT

will print the number in the accumulator on to the terminal (like PRINT in BASIC)

Note: these two instructions have no operands

Moving numbers between the accumulator and memory is done with STORE and LOAD instructions:

STORE AREA

will copy the number in the accumulator into the memory location and named AREA, and

LOAD GREY

will copy the number in memory location GREY into the accumulator. Signed numbers can also be loaded, so

LOAD +2

will move the number +2 into the accumulator. Note that the sign (+ or -) on the number is required. Loading numbers like this is called loading *immediate* data.

Arithmetic is all integer, as no decimal numbers are allowed in CESIL. The standard four functions are available. Operands may be immediate data or named memory locations. The answer to all arithmetic operations is put in the accumulator, so

ADD LENGTH

adds the number in location LENGTH to the

accumulator.

SUBTRACT + 2

subtract 2 (ie adds 2) from the accumulator

MULTIPLY + 12

multiples the number in the accumulator by 12.

DIVIDE AREA

divides the number in the accumulator by the number in memory location AREA. The remainder is ignored. Jumps in the programs are always to a labelled line elsewhere in the program (like GOTO in BASIC), so

JUMP LINES

causes a jump to the line with label LINES. There are two indicators or 'flags' in CESIL, which at all times reflect the 'condition' of the number currently in the accumulator. These are the negative and zero flags, and a conditional jump is available upon each, so

JIZERO RERUN

causes a jump to line labelled RERUN *only* if the zero flag is set, i.e. the number in the accumulator is zero, and

JINEG B40

causes a jump to line labelled B40 *only* if the negative flag is set, i.e. the number in the accumulator is zero. The last instruction may be used anywhere, but is commonly the last in the program listing. It stops the program running, as END in BASIC:

HALT

These 12 instructions are all there is to CESIL. They are summarised in Table 1. Note that all instructions can (and commonly are) abbreviated to the first three letters. Programs can now be written in CESIL, and an example is given in Fig 1.

When assembler listings are seen, they always have these three columns (although occasionally they are crushed together, which is very bad practice). Sometimes there are also numbers on the left and comments on the right of the listings. These can be explained using an extension to CESIL as a model: comments to the right are just that—comments, put in a fourth column, to help anyone else reading the program understand its functions (like REM in BASIC).

As the program itself is held in memory, to simplify things for the processor, instructions are held as coded numbers called op-codes (see Table 1). Locations used for storing numbers are reserved at the beginning of the program using the NUM(BER) command in the instruction column (note this is not a true instruction to the processor, but a command to the assembler). Any reference to a location is then made to the displacement, or offset of the location from the start of the program, so:

01 00 NUM FIRST

02 00 NUM SECOND

07 03 01	LOA	FIRST	CODE STARTS HERE
09 11 13	JIZ	NEXT	
11 05 05	ADD	+ 5	ONLY IF FIRST <> 0
13 02	NEXT	out =	

may be described thus:

the lefthand column of numbers is a count not of lines, but of memory locations since the start of the program. The other numbers are actually how the program is held in memory ready for processing.

The first two lines reserve two memory locations, calling them FIRST and SECOND, and initialises them to zero.

Later in the program, at the seventh location, the op-code of 03 indicates LOAD, and 01 indicate to load from the first location past the start of the program.

On the next line 11 indicates the JIZERO instruction, and the label is indicated by where in memory to jump to, i.e. the 13th location past the start of the program.

Immediate data is held as the signed integer as indicated in the next line.

This format is the author's extension which may be used to extend the idea of CESIL to help to explain assembler listings.

Another example is given in Fig 2.

CESIL has many critics (as does BASIC!), but its extreme simplicity has proved in many cases to be a good introduction to low level programming. It can open the door to the amazingly versatile world of Assembly programming to many people who would otherwise take one look and pass it by as 'an unintelligible load of letters and numbers'.

Table 1 CESIL SUMMARY

Instruction	Abbrev	Operation	Opcode
IN	IN	Acc ← next in data list	01
OUT	OUT	Terminal ← Acc	02
LOAD	LOA	Acc ← nm	03
STORE	STO	m ← Acc	04
ADD	ADD	Acc ← Acc + nm	05
SUBTRACT	SUB	Acc ← Acc - nm	06
MULTIPLY	MUL	Acc ← Acc × nm	07
DIVIDE	DIV	Acc ← Acc ÷ nm	08
JUMP	JUM	Jump to m	09
JINEG	JIN	Jump to 1 if Acc = 0	10
JIZERO	JIZ	Jump to 1 if Acc ≠ 0	11
HALT	HAL	stop execution	12

Key: Acc accumulator

becomes equal to

nm operand immediate data or memory location

m operand memory location

l label

Fig 1

CESIL example to find the average
of the 5 numbers:
12, 13, 7, 2, -9

	LOA	+ 0	
	STO	TOTAL	
	LOA	+ 5	
NEXT	STO	COUNT	
	IN		
	ADD	TOTAL	
	STO	TOTAL	
	LOA	COUNT	
	SUB	+ 1	
	JIZ	END	
	JUM	NEXT	
END	LOA	TOTAL	
	OUT		
	HAL		

Data List: 12 13 7 2 -9

Fig 2

Extended CESIL example to find the remainder after
127 is divided by 13

01 00	NUM	NUMBER	
02 00	NUM	DIVBY	GET DIVIDEND
03 01	IN		
04 04 01	STO	NUMBER	
06 01	IN		GET DIVISOR
07 04 02	STO	DIVBY	
09 03 01	LOA	NUMBER	START CALCULATIONS
11 08 02	DIV	DIVBY	NOTE INTEGER RESULT
13 07 02	MUL	DIVBY	
15 06 01	SUB	NUMBER	
17 07 -1	MUL	-1	
19 02	OUT		PRINT RESULT
20 12	HAL		

data list will
be put here



W. H. Powell

THE ACORN has much in common with KIM-1. It has the same 1½ kbyte RAM and uses the same CPU (6502 used by PET, APPLE, and Superboard). Both ACORN and KIM have calculator style displays and keyboards with an operating system in ROM that makes them good low cost tools on which to learn machine code programming for the 6502. Both provide tape interfaces for storing and reloading programs.

These similarities suggest running KIM software on ACORN. The author soon found that although the ACORN display routine is better than KIM, it is easier to run KIM software by writing routines for ACORN which make it behave like KIM's keyboard/display.

On KIM one has access to a routine (SCANDS) which scans the display once, and another (GETKEY) which scans the keys to see if one is depressed. To light the display steadily, one puts SCANDS in a program loop to keep scanning faster than the eye can see. Similarly, some pretty programming is necessary to make GETKEY debounce the keys. The programmes in the FIRST BOOK OF KIM show how to do this, and also how to manipulate the display for various games. A typical KIM key/display program contains:

```
START JSR SCANDS Light Display
      JSR GETKEY Read keyboard
      CMP PREV Same key as last time?
      BEQ START Else continue...
```

The ACORN monitor does all this for you, but in spite of this I have found it much easier to run KIM programmes by using routines that run like SCANDS etc so that the KIM software need be changed as little as possible. Fig 1 gives my programme for simulating KIM, and the SUMMARY TABLE lists the differences in mnemonics and locations for the two machines.

e.g.

20 1F 1F JSR SCANDS on KIM
becomes 20 AB 0E JSR SCANDS on ACORN

There are other points of note; the control keys on both KIM and ACORN return Hex values above 0F. Section 1 of the table shows the labeling and Hex values for these keys on each machine.

The I/O ports used for the calculator style display and keyboard are at different addresses and must be initialised to different values. These are summarised in sections 2 and 4 of the table. Section 3 lists the most common KIM pointers held on page zero.

KIM uses six digits which it numbers strangely 09 0B 0D 0F 11 and 13. The routine CONVD corrects this to the simpler ACORN numbering 0 1 2 3 4 5 (digits 6 and 7 are not required).

In order to make KIM software run on ACORN one first loads the programme in Fig 1 into the 128 bytes of RAM in the I/O port. Then one edits the KIM subroutine jumps and data direction ports by referring to the addresses in the TABLE.

One difficulty may remain ... the lack of a timer on ACORN. In most cases one can get around this problem. The first use for KIM's timer is to pause during execution. This can be done equally well without a timer by using a random number for games etc. The location 'TIMER' in Section 4 of the table acts as a source of random numbers for ACORN. The third use is to actually measure time while the CPU is running some other software. This really does need a timer, and the basic ACORN must be extended to run such programmes.

Fig 2 and 3 are Hex dumps of programs from FIRST BOOK OF KIM modified to run on ACORN. The changes to HI-LO (Fig 2) are straightforward, but the changes in ASTEROID (Fig 3) are more elaborate.

The different display routines are not the only differences between KIM and ACORN. The KIM monitor is much bigger (2 kbyte), and includes TTY routines as well as those for a calculator type display discussed above. Physically KIM is on a large single board whereas ACORN is in two neat Eurocards one of which, the CPU card, is clearly intended for building into other equipment. For this reason it carries a socket for

EPROM (1, 2 or even 4 kbytes with a suitable device) which could allow it to act as a stand alone controller. The CPU board is also easier to expand to fully memory since only the top and bottom 4 kbytes have been allocated to CPU card memory and I/O ports. Unlike KIM there are no assumptions about the remaining 56 Kbyte making it much easier to expand ACORN in a modular manner. On the other hand KIM has an established range of extensions already. One of the most interesting is MEMORY-PLUS which carries an EPROM programmer as well as sockets for ROM and RAM.

SUMMARY TABLE of KIM Display Routines for ACORN

1 Keys	KIM	Key	ACORN
	AD	10	M
	DA	11	G
	+	12	P
	GO	13	S
	PC	14	L
No Key	15	R	
	-	16	
	-	17	

2 Data Directions

KIM	ACORN
Data #7F	#07
Location PADD = 1741	ADDR = 0E22

For ACORN, this can be left in this state for reading keyboard. But KIM requires 00 → PADD to read keys.

3 Page Zero

POINL = FA INL = F8 Input Buffer
 POINTH = FB INH = F9 Input Buffer
 TEMP = FC

These are same for KIM/ACORN simulation of KIM.

4 Display/Keyboard Routines

KIM	Routine	ACORN	
1704	"TIMER"	0EFF	Generates Random numbers during KEY PR
1F19	SCAND	0EAS	Display address & contents.
1F1F	SCANDS	0EAB	Display 6 Hex digits.
1F40	KEY PR etc.	0ECD	Test Keyboard for any key.
1F6A	GETKEY	0ED8	Test which key pressed.
*{ 1F48	CONVD	0E80	Display 1 Hex digit.
1F4E	CONVD+6	0E86	Display 1 digit 7 segment pattern.
IF63	INCPT	0E9B	Increment display address
1FE7	TABLE	FFEA	Table of 7 segment patterns.
* These require instead of	07 → ADDR <= 0E22 >	on ACORN	on KIM
	7F → PADD <= 1741 >		

* 1742 SBD → 0E20 (Part A) Digit drive
 1740 SAD → 0E21 (Part B) Segment drive on ACORN/KIM

KIM digit numbers 09, 0B, 0D, 0F, 11, 13
 ACORN digit numbers 00, 01, 02, 03, 04, 05, - six digits.

Fig 1 **Subroutine : CONVD (1F48)**
 This converts one Hex digit to 7 Segment and shows it.

0E80	84FC	Convd.	STY Temp.	Save Y
2	A8		TAY	Get 7 segment pattern and send to display
3	B9 EA FF		LDA Table,Y	
6	8D 21 0E	Convd +6	STA Segmen	
9	8A		TXA }	Change X to value for ACORN
A	4A		LSRA }	
B	38		SEC	
C	E9 04		SBC 04	
E	8D 20 0E		STA Port A	Light display
0E91	A0 7F		LDY 7F	
3	88	Delay	DEY]	Delay
4	D0 FD		BNE Delay	
6	E8 E8		INX INX	Next digit.
8	A4 FC		LDY Temp	Restore Y
A	60		RTS.	

Subroutine : INCPT

0E9B	E6FA	Inept	INC Pointh
D	D0 02		BNE Inept 2
F	E6 FB		INC Pointh
0EA1	60	Inept 2	RTS.
	A2 EA EA EA		NOP

Fig 1 (continued)

Routine SCAND (1F19)

Output to display

0EAS	A0 00 Scand	LDY #00	Get data pointed to LDA (Pointh),Y into display buffer.
7	B1 FA		STA Inh
9	85 F9		
B	A9 07 Scands	LDA #07	Set scan lines
D	8D 22 0E	STA Addr (PADD)	
0EB0	A2 09	LDX #09	Initialise digit zero
2	A0 03	LDY #03	3 bytes to display
4	B9 F8 00 Scand!	LDA Int,Y	Get Byte ready for LSD.
7	4A 4A	LSRA LSRA	M.S.D.
9	4A 4A	LSRA LSRA	display it;
B	20 80 0E	JSR Conv.	
E	B9 F8 00	LDA Int, Y.	Get Byte back.
0EC1	29 0F	AND 0F	Ready for LSD.
3	20 80 0E	JSR Convd.	
6	88	DEY	
7	D0 EB	BNE Scand 1.	
9	8C 21 0E	STY Segmen	
C	EA	NOP	

Kim 'AK' (IEFE) and 'KEY PR' (1F40)

Returns 'A' = \$0 if no key is pressed.
 'A' ≠ \$0 if key is pressed.

0ECD EE FF 0E Key pr INC "TIMER"

D0 20 E5 0E JSR Keyfind.

3 90 02 BCC Exit Key was found: A ≠ 0
 5 A9 00 LDA #0 no key ∴ 'A' ← 0
 7 60 Exit RTS

Fig 1 (continued)

KIM "GETKEY" (1F6A)

Returns 'A' = key if a key is pressed.
 'A' = \$15 if no key is pressed.

0ED8 20 E5 0E Getkey JSR Keyfind.

B B0 05 BCS No key
 D 49 38 EDR #38 Make binary
 F 29 1F AND #1F

0EE1 60 RTS

2 A9 15 LDA #15 Return \$15 for
 no key.

4 60 RTS

Subroutine 'Keyfind' (New)

0EE5 A0 00 Keyfind LDY #0 } Turn display
 7 8C 21 0E STY Segmen } off
 (Part B)

A A2 07 LDX #0 } set seam
 C 8E 22 0E STX ADDR } lines
 (ie PADD)

F 8E 20 0E Seam STX Part A } Read keyboard
 0EF 2 AD 20 0E LDA Part A } at 'X'
 5 29 3F AND #3F Remove top bits.

7 C9 38 CMP #38 Are we pressing
 a key?

9 90 03 BCC Exit Yes we are.
 B CA DEX
 C 10 F1 BPL Seam More scans to do.
 E 60 RTS Carry clear if
 key found.

==== HEX DUMP - ASTEROID =====

```

0200- A9 00 B5 F9 B5 FA 85 FB A2 06 BD CE 02 95 E2 CA
0210- 10 1B A5 E8 49 FF B5 EB A2 05 20 4B 02 20 97 02
0220- CA D0 F7 20 [H] [P] 20 [S] [F] C9 15 10 E5 C9 00 F0
0230- 06 C9 03 DA 00 D0 D8 05 E7 A9 40 C5 E7 00 D3 46
0240- E7 D0 C9 38 26 E7 D0 CA A9 [7] BD [6] [D] A9 [D] 8D
0250- [6] [7] A9 20 85 E0 A0 02 A9 00 85 E1 B1 E2 25 E0
0260- F0 07 A5 E1 19 E4 00 85 E1 88 10 F0 A5 E1 C4 E8
0270- D0 08 A4 E0 C4 E7 D0 02 09 08 [D] [4] [3] A9 [3] 8D
0280- [6] [7] [D] [7] [7] F0 F0 [6] [D] 8D [4] [7] E6 [6] [7] EE
0290- [6] [7] 40 00 C0 60 C6 E9 D0 1A A9 30 85 E9 BA
02A0- 48 A2 FD F8 38 B5 FC 69 00 95 FC E8 D0 F7 D8 68
02B0- AA E6 E2 A5 E2 C9 30 F0 D0 A0 00 A5 E7 31 E2 D0
02C0- 07 60 A9 00 B5 E2 F0 F1 20 [H] [P] 4C C8 02 05 02
02D0- 09 40 01 04 FF 00 00 00 04 00 00 00 06 12 00 11
02E0- 00 05 00 2C 00 15 00 29 00 16 00 29 00 18 00 26 00 19
02F0- 00 17 00 38 00 2E 00 09 00 1B 00 24 00 15 00 39
0300- 00 DD 00 21 00 10 00 00

```

ACORN (Modifications)

```

0200- CA D0 F7 20 [D] [D] 20 [S] [D] etc
0240- .....CA [D] [D] BD [D] [D] A9 [D] [D] BD
0250- [D] [D] etc
0270- .....08 BD [6] [D] A9 [D] [D] BD
0280- [D] etc
0290- [D] [D] 46 BD etc
02D0- .....F1 20 [D] [D] [D] [D] etc

```

Fig 3 ASTEROID for KIM and ACORN

==== HEX DUMP - HI LO =====**KIM**

```

0200 FB A5 E0 38 69 00 A2 01 C9 99 D0 01 8A B5 E0 20
0210 [D] [P] ED D8 A9 99 B5 FB A9 00 85 FA A2 F0 B6
0220 F9 B5 E1 20 [P] [P] 20 [S] [P] C9 13 F0 D3 C5 E2 F0
0230 F2 B5 E2 C9 DA F0 10 B0 EA 0A 0A 0A 0A 0A 0A 03 DA
0240 26 F9 CA 10 FA 30 DC A5 F9 C5 E0 90 06 C5 FB B0
0250 D2 B5 FB A6 E0 E4 F9 90 08 A6 FA E4 F9 B0 C4 B5
0260 FA A6 E1 EB E0 AA F0 B5 D0 B5

```

ACORN

```

0210 [D] [D] etc
0220 F9 B6 E1 20 [S] [D] 20 [P] [D] etc

```

Fig 2 The game HI-LO for KIM and ACORN

Z-80 Processor Profile

Registers

Bank 0

Repeated in
Bank 1

	S	Z	0	H	0	P/V	N	C	Accumulator	A	0
F, Flags											
JS									BC	0	
7		B	0	7			C	0			
JS							DE	0			
7		D	0	7			E	0			
JS							HL	0			
7		H	0	7			L	0			

15	1X	0
Index Register, X		
15	1Y	0
Index Register, Y		
15	SP	0
Stack Pointer		
15	PC	0
Program Counter		
7	Refresh	0
7	R	0
7	I	0
Interrupt vector		

MAIN Z80 CODES

Z80: disp, dd, outc, colors followed by 1 byte, addr, (addr) codes followed by 2 bytes

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
O	NOT	Z80	JR	JR	LD	LD	LD	LD	ADD	SUB	AND	OR	RET	RET	RST	
		disp	HR,disp	HR,disp	B,E	B,B	H,B	(H),B	A,B	B	B	B	NE	NC	RD	
1	LD	LD	LD	LD	I,D	LD	LD	LD	ADD	SUB	AND	OR	POP	POP	RP	
	RL,dd	RL,dd	RL,dd	RL,dd	B,C	D,C	H,C	(H),C	A,C	C	C	C	BC	DE	AF	
2	LD	LD	LD	LD	I,D	LD	LD	LD	ADD	SUB	AND	OR	JP	JP	JP	
	(H),A	(H),A	(H),A	(H),A	B,D	D,D	H,D	(H),D	A,D	D	D	D	NE,addr	NE,addr	RD,addr	
3	INC	INC	INC	INC	LD	LD	LD	LD	ADD	SUB	AND	OR	JP	JP	JP	
	BC	BC	BC	BC	B,E	D,E	H,E	(H),E	A,E	E	E	E	addr	port,A	EX	
4	INC	INC	INC	INC	LD	LD	LD	LD	ADD	SUB	AND	OR	CALL	CALL	CALL	
	B	B	B	B	B,H	B,H	H,H	(H),H	A,H	H	H	H	NE,addr	NE,addr	CALL	
5	DEC	DEC	DEC	DEC	LD	LD	LD	LD	ADD	SUB	AND	OR	PUSH	PUSH	PUSH	
	B	B	B	B	B,L	D,L	H,L	(H),L	A,L	L	L	L	BC	BC	BL	
6	LD	LD	LD	LD	I,D	LD	LD	LD	HALT	ADD	SUB	AND	OR	OR	RD	
	D,dd	D,dd	D,dd	D,dd	(H),(H)	D,(H)	H,(H)	(H),(H)	A,(H)	(H)	(H)	(H)	NE,addr	NE,addr	dd	
7	RLCA	RLCA	RLCA	RLCA	SCF	LD	LD	LD	LD	ADD	SUB	AND	OR	RST	RST	
					B,2	D,2	A,2	H,A	(H),B	A,A	A,A	A,A	NE	NE	JO	
8	EX	EX	EX	EX	JR	JR	LD	LD	LD	ADC	SBC	XOR	CP	RET	RET	RET
	AF,AF'	AF,AF'	AF,AF'	AF,AF'	disp	C,disp	C,B	E,B	L,B	A,B	A,B	A,B	B	Z	PC	M
9	ADD	ADD	ADD	ADD	LD	LD	LD	LD	ADC	SBC	XOR	CP	RET	RET	RET	
	HL,BC	HL,BC	HL,BC	HL,BC	C,C	E,C	E,L	L,C	P,G	A,C	A,C	A,C	C	BC	BC	BL
A	LD	LD	LD	LD	I,D	LD	LD	LD	ADC	SBC	XOR	CP	JP	JP	JP	
	(H),(H)	(H),(H)	(H),(H)	(H),(H)	C,D	E,D	H,D	(H),D	A,D	A,D	A,D	A,D	D	D	D	M,addr
B	DEC	DEC	DEC	DEC	LD	LD	LD	LD	ADC	SBC	XOR	CP	IM	IM	EI	
	B,C	B,C	B,C	B,C	C,E	E,E	H,E	(H),E	A,E	A,E	A,E	A,E	E	E	E	E,addr
C	INC	INC	INC	INC	LD	LD	LD	LD	ADC	SBC	XOR	CP	CALL	CALL	CALL	
	E,C	E,C	E,C	E,C	C,H	E,H	L,H	(H),H	A,H	P,H	H	H	T,addr	T,addr	M,addr	
D	DEC	DEC	DEC	DEC	LD	LD	LD	LD	ADC	SBC	XOR	CP	CALL	CALL	CALL	
	C,E	C,E	C,E	C,E	C,L	E,L	L,L	(H),L	A,L	P,L	L	L	E,addr	E,addr	M,addr	
E	LD	LD	LD	LD	I,D	LD	LD	LD	ADC	SBC	XOR	CP	ADC	ADC	CP	
	(H),dd	(H),dd	(H),dd	(H),dd	C,(H)	E,(H)	(H),(H)	(H),(H)	A,(H)	A,(H)	A,(H)	A,(H)	dd	dd	dd	DD,addr
F	ARLA	ARLA	ARLA	ARLA	CPL	CCF	LD	LD	ADC	SBC	XOR	CP	RST	RST	RST	
					C,A	E,A	E,A	E,A	A,A	A,A	A,A	A,A	BB	BB	BB	BB,addr

CB CODES

E.D. Cope

REVAS & ZEAP

John Haigh

PARKINSON's Revas, the disassembler for Nascom recently serialised in Personal Computer World, produces an assembly listing from a section of machine code. A patch can be added to the routine which displays the assembly listing so that the output is simultaneously stored in the edit buffer of Zeap, the Nascom assembler. The disassembled program can then be edited, labelled, relocated and reassembled as required. The patch provides two start addresses for Revas. The normal start

only sends the output to the VDU. When the 'Zeap' start is used, a test is first made to see if the Zeap edit buffer is empty; an ORG statement is inserted and a line number is entered at the beginning of each line.

Revas does not use standard Z80 mnemonics for some instructions. For example, ADD r is used in place of ADD A,r. Lines containing non-standard mnemonics will be rejected by Zeap on assembly and must be edited to the correct form.

8E61	0010	ESTRT	EQU £8E61	;equates from Revas
8E87	0020	BUFF	EQU £8E87	
1E0D	0030	EDBUFF	EQU £1B0D	:from Zeap
0C0E	0040	ARG2	EQU £C0E	:from Nascom
0232	0050	TBCD3	EQU £0232	
8E00	0060		CRG £8E00	:normal start for
8E00	C39C8F	0070	NORMST	:Revas in 32K system
8F00		0080		:start for Revas
8F00	2A0D1B	0090	ZPSTR	:with Zeap
8F03	11111B	0100		:compare value stored
8F06	B7	0110	LD DE, £1B11	:at £1B0D with
8F07	ED52	0120	OR A	:1B11
8F09	280F	0130	SBC HL,DE	
8F0B	EF1E	0140	JR Z,ZSE-\$:if equal, jump
8F0D	5A..	0150	DEFB £EF,£1E	:if unequal, Zeap
8F17	1F00	0160	DEFM/Zeap error/	:is not ready for
8F19	76 (CF)	0170	DEFB £1F,00	:the program
8F1A	21068E	0180	ZS1	HALT(RST 8)
8F1D	36C3	0190		:use RST 8 with T4
8F1F	215D8F	0200	LD HL, NORMST + 6	:insert jump
8F22	22078E	0210	LD (HL),£C3	:instruction at £8E06
8F25	210100	0220	LD HL, Z INPUT	
8F28	22101B	0230	LD (NORMST + 7),HL	
8F2B	23	0240	LD HL,1	:line number one
8F2C	22AA8F	0250	LD (£1B10),HL	:enter in Zeap
8F2F	21121B	0260	INC HL	
8F32	22180C	0270	LD (LINEO),HL	:initialise LINENO
8F35	060C	0280	LD HL,£1B12	:start of first line
8F37	3620	0290	LD (£0C18), HL	:reset cursor
			LD B,£0C	:clear space to remove
			LD (HL),£20	:zeros, which interfere

8F39	2C	0300	INC L	;with cursor routine
8F3A	10FB	0310	DJNZ-3	
8F3C	EF	0320	RST 40	
8F3D	20,...	0330	DEFM/ORG E/	
8F43	00	0340	NOP	
8F44	2A0E0C	0350	LD HL, (ARG2)	:get start address
8F47	CD3202	0360	CALL TBCD3	:put in CRG line
8F4A	2A180C	0370	LD HL, (£0C18)	:mark end of line
8F4D	2B	0380	DEC HL	:with zero and FF
8F4E	3600	0390	LD (HL),0	
8F50	23	0400	INC HL	
8F51	36FF	0410	LD (HI),-1	
8F53	23	0420	INC HL	
8F54	220D1B	0430	LD (EDBUFF),HL	:start of free memory
8F57	EF1F00	0440	DEFB £EF,£1F,00	:store in EDBUFF
8F5A	C3618E	0450	JP ESTRT	:new line resets cursor
8F5D	D5	0460	ZINPUT PUSH DE	:start disassembly
8F5E	21AA8F	0470	LD HL, LINENO	:this routine copies
8F61	ED5B0D1B	0480	LD DE,(EDBUFF)	:one line from Revas
8F65	1B	0490	DEC DE	:to Zeap buffer
8F66	7E	0500	DEC DE	
8F67	12	0510	LD A,(HL)	:enter line number
8F68	C601	0520	LD (DE),A	
8F6A	27	0530	ADD A,I	:increment line number
8F6B	77	0540	DAA	:convert to decimal
8F6C	23	0550	LD (HL),A	
8F	13	0560	INC HL	:get next digits of
8F6E	7E	0570	INC DE	:line number
8F6F	12	0580	LD A,(HL)	
8F70	3004	0590	LD (DE),A	
8F72	C601	0600	JR NC,ZT1-\$:jump if no carry
8F74	27	0610	ADD A,I	:add carry to upper
8F75	77	0620	DAA	:digits
8F76	21A18E	0630	ZT1 LD HL,BUFF + 26	:start of line in
8F79	7E	0640	ZT2 LD A,(HL)	:Revas buffer
8F7A	23	0650	INC HL	:copy line to
8F7B	13	0660	ZT3 INC DE	:Zeap until a space
8F7C	12	0670	LD (DE),A	:is encountered
8F7D	FE20	0680	CP £20	
8F7F	20F8	0690	JR NZ,ZT2-\$	
8F81	7E	0700	ZT4 LD A,(HL)	:skip consecutive
8F82	23	0710	INC HL	:spaces until next
8F83	FE20	0720	CP £20	:non-space character
8F85	28FA	0730	JR Z,ZT4-\$:is found
8F87	FE3B	0740	CP £3B	:is it a semi-colon?
8F89	20F0	0750	JR NZ,ZT3-\$:if not, continue copy
8F8B	AF	0760	XOR A	:if so, end of line
8F8C	12	0770	LD (DE),A	:mark end of line in

8F8D	3D	0780		DEC A	;Zeap with 00,£FF
8F8E	13	0790		INC DE	
8F8F	12	0800		LD (DE),A	
8F90	13	0810		INC DE	:put start of free
8F91	ED530D1B	0820		LD(EDBUFF).DE	:memory in EDBUFF
8F95	D1	0830		POP DE	:restore DE and HL
8F96	21A18E	0840		LD HL,BUFF + 26	:and return to Revas
8F99	C3098E	0850		JP NORMST+ 9	:output routine
8F9C	21068E	0860	REPTCH	LD HL,NORMST+6	:replace patch for
8F9F	3621	0870		LD (HL),£21	; normal start
8FA1	21A18E	0880		LD HL,BUFF 26	
8FA4	22078E	0890		LD (NORMST + 7),	
8FA7	C3618E	0900		JP ESTRT	
0002		0910	LINENO	DEFS 2	

One problem with Zeap is that although the assembler options are reset to zero each time Zeap is re-entered, the memory offset P is not reset. Having overwritten a program I wished to save by forgetting to reset P, I

decided to add a patch to display the current values of the control parameters. It also shows the starting address of the label store. When recording a program on tape you only need to record from £1BOD to the address LB.

0F6A		0005		ORG £0F6A	
0F6A	FD7E01	0010	WRTOPO	LD A,(Y + 1)	;get assembler options
0F6D	11D00B	0015		LD DE,£BD0	;set cursor
0F70	CD6514	0020		CALL RSTCR1	
0F73	CD4402	0025		CALL B2HEX	;display options
0F76	2A190F	0030		LD HL, (OFFSTO)	;get offset in HL
0F79	11D50B	0035		LD DE,£BD5	;set cursor
0F7C	CD6514	0040		CALL RSTCR1	
0F7F	CD3202	0045		CALL TBCD3	;display offset
0F82	2A240F	0050		LD HL,(LINDEL)	;get delay in HL
0F85	11DC0B	0055		LD DE,£BDC	;set cursor
0F88	CD3202	0060		CALL RSTCR	
0F8B	CD3202	0065		CALL TBCD3	;display delay
0F8E	2A0D1B	0070		LD HL,(£1B0D)	;end of edit buffer
0F91	11E50B	0073		LD DE,£BES	;set cursor
0F94	CD6514	0080		CALL RSTCR1	
0F97	CD3202	0085		CALL TBCD3	;display end address
0F9A	118AOB	0090		LD DE,£B8A	;restore DE
0F9D	C9	0095		RET	

The screen heading, stored between £1003 and £1020, is changed to:—

£1000	4F	20	20	20	20
£1008	50	20	20	20	20
£1010	20	20	20	20	20
£1018	20	20	20	20	20
£1020	4D				

The subroutine WRTOPO is called by a patch added at £1889.

1889	210310	0100	LD HL,TITLE	:address in Zeap
188C	11CF0B	0105	LD DE,EBCF	:position on screen
188F	011E00	0110	LD BC,30	:number of characters
1892	EDB0	0115	LDIR	:copy to screen
1894	CD6A0F	0120	CALL WRTOPO	:insert values
1897	CD6514	0130	CALL RSTCR1	:reset cursor
189A	;end of patch			

The third patch adds two editor commands; L lists the labels used in a program, and M outputs the label list to a printer. When adding this and the previous patch two points should be noted. Firstly, when any change is made

to the Zeap program between £1000 and £1BOD a checksum error message (Error 90) will appear on the screen when Zeap is restarted. Secondly, the patches themselves are not protected by the parity check.

0E80		0005	ORG £E80	
0E80	FE4C	0010	LABOUT CP "L	:test command character
0E82	280C	0015	JR Z LABLST	:if L, list labels on VDU
0E84	FE4D	0020	CP "M	:if M, print labels
0E86	C26817	0025	JP NZ DEFOLT	:jump to default routine
0E89	FDCB00D6	0030	SET 2,(IY)	:set output flag
0E8D	CDB213	0035	CALL CARRT	:output £1F, £0D, and £0A
0E90	DD2A0D1B	0040	LABLIST LD IX,(EDBUFF)	:IX points to label addresses
0E94	210F1B	0045	LD HL,EDBUFF + 2	:start of listing
0E97	CD8813	0050	LBL0 CALL SRCH1	:find next line number
0E9A	CA1A18	0055	JRZ ST1	:if zero, end of listing
0E9D	5E	0060	LD E,(HL)	:get line number in DE
0E9E	23	0065	INC HL	
0E9F	56	0070	LD D,(HL)	
0EA0	23	0075	INC HL	
0EA1	E5	0080	PUSH HL	:gave address of line start
0FA2	ZA0C0C	0085	LD HL,(ARG1)	:first line requested
0EA5	ED52	0090	SBC HL, DE	:compare with current line no.
0FA7	E1	0095	POP HL	:recover start of line
0EA8	280D	0100	JR Z,LBL1	:jump if current line no. is
0EAA	380B	0105	JR C,LBL1	:equal or greater
0EAC	7E	0110	LD A,(HL)	:test first character of line
0EAD	FE40	0115	CP £40	:is it a letter (i.e., a label
0FAF	38E6	1020	JR C,LBL0	:if not, get next line
0EB1	DD23	0125	INC IX	:if so, skip label address
0EB	DD23	0130	INC IX	
0FB5	18E0	0135	JR LBL0	:get next line
0EB7	2B	0140	LBL1 DEC HL	:decrement HL to line number
0EB8	2B	0145	DEC HL	
0EB9	CD7114	0150	LBL2 CALL ENDLIN	:have we reached last line?
QEBC	DA1A18	0155	JP C, ST1	:if so, stop
0EBF	23	0160	INC HL	:increment HL to first
0EC0	23	0165	INC HL	:character of line
0EC1	7E	0170	LD A,(HL)	:test first character
0EC2	FE40	0175	CP £40	:is it a letter?
0EC4	382E	0180	JR C,LBL4	:if not, jump

0EC6	E5	0185	PUSH HL	:save address
0EC7	EB	0190	EX DE,HL	:put line number in HL
0EC8	CD3202	0195	CALL TBCD3	:display line number
0ECB	E1	0200	POP HL	:recover address
0ECC	11980B	0205	LD DE, £B98	:reset cursor to £B98
0ECF	CD6514	0210	CALL RSTCR1	
0ED2	11900B	0215	LD DE,£B90	:start of second column
0ED5	CD8214	0220	CALL PRTCHR	:display the label
0ED8	38FB	0225	JR C,LBL3	:
0EDA	EF2300	0230	DEFB £EE,£23.00	:pound sign denotes hex.
0EDD	E5	0235	PUSH HL	:save current address
0EDE	DD6E00	0240	LD L,(IX)	:put label address into HI
0EE1	DD6601	0245	LD H,(IX 1)	
0EE4	CD3202	0250	CALL TBCD3	:display label address
0EE7	E1	0255	POP HL	:recover current address
0EE8	DD23	0260	INC IX	:increment IX to next
0EEA	DD23	0265	INC IX	:label address
0EEC	3EA0	0270	LD A,£A0	:£A0 marks end of line
0EEE	CD3B01	0275	CALL CRT	
0EF1	CDC618	0280	CALL WTLBIT	:output the line
0EF4	CD8813	0285	CALL SRCH1	:find next line
0EF7	18C0	0290	JR LBL2	:continue

**CRYSTAL ELECTRONICS
CC ELECTRONICS**

The newest Z80 Basic! **XTAL Basic 2.2**

HAS to be the best yet for your Nascom 1 or 2. All the usual features of other 8K floating-point BASICs Plus: Extra commands/functions—INCH, KBD, CMD\$ ON ERR GOTO, ERR, PI, CLOAD? (tape verify)

And Add up to 64 reserved words of your choosing—Now put your own disc, tape, control, graphics commands, etc for the ULTIMATE in BASIC flexibility! Fully upward compatible with version 2.1 (see earlier Ads). Can be easily adapted to most Z80 systems. Works with T2, B-BUG, T4 and NAS-SYS monitors. Existing version 2.1 users—Return your original tape (less manual) with 50p P&P and we will update it FREE of charge!

Price: still only £35 + VAT!

CREED PRINTER INTERFACE

For NASCOM or APPLE—lowest cost hard copy! Complete kit of parts (with software) £18 + VAT.

16 CHANNEL RELAY BOARD

Now in stock for NASCOM 1/2. For £49.95 + VAT. Sixteen switched (isolated) channels for many control applications. This kit will greatly increase the flexibility of your NASCOM.

Members of Computer Retailers Association & Apple Dealers Association

Shop open 0930-1730 except Wed. & Sun.
40 Magdalene Road, Torquay, Devon, England. Tel: 0803 22699

Access and Barclaycard welcome.



**You've heard about it
Read about it — HERE IT IS**

NASCOM-2 SCOOP OFFER
ONLY 500 KITS AVAILABLE AT THIS PRICE

AVAILABLE EX-STOCK
COMPLETE KIT AS PER
MANUFACTURER'S SPECIFICATION
With provision for RX on board expansion. Excludes 4110 ~~xc+~~

INCLUDES FREE 16K EXPANSION
Value £140 includes ALL parts with every kit

NASCOM-2 ON DEMONSTRATION NOW

AVAILABLE ONLY FROM US ON THE COUPON BELOW

OPTIONAL EXTRAS	8 OFF 4110*
3 AMP POWER	+ 15% WITH VAT FREE
SUPPLY	VAT 15%
£29.50	Post £1.50
	For NASCOM-2

8 OFF 4110*	For NASCOM-2
For PRU Chassis	Early Delivery
£80	

RS232 COMPATIBLE 80 COLUMN PRINTER

brand new
List price £550. If sent by carrier £5 extra

OUR PRICE
£325 + VAT

FULL MANUFACTURER'S WARRANTY — DON'T DELAY, ORDER TODAY
Please send me my **NASCOM-2 KIT** with the **FREE 16K EXPANSION** for £295 + VAT.

I enclose remittance

Name & Address

Also in stock **NASCOM-1 • ELF • TRS80** as previously advertised



Henry's

Computer Kit Division
404 Edgware Road, London, W2, England
01-402 6822



Application Software for Micro Computers

PAUL RAYNER

Paul Rayner is the Managing Director of Great Northern Computer Services Ltd. This article is derived from his experience in developing and installing high quality business software.

THERE are a number of problems that face the designer of application software when he starts to work with micro computers which do not face his colleagues who are working with larger 'mini' and 'mainframe' computers. The most obvious problems stem from the characteristics of the equipment itself.

Micro-computers are small. Memory size available to any one user is normally limited to 64K. Up to half of this is taken up by the operating system and the language interpreter or compiler. This leaves 32K of 8 bit words for application programs. The result is that systems must be split up into separate programs that are 'CHAINED' together.

Disk storage is limited. Double and quadruple density diskettes are reducing this problem and, when proper back-up facilities are available, fixed 'Winchester' disks may remove it completely. But for the time being most users must fit their applications into less than 1 megabyte of disk storage.

This is adequate for simple systems for small businesses, but insufficient for complex systems. For example a 1 megabyte, two drive system can hold 5,000 part numbers in a simple stock control system. In a more complex system, with all outstanding orders and requisitions stored in a chained and indexed 'data base' structure, only about 2,000 part numbers could be held.

Rarely can a single file be spread over more than one diskette. This limits the maximum file size to 250,000 or 500,000 characters. To store more items (say 5,000 stock records of 150 characters each in a stock control system) some form of segmented file must be used. By this method the stock file becomes a series of small files scattered over the available drives, linked together by an index.

Limited disk space, combined with limited memory, can be a problem if one needs to sort large data files. There is neither the space for long strings in memory nor for large files on disk. One way of overcoming the problem is not to sort at all. Instead the necessary processing can be done by repeatedly scanning the file and pulling

out successive items on successive scans. This is very slow but can be acceptable in jobs performed infrequently, such as month-end audit reports which are printed out after everything else has been processed.

On the whole, the micro-computer systems designer has time on his side. Volumes are relatively small and so he can use methods which are slow in order to save disk or memory space. The user who is replacing a manual system, which takes five days to do month-end accounts, will not complain if the micro-computer based replacement system takes five hours as opposed to four hours. Thus with skill and ingenuity in system design and programming, all the problems outlined above can be solved.

The more serious problems are not so easily dealt with. These are the ones that stem from the characteristics of the micro-computer market and from the characteristics of the people who use them.

The hardware for a good, reliable business micro-computer now costs around £5,000. This is a quarter of the price that would have been paid for a system with the same power only a few years ago. Unfortunately users expect the cost of software to have fallen in equal proportion. Thus the market place demands that software which could be sold for several thousand pounds if supplied to an IBM or ICL mainframe installation, can only be sold for a few hundred pounds when supplied to a micro-computer user.

The 'democratisation' of computing, which has resulted from the spread of micros, means that there are many people around with a basic knowledge of computers and BASIC programming. Such people provide a pool of part-time or 'moonlight' programming resource at 25% to 30% of the going rate for a full-time professional. Although such part-timers do not have the same depth of experience as the professional, and may not be available to help when the system comes unstuck at its first financial year-end, their effect is to depress the price that can be charged for quality commercial software.

The way out of this profit squeeze is for the software house to offer standard packages at low price. Instead of

writing one system and selling it with some modifications to ten mini-computer users at £2,000 each, the professional software house must sell 80 packages with no modifications at £250 each.

For a package to sell in such volumes it must work with the widest possible range of equipment configurations and user requirements. This means that it must be designed to work with the typical commercial micro configuration; viz 8080/Z-80 processor, CP/M operating system, 48K memory and twin drive diskette unit. Such a computer will come with one of the standard BASICs, such as CBASIC2, and any of a wide range of V.D.U. and printer.

Because every V.D.U. has a different set of commands and command codes, packages should avoid using special V.D.U. features. This rules out 'sexy' screen formatting, but the benefit of such formatting is largely cosmetic, and, most users will not notice the loss. With languages such as CBASIC, those V.D.U. based commands, which have to be used, (such as clear screen), can be put into a sub-routine. The one subroutine can then be adjusted to suit the users V.D.U. and the altered code automatically 'INCLUDEd' in all programs when they are compiled.

One cannot, in Britain, assume that users will all have wide-carriage printers. This is a contrast to the U.S.A., where most business systems include a 'daisy wheel' printer, and all have 132 or 120 column printers. Here it is best to design for 75 to 80 column printers. Such reports will also file nicely into A4 size ring binders, and, are thus much easier to read and transport than conventional full-width computer reports.

The biggest source of problems for the micro-computer systems designer is the user himself. Mini-computers or mainframe are run by trained data processing specialists. Micro-computers, by contrast, are run by existing staff, who have no special training, and, who all have other work to do. There is thus a premium on simplicity, ease-of-use and robustness.

In a small business there are not the clear demarcations of jobs that are found in larger organisations, for small businesses cannot afford specialists. Instead staff have to perform several functions. This will also apply to computer operation. Whilst one person should have overall responsibility for ensuring proper back-up of files, servicing, etc., any responsible member of staff, from Managing Director to secretary, should be able to use it.

It follows that the system must be easy to use. It must also be well documented, but, ideally, so designed that the user rarely needs to refer to the manual, since full instructions will be given on the V.D.U. screen.

The system should be 'foolproof'. If, for example, items should balance out to zero, then the program should check that they do and reject them if they do not. As another example, if the user is entering an account number, the program should verify that it exists, and should print out the account name so that the user can check visually. All this must be done without making the programs so large that they no longer fit within available memory.

Another characteristic of small business operation is that there are many interruptions. Staff must switch from job to job in response to events. Thus, for example, in the middle of entering the week's invoices the user must be able to stop and printout the current status of an account before returning to enter the rest of the invoices. This really means 'in-situ' updating of files. The alternative is a long, drawn-out process of entering data, sorting and updating before any enquiry can be made.

In-situ updating also means that if the computer is accidentally switched off, or, if there is a power cut, then at most only the last transaction entered is lost.

Probably the most important requirement of micro-computer application software is simplicity. Users rarely have either the time or the need for sophisticated systems. Often they are going from very basic manual or accounting machine systems. Thus the ideal system should require no changing of diskettes. Instead it should run just two diskettes; one for programs and one for data: because where diskettes need to be changed in the middle of a run there is grave risk of confusion and error.

The actual features within the system need to be kept simple. This can be difficult. For example, most users want to get the benefits of 'Open Item Sales Ledgers'. Yet the number of possible transactions on such a system (e.g. cash against earliest, cash against month, cash against invoice, overpayments, underpayments, etc.) are so great, and, the results of incorrect posting so potentially chaotic, that most micro-computer users are better off in the long run with a simple but effective 'Balance Forward' system.

With the basic equipment and the software so relatively inexpensive, the cost of media becomes significant. It is thus important to consider the cost of stationery. Wherever possible standard stationery should be used. The packages marketed by the Author only need one pre-printed form for all invoices and statements. These forms cost under £40.00 per 1,000. Even so, a user sending out only 600 statements a month can spend more on pre-printed stationery in one year than the total cost of the Sales Ledger package.

No matter how well the package is designed, and no matter how thorough the user manual, there will be times when the user needs training and support. Clearly the software house cannot provide such support within the prices discussed. For this reason it makes sense for packages to be supplied, along with the hardware, through reputable computer dealers. The dealer will then provide the training and the practical support during the first traumatic weeks of operation. If supplied with source code the dealer may also be able to make special 'customization' required by the user. In return the dealer will get a discount on the packages and will be able to sell more hardware by having good quality application software available off-the-shelf.

Selling standard packages through established dealers is very much the way that the problem of supplying application software for micros has been solved in the U.S.A. It is the Author's belief that this is also the best way to solve the problem in Britain.



Pets Corner J.Stout

BEFORE starting on the software for the PET in this issue, it should be said that the contents so far (in two issues) reflect very closely my interests as far as the PET is concerned, that is machine code and the writing of utility programs (programs to help other people write programs). Until readers start sending me some indication of their interests, if not actual articles to publish, that is the way it will stay.

THE PET REVEALED

This is an A5 size book, of about 260 pages, published by COMPUTABITS, LTD., P.O. BOX 13, SOMERSET, ENGLAND, for £10.00. The acknowledgements describe it as 'a collection of discoveries about the PET, how and why it works, and how to use these facts to write better programs and perform more interesting functions'.

After a fairly brief inspection of the book the following points should be made:

- 1 The book is written for the owner of a PET with OLD ROMs. A NEW ROM version should be out in the New Year. Nowhere in the advertising (principally Practical Computing) does it make this clear.
- 2 Its main interest would be to someone who is happy to use machine code since although it does provide a number of BASIC programs which can be typed in, the majority of information it provides (and it does provide a lot) would need to be applied in some specific way to be useful.
- 3 The book is well presented and printed, although the type face is a little small, but the binding used, gluing the pages into the cover, although presumably cheaper than other forms, means that very soon the pages start to fall out.
- 4 A lot of the book is concerned with hardware, both the hardware internal to the PET and external add-ons. Since one of the reasons many people have bought the PET is so that they need not get involved

with the hardware, this approach seems at odds with the advertising for the book, which describes it as 'ESSENTIAL READING FOR ALL PET OWNERS'.

The general impression I gained of the book is that it would be ideal for someone who was accustomed to working in machine code and/or with a soldering iron, but that for someone who is happy with BASIC and has no wish to look inside the black box it is probably a waste of money.

There seem to be few actual mistakes in the book, although the text sometimes reads as though someone has missed out the punctuation. The section on 'adding new commands to BASIC' simply tells you how to link into the BASIC, not how to implement your own commands, and also leaves some questions unanswered, e.g. how do you make sure that your added command does not interfere with BASIC itself, how do you check that it is one of the extra commands that is to be executed and not a standard BASIC command.

To anyone thinking of buying the book I would be inclined to say that they should try and see a copy of it first before parting with £10. As 'essential reading for all PET users' it does not seem worth the money, but as a compilation of all the machine code facts that have appeared in numerous other magazines and books it may well be.

News from Canada

We are fortunate to have another letter from Jim Butterfield in Canada. He enclosed the following list of known entry points into the PET ROMs, which will be of great help to programmers in machine code. Although some supplementary work will need to be done to discover how to use the routines properly, at least they have been located. He also offers the following advice to such programmers:

ZERO PAGE usage with the new ROMs

- i. If you are not using tape I/O, help yourself to locations \$B1 to \$C3 (177-195).
- ii. If you do not use any ROM routines, locations \$00 to \$8C (0-140) may be swapped out, i.e. saved elsewhere, perhaps on the stack, and then restored before return to BASIC.
- iii. Don't touch \$8D to \$B0 or \$C4 to \$FA (141-176 or 196-250), unless the interrupts are totally disabled, disabling keyboard, clock update and cursor flash.
- iv. The only real need for page zero locations now is for indirect pointers, i.e. the (.X) and (.Y) addressing modes of the 6502, since everything else can be moved elsewhere in memory without much effect on speed, although with a 50% increase in memory on instructions which reference data stored elsewhere. Since page zero space is now hard to get use it sparingly.

A few entry points, original/upgrade ROM Jim Butterfield

Entry points seen in various programmer's machine language programs. The user is cautioned to check out the various routines carefully for proper setup before calling, registers used, etc.

ORIG UPGR	DESCRIPTION
C357	C355 ?OUT OF MEMORY
C359	C357 Send Basic error message
C38B	C389 Warm start, Basic
C3AC	C3AB Crunch & insert line
C430	C439 Fix chaining & READY.
C433	C442 Fix chaining
C48D	C495 Crunch tokens
C522	C52C Find line in Basic
C553	C55D Do NEW
C56A	C572 Do CLR
C59A	C5A7 Reset Basic to start
C6B5	C6C4 Continue Basic execution
C863	C873 Get fixed-point number from Basic.
C9CE	C9DE Send Return,LF if in screen mode
C9D2	C9E2 Send Return, Linefeed
CA27	CA1C Print string
CA2D	CA22 Print precomputed strings
CA49	CA45 Print character
CE11	CDF8 Check for comma
CE13	CDF8 Check for specific character
CE1C	CE03 'SYNTAX ERROR'
D079	D069 Bum ⁿ Variable Address by 2
D0A7	D09A Float to Fixed conversion
D278	D26D Fixed to Float conversion
D679	D678 Get byte to X res
D68D	D68F Evaluate String
D6C4	D6C6 Get two parameters
D73C	D773 Add (from memory)
D8FD	D934 Multiply by memory location
D9B4	D9EE Multiply by ten
DA74	DAAE Unpack memory variable to Accum #1
DB1B	DB55 Completion of Fixed to Float conversion

As a postscript to Jim's advice it is worth pointing out that most if not all of the problems in converting machine code programs to new ROMs from the old ones are concerned with page zero.

For readers who have encountered the PETUNIA or MTU music systems (which use an 8-bit digital to analog interface on the user port, Jim reports that Frank Covitz, who is responsible for most of the music currently available for the system, is putting the finishing touches to an advanced version, whose quality of sound Jim reports as being 'astonishing'. It will be available through MTU.

We have permission to reprint the circuit diagram of the interface from September 1978 BYTE, and are at the moment looking round for some software to go with it.

DC9F DCD9 Print fixed-point value
 DC99 DCE3 Print floating-point value
 DC9F DCE9 Convert number to ASCII strings
 E3EA E3D8 Print a character
 na E775 Output byte as 2 hex digits
 na E7A7 Input 2 hex digits to A
 na E7B6 Input 1 hex digit to A
 F0B6 F0B6 Send 'talk' to IEEE
 F0BA F0BA Send 'listen' to IEEE
 F12C F128 Send Secondary Address
 E7DE F156 Send canned message
 F167 F16F Send character to IEEE
 F17A F17F Send 'untalk'
 F17E F183 Send 'unlisten'
 F187 F18C Input from IEEE
 F2C8 F2A9 Close logical file
 F2CD F2AE Close logical file in A
 F32A F301 Check for Stop key
 F33F F315 Send message if Direct mode
 na F322 LOAD subroutine
 F3DB F3E6 ?LOAD ERROR
 F3E5 F3EF Print READY & reset Basic to start
 F3FF F40A Print SEARCHING...
 F411 F41D Print file name
 F43F F447 Get LOAD/SAVE type parameters
 F462 F466 Open IEEE channel for output.
 F495 F494 Find specific tape header block
 F504 F4FD Get string
 F52B F521 Open logical file from input parameters
 F52D F524 Open logical file
 F579 F56E ?FILE NOT FOUND, clear I/O
 F57B F570 Send error message
 F58E F586 Find any tape header block
 F64D F63C Get pointers for tape LOAD
 F667 F656 Set tape buffer start address
 F67D F66C Set cassette buffer pointers
 F6E6 F6F0 Close IEEE channel
 F78B F770 Set input device from logical file number
 F71C F7BC Set output device from LFN.
 F83B F812 PRESS PLAY.; wait
 F87F F855 Read tape to buffer
 F88A F85E Read tape
 F8B9 F886 Write tape from buffer
 F8C1 F88E Write tape, leader length in A
 F913 F8E6 Wait for I/O complete or Stop key
 FBDC FB76 Reset tape I/O pointer
 FD1B FC9B Set interrupt vector
 FFC6 FFC6 Set input device
 FFC9 FFC9 Set output device
 FFCC FFCC Restore default I/O devices
 FFCF FFCF Input character
 FFD2 FFD2 Output character
 FFE4 FFE4 Get character

MENUS and the SCANNER alternative

To simplify the choices a user faces in running a program the concept of a menu is often used. The possible alternatives are displayed together with some sort of key, either numerical (1 Compiler, 2 Editor etc) or

alphabetical (C Compiler, E Editor etc). By pressing one key on the keyboard the relevant option can be called up and executed. Whilst simple to program and relatively foolproof in operation, it makes the user adapt to the computer, rather than the computer to the user.

The alternative method illustrated here uses the idea

of a SCANNER to break down a line of natural English into its component parts (also known as tokens or atoms). A token might be a number, a reserved word in the input language, or an identifier. Thus the user of an Adventure-type game might enter 'TAKE 3 PACES NORTH' and 4 successive calls to the SCANNER would return the reserved word 'GO', the number 3, the reserved word 'PACES' and the reserved word 'NORTH'. For an assembler the input line might be 'LDA (\$10,X)' and calls to the SCANNER would produce 'LDA', '(16 (the SCANNER is the best place to do hexadecimal-decimal conversions so that the rest of the assembler can work in decimal).)', 'X' and ')'. In order to make transfer of control easy for BASIC a number might be returned describing which reserved word had been input, so than an ON...GOTO or GOSUB could be used.

The program listed illustrates the use of a SCANNER, and also includes the general line input routine mentioned in last issue. Subroutine 50100 returns in LS the string entered by the user, with a null byte and a space appended to the end. LL is the total length of the string and CP (for Character Pointer) is set to zero, so that the SCANNER's first call to subroutine 50000 will result in the first character of LS being returned in CH\$. Subroutine 50100 allows the use of the delete key, but ignores all other cursor control characters. A string greater than 254 characters in length reports an error and restarts. The only other peculiarity is that " is changed to ' (double to single quotes). Commas, semicolons etc are dealt with normally, so BASIC's aversion to dealing with commas etc in sentences (unless enclosed within quotes) is avoided.

The SCANNER program is borrowed almost unchanged from the P-Compiler in the October 1978 issue of BYTE (Mentioned In Last Issue). It is set up to return tokens from Pascal program lines, and in line 10 of the complete program N is the number of reserved words, LR is the maximum length of a reserved word and ML is the maximum length of an identifier (e.g. a variable name). RW\$ is an array holding the reserved words. In line 20 SPS becomes a string of ML spaces, used to left justify reserved words with less than LR characters. Lines 30-80 initialise the reserved word array, while line 80 initialises the SCANNER variables. Line 90 calls the SCANNER and prints the results from TKS(for ToKen string), SG\$ (for StrinG string) and NR (for NumbeR). If the first byte of TKS is a null then we have reached the end of a line (in Pascal this does not really matter, and was not included in the original SCANNER). If the first byte of TKS has an ASCII value between zero and thirty-two it represents an error. TKS = "%" is used as the terminator for the program.

50000 is the subroutine to get the next character from the line buffer LS and move it to the character buffer (or look-ahead) CH\$. If all the characters in LS have been used (CP greater than or equal to LL) then a call is made to 50100 to clear LS and read in a new line.

50100 as described before reads in an input line into LS.

50300 is the SCANNER proper. 50320 skips leading spaces. 50330 checks for a first significant character being a letter, in which case the letters and digits following it are added into SG\$ to form the identifier or reserved word. Only the first ML characters are significant. A search is then made in lines 50390 to 50420 for a reserved word corresponding to the identifier. A possible alteration here would be to have in line 50340 RP = 0 and in line 50410 RP = RW; RW=N etc, so that on exit, if RP = 0 then we do not have a reserved word, otherwise RP acts as an index to the array of reserved words and can be used in an ON...GOTO statement. If the identifier is a reserved word then TK\$ is set to the reserved word, otherwise TK\$ is set to "IDENT" and SG\$ contains the identifier name.

If the first significant character is a digit the SCANNER assumes a number is being input, sets TK\$ to "NUM" and inputs the number to NR (lines 50450 to 50470). The number must be integer, and is limited to 10 digits. Lines 50480 to 50580 check for special symbols, which are written as a combination of two characters but stand for 1 token, e.g.: = . <> (not equal). Lines 50600 to 50620 take ' as introducing a string any build up the string into SG\$. The error test in line 50620 will never be used (Why?). Lines 50650 to 50660 ignore comments (surrounded by exclamation marks) and lines 50680 to 50740 handle hexadecimal constants, which are preceded by %. If the input character does not fit into any of the categories mentioned it is simply returned as the token in 50670.

Notice the way that whenever the SCANNER has used a character it calls 50000 in order to get the next, but if it does not use it CH\$ is left unchanged. In this way the routine can look 1 character into the future, in order to find out what is in store for it.

Further thoughts

Besides the array of reserved words the SCANNER could use an array of meaningless words, e.g. 'A', 'THE' etc, which it would pass over. Some type of scanner must of course be used in the input phase in BASIC, where the program text is turned into a series of tokens and stored in the memory. Similarly the famous ELIZA program probably contains one, although much more complicated than this. A thought for future expansion might be a scanner whose array of reserved words expands as type goes on, for example in an Animal game. For large sizes of arrays a more efficient search function should be used, e.g. binary search or hashing. A detailed look at the two complementary topics of searching and sorting will feature in the next PETs Corner.

ALTERATIONS FOR NEW ROMS

Lines 50120 and 50130: 548 should be 167. The cursor character in line 50130 is cursor left.

```

READY.
10 N=32:LR$="":ML=8:DIM RH$(N)
20 SP$="" :FOR I=1 TO ML:SP$=SP$+" " :NEXT I
30 FOR I=1 TO N:READ RW$(I):NEXT I
40 DATA "AND ","ARRAY,BEGIN,"CALL ","CASE ","CONST,"DIV ","DO ","DOWNT,"ELSE "
50 DATA "END ","FOR ","FUNC ","INTEG "MEM ","MOD ","NOT ","OF "
60 DATA "OR ","PROC ","READ ","REPER,"SHL ","SHR ","THEN ","TO ","TYPE "
70 DATA UNTIL "VAR ":"WHITE WRITE
80 CH$="" :CP=0:LL=0
90 GOSUB 50300:PRINT TK$,SG$,NR
95 IF (ASC(TK$)=0) THEN PRINT "END OF LINE":GOTO 90
98 IF (ASC(TK$)<32) THEN PRINT "ERROR":ASC(TK$):GOTO 90
100 IF (TK$>?") THEN 90
110 STOP
50000 REM GET A CHARACTER FROM THE LINE BUFFER L$.
50010 IF (CP>LL) THEN GOSUB 50100:GOTO 50010 REM IF PAST END OF LINE GET ANOTHER
50020 CP=CP+1:CH$=MID$(L$,CP,1):RETURN:REM MOVE TO NEXT CHARACTER AND READ TO BUFFER.
50030 REM READ A LINE TO VARIABLE L$.
50040 L$="":REM LINE INITIALLY IS NULL.
50050 IF (ASC(CH$)>>20) THEN 50190:REM DELETE KEY HAS ASCII CODE 20.
50060 IF (LEN(L$)=0) THEN 50120:REM NOTHING IN LINE SO CAN'T DELETE.
50070 PRINT CH$;:IF (LEN(L$)=1) THEN 50110:REM IF ONLY 1 CHARACTER IN LINE REST
50080 L$=LEFT$(L$,LEN(L$)-1):GOTO 50120:REM DELETE LAST CHARACTER.
50090 IF (ASC(CH$)<32) THEN 50120:REM IGNORE CONTROL CHARACTERS.
50100 PRINT CH$;:L$=L$+CH$:IF (LEN(L$)<255) THEN 50120
50110 PRINT :PRINT "LINE TOO LONG":GOTO 50110 REM AND START AGAIN.
50120 PRINT :L$=L$+CHR$(0)+" " :L$=LEN(L$):CP=0:RETURN:REM NULL LINE=CHR$(0)+" ".
50130 REM GET A TOKEN (ATOM) FROM THE LINE IN L$.
50140 REM RETURN TK# FOR TOKEN TYPE, SG# FOR STRING, NR FOR NUMBER VALUE.
50150 IF (CH$=" ") THEN GOSUB 50000:GOTO 50320 REM FLUSH SPACES.
50160 IF (CH$<"A">OR(CH$)>"Z") THEN 50430:REM NOT A LETTER.
50170 LI=0:SG$="":REM LI=LENGTH OF IDENTIFIER.
50180 IF (LICHL) THEN LI=LI+1:SG$=SG$+CH$:REM ML=MAXIMUM LENGTH OF IDENTIFIER.
50190 GOSUB 500000 IF ((CH$)="A">)AND(CH$)<"Z") OR((CH$)="0">)AND(CH$)<"9") THEN

```

MICRODIGITAL 1980

Apple II plus



Apple II Plus will change the way you think about computers. That's because it is specifically designed to meet the day-to-day activities of education, business, financial planning, scientific calculations and more.

APPLESOFT

A fast, extended 10K BASIC with 9-digit precision and graphics extensions. 10K BASIC with 16K memory expansion. On a matrix of 280 x 192 individually addressable points.

ALL COLOR RGB

With power over 100 applications programs, nest protection and improved memory management.

INTERNAL MEMORY EXPANSION TO 64K BYTES

Fast processing performance at a low cost EIGHT EXPANSION SLOTS

To let the system grow with your needs

Net V.A.T. Total

Apple II Plus 495.00 104.25 599.25
16K RAM ... 495.00 104.25 599.25

APPLE PASCAL

Apple Pascal is the new extension to microcomputer power.

Pascal Incorporating UCSD PASCAL™, the most advanced features in a complete interactive package now available today's most sophisticated structured programming language. It provides a fast, efficient environment for rapid performance and cut development time for large business, scientific and educational programs.

This software package provides the most powerful set of tools available for the microcomputer programmer.

Apple Pascal Net V.A.T. Total
299.00 64.85 343.85

FLOPPY DISCS

Give your system immediate access to large quantities of data. The subsystem consists of a 5 1/4" floppy disk drive, a powerful Disk Operating System and one or two mini-floppy drives.

Net V.A.T. Total

Floppy disk Subsystem ... 349.00 52.85 401.35
Second disk drive and connecting cable ... 299.00 44.85 343.85

Parallel Printer Interface Card

Allows you to connect almost any parallel printer to your Apple. A BASIC program can print directly from the printer as easily as to print to the TV monitor screen. Command and control codes and control details are handled by the software built into the card, to eliminate user programming requirements.

Parallel Printer Interface Card Net V.A.T. Total
101.00 13.60 115.60

Communication Interface Card
Allows your Apple to "talk" through a modem with other computers and terminals over ordinary telephone and baseband lines. Send and receive messages to remote terminals or access your office computer from the comfort of your home.

Net V.A.T. Total

Communications Interface Card ... 130.00 19.50 149.50

Nascom 2

Microprocessor
2MHz 8 bit CPU. This will run at 4 MHz but is selectable between 1/2-4 MHz.
Hardware
12 x 8 Card
All bus lines tie to the Nascom expansion slots.
All bus lines are fully buffered.
Memory
One board, addressable memory:
2K Video RAM - Nascom 2
1K Video RAM (MX418)
1K Work space/User RAM (MX418)
1K ROM (MX418)
16K ROM (MX418)
8K Basic RAM/276 EPROM



Keyboard
New expanded 97 Key Linear solid state keyboard especially built for Nascom. Uses standard Nascom, monitor connection, encoding.

T.V.
The T.V. Peak to peak video signal can drive a standard monitor or it can be fed to a line card module to drive the domestic T.V.

I/O
One board UART (Intel 6402) which provides serial handling for the on-board cassette interface or the RS 232-20mA serial interface. The cassette interface is Nascom standard with either 1200 or 300 baud. This is a little operation on the Nascom 2.

There is also a totally uncommitted PIO (MK3881) giving 16 programmable I/O lines.

Character Generator
The 1K video RAM drives a 2K ROM character generator providing the standard ASCII Character set and some additional 128 characters. In all, there is a second 2K ROM socket for an on-board graphics package which is software selectable.

Net V.A.T. Total
Nascom 2 in Kit 299.00 44.25 339.25
Power Supply 25.00 3.68 28.68
Graphics ROM 15.00 2.25 17.25

Superboard II

The sensational single board computer from Chris Scientific. Superboard comes fully assembled and tested. On board is a 68000 processor, 128K RAM, 128K ROM, a board to 88K Microsoft BASIC in ROM, CRTC cassette interface, full ASCII keyboard, 128x25 character display with a video monitor or domestic television (via UHF Modulator) and provides a 24 x 24 dot matrix display. Superboard also has a wide range of graphics/gaming characters.

Superboard comes complete with documentation and sample software on cassette.

Net V.A.T. Total
Superboard II ... 168.00 28.20 216.20
U.H.F. Modulator 2.95 0.38 2.98

Video Genie

A third generation personal computer system, the video genie is a powerful system and is completely compatible with the Tandy TRS 80 TM.

Central Processor
2MHz 8 bit CPU. A system monitor is mounted at the rear of the console. Power supply is NOT required should the system ever crash.

Video Display
16 lines of 32 (2 pixel) or 64 characters, switch selectable. Full software cursor control.

Composite video output to a domestic television

Memory
RAM - 2K Screen RAM
16K User RAM

ROM - 12K Extended Level II BASIC interpreter, system monitor, interrupt routine, 16K ROM, 8K ROM, 8K ROM Level II BASIC.

Cassette
Internal 500 b.p.s cassette deck encloses tape and circuitry. Additional interface for second external cassette deck. Manual controls of cassette deck and monitor. Cassette deck is automatically associated with the storage medium.

Keyboard
An extended Level II BASIC compatible with TRS 80 Level II BASIC.

Features line editing, automatic quoting, command substitution, AUTO Line numbering, Program tracing. A huge range of software, no cassette is absolutely indispensable.

Peripherals
Full ASCII keyboard with 10 key numeric selection, keyboard layout. Expansion port which converts a parallel LPT Port to printer.

Video Genie Net V.A.T. Total

369.50 55 43 425.00

Sharp

SHARP MZ-80K

2.80 based CPU

4K Byte monitor in ROM

16K memory capacity from 4 to 48K

14K Extended BASIC

16K in video display, 40 chars. of 24 lines

80 x 50 bit mapped graphics

8K RAM, 16K ROM and upper, lower case, graphics mode

Full 79 Key Keyboard

Extremely reliable monitor with 3 octaves

Fast reliable cassette unit with tape counter, 1200 b.p.s.

Wide variety of system software and expansion

50 pin bus connector for system expansion.



A complete personal computer system for the microcomputer user, at an economic price. The Sharp MZ-80K is a system with all necessary peripherals, sample software and essential documentation - giving the user maximum compatibility and flexibility and ease of use. At the heart of the machine is the 2.80 CPU, widely accepted as the best in its class. In CPU power it is second to none and in CPU speed it is the best. A 4K byte system monitor controls system operation. From 4 to 48K memory can be increased on board; enough room for the most demanding applications.

An extensive graphics character set, plus 32 colour sets, graphics and font cassette. The MZ-80K has a built-in monitor which complements these basic facilities. It has the widest use and compatibility of "black box" software and compatibility with all peripherals and facilities for expansion.

Sharp Basic occupies 16K of RAM, and offers extended features above those of standard instrumental microcomputers.

Model	Net	V.A.T.	Total
5200	78.00	59.80	137.80
1050	64.00	81.00	145.00
1220	64.00	81.00	145.00
22K	96.00	72.00	178.00
34K	111.00	85.00	196.00

Acorn



This compact stand-alone micro-computer is based on Ethernet technology and employs the highly popular 68020 MPU. Take a look at the full specifications, and see if the Acorn meets your requirements.

The Acorn consists of two simple components:

1. MPU card: 68020 microprocessor, 512 x 32 bit memory, 8MHz 32 bit 16-way cache, 5V 32Mbit RAM, 2K EPROM crystal, SV 960 CRT for 2K EPROM display.

2. Keyboard card: 25 click keys (16 bits), 8 control, 8 digit, 7 segment display CUTS control, and crystal controlled tape interface circuitry.

Acorn Operating Manual

With Acorn, you'll receive an operating manual that covers computing in full, from the basics to the more advanced topics, and how to programme with the 68020 instruction set. The manual also includes a listing of the software available and the hardware options, and other useful publications, plus sample programs.

Net V.A.T. Total

85.00 9.72 94.72

75.00 11.25 86.25

Acorn Monitor
A highly reliable glass-through hole plated PCB with solder resist and component identification, this award winning monitor has a resolution of 1280 x 960 pixels and a memory of 1M RAM (2732).

ACORN V.D.U.
The Acorn V.D.U. Board connects to the monitor and contains a 16K character memory mapped character storage RAM which is transparently written to or read from the monitor.

An MC 6845 programmable controller IC provides all the synchronisation signals to the V.D.U. 625 nsec 1024x768 pixel resolution V.D.U. which reads addresses for the character RAM. Characters are then fed via an SAA 5050 character generator IC to the monitor. The monitor also has the ability to create the characters to refresh the screen.

The V.D.U. 6800 produces Teletext standard characters and has Red, Green and Blue drive outputs, giving coloured characters or graphics.

V.D.U. Card (Kit) Net V.A.T. Total

880.00 13.20 101.20

**NEW
LOW PRICES**

Bigger and better than ever!

Commodore Pet



A complete Computer for the price of a good typewriter! With a library of over 200 programs including business, science, education and entertainment!

Pet can store and retrieve data which conveniently occupies large storage capacities. Programs can be run traditionally before and time sequencing.

Ease of Operation

The Commodore PET comes complete with a built-in TV screen, keyboard and power supply. All you need to do is plug it in and away goes any need for computer knowledge - all programs can readily be written in the BASIC computer language of PET which is easily learned.

An Expandable System

For example, the PET is a prime design concept enabling PET to be made the heart of a much larger system incorporating printer, floppy discs etc., as and when required.

Computers

PET 2001-B — PET with internal cassette and calculator size keyboard. External memory optional.

New VAT Total
550.00 82.50 632.50

PET 2001-MN — PET with 16K bytes memory and large keyboard. External cassette optional.

VAT Total
675.00 101.25 776.25

PET 2001-SN — PET with 32K bytes memory and large keyboard. External cassette optional.

New VAT Total
795.00 119.25 914.25

Compatibility Disk Units

400K Random Access Pet

400K Random Access Pet

New Pet 2

840.00 128.00 968.00

800K Random Pet

New Pet 2

995.00 149.25 1144.25

NAS Corner

	Net	VAT	Total
Monitor 3	125.00	18.75	143.75
Monitor 2	125.00	40.00	165.00
Power Supply	24.50	3.68	28.18
Mem. 8K	25.00	3.75	28.75
74	25.00	3.75	28.75
Mon.	2.90	0.44	3.34
Motherboard	10.00	1.50	11.50
Memory board	10.00	1.50	11.50
Buffer board	35.00	5.25	40.25
RAM Board (8K)	35.00	5.25	40.25
RAM Board (16K)	35.00	5.25	40.25
RAM Board (32K)	35.00	5.25	40.25
T-Board	25.00	3.75	28.75
TI BASIC ROM	25.00	3.75	28.75
8K BASIC ROM	40.00	6.00	46.00
8K BASIC ROM	15.00	2.25	17.25
Monitor 8K ROM	49.95	7.49	57.44
Relay Board (assembled)	60.00	9.00	69.00

Media

Microdigital are famous for their high quality, low cost magnetic media

	Net	VAT	Total
C15 cassette	1.00	0.15	1.15
Idea 10 in.	4.40	0.66	5.06
Library cassette	2.00	0.30	2.30

TI Programmer

A calculator designed for the computer user.

Allows accurate conversion between Octal, Decimal and Hex.

Calculations in all three number bases.

Handles mixed number bases and

combined logical and arithmetic operations.

Prints values automatically in user specified format.

Ex Stock

	Net	VAT	Total
TI Programmer	46.25	6.93	53.18

Hire

Avail an inexpensive mistake Microdigital have a range of microcomputers at low cost. Both long and short term leases, a wide range of machines and the ability to tailor make a system to your needs. We have the finest hire service available in the U.K. Contact Malcolm Reidly on 051-227 2535 for further details.

Please enclose:
Access Barcard No.
Number on
051-236 0707
or complete
this order
form



25 Brunswick Street, Liverpool L2 0PF
Tel: 051-236 0707 (24 Hour Mail Order)
051-227 2535/6/7/8 (All other Depts.)

Please Send Me:

I Enclose:
Cheque/Postal Order No. _____

Barclaycard No. _____ Access No. _____

Name _____

Address _____

Post Code _____

All Prices Include Carriage

LSD

MICRODIGITAL

25 Brunswick Street, Liverpool L2 0PF
Tel: 051-236 0707 (24 Hour Mail Order)
051-227 2535/6/7/8 (All other Depts.)
Mail orders via MICRODIGITAL LIMITED
FREEPOST (No Stamp Required)
Liverpool L2 2AB

The Mighty Microdigital

Microdigital are one of the larger and longest established Microcomputer firms in Europe.

We sell a wide range of systems, backed up by expert services that are second to none.

Our present retail outlet is at 25 Brunswick Street, Liverpool. Our well informed staff are happy to demonstrate equipment, provide technical help, or just chat.

Microdigital also offer an extensive mail order service. All orders are handled personally and we guarantee a prompt service.

Microdigital also provides a service for potential customers - the capabilities of a particular machine can be evaluated without a substantial capital investment.

Microdigital also try and provide the most competent service in the Microcomputer industry.

The Microcomputer shop providing a complete service from single-chip to a data processing console.

Opening hours: 9-5.30 Mondays to Saturday

Friendly, expert staff always on hand.

Our new, glossy, 16 page brochure is now

available to the industry - Send for your free copy today.



MICRODIGITAL
1980

Microdigital Software Announce

5 packages which are:

General Ledger	£295.00
Payroll System	£295.00
Sales Ledger	£295.00
Stock Control	£200.00
Printfit	£360.00

These packages are now available, demonstrations on request.

Ring 051-227 2535 and ask for Graham Jones (Software Manager).

These are fully tested systems which run on the Apple II+ 2020 with one or two disk drives.

The Stock Control package can handle up to 1250 stock items and uses two disk drives.

Liverpool Software Gazette

Britain's very first journal for Micro Software. Review, tutorials, news - PET, Apple columns, keep yourself informed with the latest trends in Microcomputing.



ONLY
50p
PER COPY

Please subscribe
the next
12 issues of
"Liverpool
Software Gazette"
Enclose cheque/
PO for £6.00

Access No. _____
Barclaycard No. _____
Name _____
Organisation _____
Address _____
Post Code _____

Mail to: Microdigital Ltd. FREEPOST
(No Stamp Required), Liverpool L2 2AB

```

50350 REM ABOVE LINE ADDS CHARACTERS TO SG# UNTIL NON-LETTER OR NON-DIGIT.
50380 REM NOW WE CAN DO A SEARCH FOR RESERVED WORDS, E.G. FOR, IF, THEN ETC.
50390 SRF=LEFT$(SG#+SP$) LRY:REM SP$=NL SPACES, PAD OUT SP$ WITH SPACES.
50400 FOR RH=1 TO N REM NUMBER OF RESERVED WORDS.
50410 IF (SR$=RUS$(RD)) THEN RH=H:NEXT RH:TK#=SR$:RETURN REM RESERVED WORD.
50420 NEXT RH:TK$="IDENT":RETURN REM IF NOT RESERVED WORD ASSUME IDENTIFIER.
50430 NM$="":REM NM WILL HOLD STRING REPRESENTATION OF NUMBER INPUT.
50440 IF (CH$<"0">OR(CH$)>"9") THEN 50430 REM SINCE NOT A NUMBER.
50450 TK$="NUM".
50460 NM$=NM$+CH$:GOSUB 506000 IF (CH$)="0"AND(CH$C=="9") THEN 50460
50470 NR=val(LEFT$(NM$,10)):RETURN REM WITH TK$="NUM" AND NR=VALUE OF NUMBER.
50480 REM NOT AN IDENTIFIER, RESERVED WORD OR NUMBER, TRY FOR SPECIAL SYMBOL.
50490 IF (CH$<>" ") THEN 50520 REM IN PASCAL MAY BE PART OF :=.
50500 GOSUB 50000:IF (CH$=="") THEN TK$==":":GOSUB 50000:RETURN
50510 TK$="":RETURN
50520 IF (CH$<>"<") THEN 50560 REM < MAY BE PART OF <> OR <=
50530 GOSUB 50000:IF (CH$=>) THEN TK$=">":GOSUB 50000:RETURN
50540 IF (CH$=="") THEN TK$="<":GOSUB 50000:RETURN
50550 TK$="<" RETURN
50560 IF (CH$<>">") THEN 50590 REM > MAY BE PART OF >=.
50570 GOSUB 50000:IF (CH$=="") THEN TK$=">":GOSUB 50000:RETURN
50580 TK$=">":RETURN
50590 IF (CH$<>"/") THEN 50640 REM ' INTRODUCES A STRING.
50600 TK$="STR":S$="":REM START OF A STRING.
50610 GOSUB 50000:IF (CH$=="") THEN GOSUB 50000:RETURN
50620 SG#=SG$+CH$:IF (LEN(SG$)<255) THEN 50610 REM TO GET NEXT CHARACTER.
50630 TK$=CHR$11:RETURN REM ERROR 1.
50640 IF (CH$<>"!") THEN 50670 REM ! INTRODUCES A COMMENT.
50650 GOSUB 50000:IF (CH$C=="!") THEN 50650:REM SEARCH FOR END OF COMMENT.
50660 GOSUB 50000:GOTO 50300:REM GET TOKEN AGAIN.
50670 IF (CH$<>"") THEN TK$=CH$:GOSUB 50000:RETURN REM SOME CHARACTER IS TOKEN
50680 TK$="NUM":GOSUB 50000:NR=ASC(CH$):IF (HX>=48)AND(HX<=57) THEN 50720
50690 FOR HD=1 TO 4:HX=ASC(CH$):IF (HX>=48)AND(HX<=57) THEN 50720
50700 IF NOT((HN)=65)AND(HN<=70) THEN RH=HD:HD=4:NEXT HD:GOTO 50730
50710 HX=HX-7
50720 HX=X-48:NP=NR*16+HX:GOSUB 50000:NEXT HD:RETURN
50730 IF (RH>1) THEN TK$=CHR$(2):RETURN REM ERROR NUMBER 2.
50740 TK$=="":RETURN REM REM ON ITS OWN.

READY.
```

A routine to determine the number of bytes in a 6502 instruction

This routine, taking only 32 bytes, when passed the first byte of a 6502 instruction in the accumulator, returns the number of bytes in the entire instruction

(either 1, 2 or 3) in the X register.

It is completely relocatable, and could form a useful routine to any program which needs to know the length of a 6502 instruction, e.g. a single step program, a relocator program or a disassembler.

```
NBYTES:    A2 01      LDX #\$01      ;Assume 1 byte long.
           C9 20      CMP #$20      ;JSR ($20) is abnormal.
           F0 17      BEQ THREE     ;JSR is 3 bytes long.
           29 9F      AND #$9F
           F0 15      BEQ ONE       ;0XX00000 (X=don't care) is 1 byte long.
           29 1D      AND #$1D
           C9 19      CMP #$19
           F0 0D      BEQ THREE     ;XXX110X1 is 3 bytes long.
           29 0D      AND #$0D
           C9 08      CMP #$08
           F0 09      BEQ ONE       ;XXXXX0X0 is 1 byte long.
           29 0C      AND #$0C
           C9 0C      CMP #$0C
           F0 01      BEQ THREE     ;XXXX11XX is 3 bytes long.
TWO:        CA         DEX
THREE:      E8         INX
ONE:        E8         INX
           60         RTS          ;Return from subroutine with correct
                                ;value for number of bytes in X register.
```

This routine was borrowed from the documentation for the ACORN colour display unit.

Disabling the STOP key on a PET.

Corrections to last issue's PETs Corner

Page 13, column 1, line 6 '=' should be '<>' (not equal to).

To disable the action of the STOP key on a PET, the following POKE command should be inserted in the BASIC program:

Page 13, column 1, line 10 should be (not equal to).

for old ROMs POKE 537,136

Page 13, column 36 'N set to 0' should be 'N set to 0' (zero).

for new ROMs POKE 144,49.

Page 13, column 2, line 2 after 'ROMs' insert the following: 'This routine saves the registers of the 6502 on the stack and executes an indirect JMP instruction through locations \$0219, \$021A (old ROMs)'.

Page 13, column 2, line 4 before 'of RAM' insert 'respectively'.

Page 14, column 1, line 30 of the BASIC program, 'FOR I' should be 'FOR I'

These locations contain the low order byte of the interrupt routine address (discussed in last issue), and the values POKEd are simply the original values incremented by 3. Looking at the interrupt routine with a disassembler shows that the first task of the interrupt routine is to execute a JSR (jump to subroutine) instruction, which takes 3 bytes, so that the POKE results in this subroutine not being executed. The jiffy clock is also disabled by the POKE, but no other ill effects have been reported with this technique, although the original values should be POKEd back before attempting any tape input/output.

Page 14, column 1, line 40 of the BASIC program, 'POKE 1,M' should be 'POKE I,M'.

TAPE HANDLING HINTS

Page 14, column 1, line 70 of the BASIC program, '(cr) (cl) (cl) (cl)' should be '(cr) (cr) (cl) (cl) (cl)'.

Page 14, column 1, line 100 of the new ROMs alteration, 'IS' should be 'IF'.

After writing the 'editorial' at the start of this section the following article was received from R. Cason, 15 Parkway, Sawbridgeworth, Herts., concerning the handling of tapes on the PET.

Page 14, column 2, line 5, 'SE1' should be 'SEI'.

Page 15, column 1, line 6, 'TOTAL: 0' should be 'TOTAL: = 0 (zero)'.

Page 15, column 2, STOP PRESS, line 1, '(pin 22)' should be '(pin 27)', and in steps 4 and 5 the 'characters around X(return) etc should not be entered.'

The layouts on page 17 depict the arrangement and assignment of the registers in the 2 PIAs and 1 VIA on the PET main logic board.

One of the most annoying things about the loading of programs and data from the PET's cassette is waiting for the FOUND ... message. Even if a tape counter were provided the program header could still be missed, and

hence the entire file missed by the operating system.

Mr Cason suggests connecting a SOUNDBOX (normally used for music making) to the USER PORT pin 6 (the first cassette read line) instead of the original connection to pin M (the CB2 output from the shift register on the VIA). On both SAVE and LOAD the following different sections of the tape should be recognisable from the sound they produce:

- (a) the HEADER TONE
- (b) the HEADER TOKEN
- (c) the HEADER 'TITLE'
- (d) the PROGRAM (or DATA)
- (e) the HALF WAY POINT
- (f) the SECOND COPY OF THE PROGRAM
- (g) the END OF FILE TOKEN

With practice the HEADER of a particular file on a multi-file tape can be found using fast forward, play and rewind. After rewinding slightly a normal LOAD can be performed.

In addition the following should also be discernable:

- (h) DROPOUTS
- (i) CROSSTALK
- (j) NOISE
- (k) VARIATION IN PITCH DUE TO TIGHT CASSETTES

(I) PROGRAM/DATA DIFFERENCES

To use the SOUNDBOX normally Mr Cason suggests adding a 3 position switch to the SOUNDBOX, position 1 being to pin M (for music), position 2 not connected (for OFF) and position 3 being to pin 6 (for the cassette). Pin N is the common connection (earth).

Mr Cason also uses an Hitachi TRQ 299 with Automatic Level Control and a Cue and Review facility, which allows fast winding using Cue to find the n'th file on a tape, then Review to position the tape before the HEADER. The tape can then be transferred to the PET cassette for LOADING.

Mr Cason's letter ends by asking if anyone can suggest a method of recovering data from a tape where the HEADER and part of the first copy of the DATA has been erased.

Having tried Mr Cason's suggestion I can report that it does work, although I only tried it on LOAD. It does seem practical to distinguish the various sections of the file, but I think I would prefer the extra expense of buying 12 or 15 minute tapes and only recording one program (possibly with multiple copies on each side) per tape. If this is not possible then Mr Casons's suggestion seem to offer a reasonably cheap and reliable method of storing more than one file per tape.

Nybbles

Nybbles 1

EVER tried neatly formatting headings etc., on print statements? Counting spaces for centring etc. Try
10AS "THIS IS A SAMPLE STRING": REM
Title etc.

20 PRINT TAB ((40-LEN(A\$))/2);A\$: REM
PRINT STRING CENTRED.

Nybbles 2

A MESSAGE from our sponsor??
Try this on your PET with new ROMS
WAIT 6502, 100

Nybbles 3

DID you know that before a 6502 does a memory write, it reads the addressed location? Any devices on the bus

structure waiting for a memory access will receive two in short succession. This can lead to some very puzzling behaviour with custom interfaces!

Nybbles 4

EVER fancied the odd 16 bit register on your 6502? Steve Wozniak, designer of the Apple, implemented a 16 bit motor processor 6502 enhancement package known as 'SWEET-16-The 6502 Dream Machine', as a sub routine in the Apple Integer Basic ROM set.

A full source listing and documentation was given in the November 1977 Byte. Sweet 16 has 15 16-bit registers, extensive pointer, stack manipulation capability but a very simple to use and understand instruction set.

Saves a lot of code, particularly in programs which require extensive 16-bit capability such as text editors, assemblers etc. Disadvantages are the reduced execution speed, plus you must have the interpreter (\$189 Bytes) resident.



Chess for the Acorn

by LAURENCE HARDWICK



THE listing given is the hex dump of a chess program to run on an Acorn System 1. The second RAM/I/O chip, IC 8, must be fitted as the program uses most of the maximum RAM in the system.

The program plays a fairly mechanical game but will make the player think, and teach him not to make too many mistakes. It is possible to set three levels of play ranging in average response time from 3 to 100 seconds. The program can also be set to play any one of 5 standard openings with the possibility of adding others if required.

LOADING THE PROGRAM

The program consists of four sections of machine code and one section of tables, loaded into the following locations.

0100 — 01B0 Section 1 Acorn RAM Page 1
 0200 — 03FF Section 2 Acorn RAM Pages 2, 3
 0980 — 09EF Section 3 RAM/I/O IC 2
 0E80 — 0EEF Section 4 RAM/I/O IC 8
 0030 — 00DF Data Block Tables Page 0

Take care as you load each section as an error at any point could destroy the program when it is run. It is preferable to save each section to tape after loading and checking.

When all sections are loaded, execution should be started at location 09A5.

PLAYING CHESS

The program represents the board and pieces as shown in Figure 1, each square on the board is described by its co-ordinates relative to the computer's right hand square at location 0,0.

The program accepts the following commands:—

C Reset Board

E Exchange computer's and player's pieces

D Calculate computer's move

F Enter player's move

Each of these will now be described:—

C — When the C key is pressed the display will show XX CC CC, re-setting the board copies a table of initial positions, into a current position table. The initial position table starts at 70 and takes the following form:—

Location:—

Computer's pieces:—

Player's pieces:—

7e	q	fr	qr	fb	qb	fn	qn	
rP	rP	nP	nP	bP	bP	PP	FP	
7g								
7f								
8e	q	fr	qr	fb	qb	fn	qn	
rP	rP	nP	nP	bP	bP	PP	FP	
8g								
8f								
7g								
7f								
8g								
8f								

It is possible to set up any initial position of pieces from which to start, a captured piece is denoted in the table by CC.

After re-setting the board the program expects to play white and have first move.

E — When the E key is pressed the display will show

XX EE EE. Exchanging the board turns the board round so that the computer is still playing towards you from the 0,0 corner but now with black instead of white, or vice-versa. It is possible to exchange the board at any time and any number of times during the game.

D — The D key causes the computer to calculate its move, the display will flash for a while, showing the moves under analysis, and then show the move chosen in the following form:—

Fb	D3	D0
↑ PIECE	↑ FROM	↑ TO

The computer can be asked to calculate a move at any stage during the game, even after it has just moved.

F — The F key allows the player to make his move, the 'from' and 'to' co-ordinates are entered in order and digits move into the field from right to left and are terminated by any monitor command key.

The piece field will only display correctly when the 'from' co-ordinates are occupied.

Having set up the required co-ordinates, the F key

should then be pressed, care should be taken to make legal moves as no checks are made; it is possible for a player to take his own pieces even with an empty space! Any number of moves may be made to allow special positions to be set up and to allow castling.

EXAMPLE GAME

A typical game will start in the following way:—

<u>Key Press</u>	<u>Display</u>	<u>Comment</u>
G 09AS G	XX XX XX	Start program
C	XX EE EE	Clear board
D	FP 13 33	P - K4 Computer's move
63 43	FP 63 43	Set up P - K4 move
F	XX 63 43	Enter move
D	Fn01 22	K - KB3 computer's move

OPENINGS

The opening in the hex. dump is designed for the computer to play white in the Giuoco Piano defence.

The openings take the form of a table in locations C0—DB. Consecutive bytes downwards from DB inclusive are:—

a) The co-ordinates moved to in the player's last move, or $\square \square$ in the case of a clear board command.

b) The address of the piece to be moved by the computer relative to the bottom of the current position table at 50.

c) The co-ordinates that this piece is to be moved to.

This pattern of three bytes describing player's move and programmed response is repeated for the first nine moves and is terminated by a 99 at location C0. Table 2 is a list of programmed openings for black or white for five of the most popular openings. If there is any deviation from the set opening the computer will continue to calculate moves as usual.

WINNING AND LOSING

If the computer wishes to resign or is in checkmate it will display XX FF FF. The program gives no indication of when the player is in check or checkmate and will take his King in that case.

DEPTH OF PLAY

The depth of look ahead and the factors considered each time the computer calculates a move may be set to one of three levels:—

Average response time 3 seconds
 Location 02F2 should contain 00
 Location 018B should contain FF

Average response time 10 seconds
 Location 02F2 should contain 00
 Location 018B should contain FB

Average response time 100 seconds
 Location 02F2 should contain 08
 Location 018B should contain FB

Figure 1. Board and Piece Representation

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7		
1,0	Rf	Rn	Rb	R	Q	Qb	Qn	Qr	COMPUTER
2,0	$r.P$	$n.P$	$b.P$	$f.P$	$q.P$	$b.P$	$n.P$	$r.P$	
3,0									
4,0									
5,0									
6,0	$r.P$	$n.P$	$b.P$	$f.P$	$q.P$	$b.P$	$n.P$	$r.P$	PLAYER
7,0	Rf	Rn	Rb	R	Q	Qb	Qn	Qr	
									7,7

The player's pieces are always identified by a decimal point following them.

ACORN CHESS Program block 1

TABLE 1

```

0100: A6 B5 30 5C A5 B0 F0 08 E0 08 D0 04 C5 E6 F0 2E
0110: F6 E3 C9 01 D0 02 F6 E3 50 1E A0 0F A5 B1 D9 60
0120: 00 F0 03 88 10 F8 B9 A0 00 D5 E4 90 04 94 E6 95
0130: E4 18 08 75 E5 95 E5 28 E0 04 F0 03 30 31 60 A5
0140: E8 85 DD A9 00 85 B5 20 4B 03 20 B2 02 20 00 02
0150: 20 B2 02 A9 08 85 B5 20 09 02 20 31 03 4C 80 0E
0160: E0 F9 D0 0B A5 60 C5 B1 D0 04 A9 00 85 B4 60 50
0170: FD A0 07 A5 B1 D9 60 00 F0 05 88 F0 F1 10 F6 B9
0180: A0 00 D5 E2 90 02 95 E2 C6 B5 A9 FB C5 B5 F0 03
0190: 20 25 03 E6 B5 60 C9 08 B0 12 20 EA 03 A2 1F B5
01A0: 50 C5 FA F0 03 CA 10 F7 86 FB 86 B0 4C A5 09 73

```

ACORN CHESS Program block 2

```
0200: A2 10 A9 00 95 DE CA 10 FB A9 10 85 B0 C6 B0 10
```

0210: 01 60 20 1E 03 A4 B0 A2 08 86 B6 C0 08 10 41 C0
 0220: 06 10 2E C0 04 10 1F C0 01 F0 09 10 0E 20 8E 02
 0230: D0 FB F0 D9 20 9C 02 D0 FB F0 D2 A2 04 86 B6 20
 0240: 9C 02 D0 FB F0 C7 20 9C 02 A5 B6 C9 04 D0 F7 F0
 0250: BC A2 10 86 B6 20 8E 02 A5 B6 C9 08 D0 F7 F0 AD
 0260: A2 06 86 B6 20 CA 02 50 05 30 03 20 00 01 20 1E
 0270: 03 C6 B6 A5 B6 C9 05 F0 EB 20 CA 02 70 8F 30 8D
 0280: 20 00 01 A5 B1 29 F0 C9 20 F0 EE 4C 0D 02 20 CA
 0290: 02 30 03 20 00 01 20 1E 03 C6 B6 60 20 CA 02 90
 02A0: 02 50 F9 30 07 08 20 00 01 28 50 F0 20 1E 03 C6
 02B0: B6 60 A2 0F 38 B4 60 A9 77 F5 50 95 60 94 50 38
 02C0: A9 77 F5 50 95 50 CA 10 EB 60 A5 B1 A6 B6 18 75
 02D0: 8F 85 B1 29 88 D0 42 A5 B1 A2 20 CA 30 0E D5 50
 02E0: D0 F9 E0 10 30 33 A9 7F 69 01 70 01 B8 A5 B5 30
 02F0: 24 C9 00 10 20 48 08 A9 F9 85 B5 85 B4 20 4B 03
 0300: 20 B2 02 20 09 02 20 2E 03 28 68 85 B5 A5 B4 30
 0310: 04 38 A9 FF 60 18 A9 00 60 A9 FF 18 B8 60 A6 B0
 0320: B5 50 85 B1 60 20 4B 03 20 B2 02 20 09 02 20 B2
 0330: 02 BA 86 B3 A6 B2 9A 68 85 B6 68 85 B0 AA 68 95
 0340: 50 68 AA 68 85 B1 95 50 4C 70 03 BA 86 B3 A6 B2
 0350: 9A A5 B1 48 A8 A2 1F D5 50 F0 03 CA 10 F9 A9 CC
 0360: 95 50 8A 48 A6 B0 B5 50 94 50 48 8A 48 A5 B6 48
 0370: BA 86 B2 A6 B3 9A 60 A6 E4 E4 A0 D0 04 A9 00 F0
 0380: 0A A6 E3 D0 06 A6 EE D0 02 A9 FF A2 04 86 B5 C5
 0390: FA 90 0C F0 0A 85 FA A5 B0 85 FB A5 B1 85 F9 4C
 03A0: 80 09 A6 DC 10 17 A5 F9 D5 DC D0 0F CA B5 DC 85
 03B0: FB CA B5 DC 85 F9 CA 86 DC D0 1A 85 DC A2 0C 86
 03C0: B5 86 FA A2 14 20 02 02 A2 04 86 B5 20 00 02 A6
 03D0: FA E0 0F 90 12 A6 FB B5 50 85 FA 86 B0 A5 F9 85
 03E0: B1 20 4B 03 4C A5 09 A9 FF 60 A2 04 06 F9 26 FA
 03F0: CA D0 F9 05 F9 85 F9 85 B1 60 05 F9 85 F9 85 B1

ACORN CHESS program block 3

0980: A5 F9 20 60 FE A5 FA A0 03 20 6F FE A5 FB C9 10
 0990: 29 EF AA A9 00 85 15 6A 85 12 B5 30 85 10 B5 3F
 09A0: 85 11 4C 0C FE D8 A2 FF 9A A2 C8 86 B2 A9 1F 85
 09B0: 0E 20 80 09 B0 FB C9 0C D0 0F A2 1F B5 70 95 50
 09C0: CA 10 F9 86 DC A9 CC D0 12 C9 0E D0 07 20 B2 02
 09D0: A9 EE D0 07 C9 0D D0 0B 20 A2 03 85 FB 85 FA 85
 09E0: F9 D0 C6 90 06 20 4B 03 4C 9D 01 4C 96 01 03 4C

ACORN CHESS program block 4

0E80: 18 A9 80 65 EB 65 EC 65 ED 65 E1 65 DF 38 E5 F0
 0E90: E5 F1 E5 E2 E5 E0 E5 DE E5 EF E5 E3 B0 02 A9 00
 0EA0: 4A 18 69 40 65 EC E5 ED 38 E5 E4 4A 18 69 90 65
 0EB0: DD 65 DD 65 E1 38 E5 E5 E5 E4 E5 E4 E5 E5
 0EC0: E5 E5 E5 E2 A6 B1 E0 33 F0 16 E0 34 F0 12 E0 22
 0ED0: F0 OE E0 25 F0 OA A6 B0 FO 09 B4 50 CO 10 10 03
 0EE0: 18 69 02 4C 77 03 03 18 69 02 4C 77 03 79 99 83

ACORN CHESS data block

0030:	00	00	75	67	75	67	75	67	D0	50	D4	54	FC	7C	67	75
0040:	67	50	50	7C	7C	54	54	73	73	73	73	73	73	73	73	CC
0050:	04	36	07	00	05	02	06	01	17	10	16	11	15	12	13	34
0060:	74	55	77	70	75	72	76	71	67	60	66	61	65	62	63	44
0070:	03	04	00	07	02	05	01	06	10	17	11	16	12	15	14	13
0080:	73	74	70	77	72	75	71	76	60	67	61	66	62	65	64	63
0090:	F0	FF	01	10	11	0F	EF	F1	DF	E1	EE	F2	12	0E	1F	21
00A0:	0B	0A	06	06	04	04	04	04	02	02	02	02	02	02	02	02
00B0:	01	36	C8	FB	F9	04	00	AD	4B	6C	B2	CF	AA	78	AA	3C
00C0:	99	25	0B	25	01	00	33	25	07	36	34	0D	34	34	0E	52
00D0:	25	0D	45	35	04	55	22	06	43	33	0F	CC	EE	00	00	00

TABLE 2

PROGRAMMED OPENINGS

ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC

ETCETERA

ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC

LISP for the Apple II -

OWL Computers announces its new software product for the Apple II computer—an interpreter for the language LISP. Apple LISP is intended for:

- university and school students who are learning LISP

or carrying out research up to M.Sc level.

- hobbyists who want hands on experience of the fundamental language of artificial intelligence research.
 - system designers who require more flexibility in data

LISP FUNCTIONS

ADD1	IN	RECLAIM
AND	LESSP	REMAINDER
APPLY	LIST	REMPROP
ASSOC	LISTP	RPAR
ATOM	LOAD	RPLACA
BLANK	LOOP	RPLACD
CALL	LPAR	SAVE
CAR	MAP	SET
CDR	MAPC	SETQ
CAAR, CADR etc.	MESSOFF	SUB1
CAAAR, CAADR etc.	MESSON	SUPBRP
CHARACTER	MINUS	SUPERPRINT
CHARP	NIL	SUPERVISOR
CHARS	NOT	T
COND	NULL	TIMES
CONS	NUMBERP	UNDEFINEDVALUE
CR	OBLIST	UNTIL
DE	ONEP	WHILE
DIFFERENCE	OR	ZEROP
DOLLAR	ORDINAL	
DOS	PAGE	
EDIT	PAGELINES	
EQ	PEEK	
ERROR	PERIOD	
ERRORCOUNT	PLIST	
ERRORSET	PLUS	
EVAL	POKE	
EXPLODE	PR	
F	PRINO	
FSUBRP	PRINT	
GET	PROGN	
GETCHAR	PUT	
GREATERP	QUOTE	
IMPLODE	QUOTIENT	
LAMBDA	READ	

and control structures than is provided by traditional programming languages.

The system consists of 6k bytes of machine code interpreter plus 4k bytes of initialised LISP workspace containing LISP utilities and constants. It is supplied on disc or cassette tape with a 44 page manual for a 16k or larger computer. Two demonstration programs are included.

The system has been designed with ease of use in mind. Important features include;

- fully interactive with explicit EVALUATE and VALUE IS messages
- automatic parenthesis count to help in typing complex expressions on the computer
- built in SUPERPRINTER to format the printing of large expressions
- editing by screen editing or built in LISP editor
- all errors trapped and optional full traceback printed
- Owl Computers' LISP contains a number of extensions to basic LISP, including;
- character string processing
- PEEK, POKE, and CALL to control Apple hardware and machine code programs
- functions can have optional arguments with default values
- improved interactive control structures using LOOP, WHILE and UNTIL functions
- Apple DOS and input output control functions

The fast compacting garbage collector automatically finds space for numbers, lists or character strings if there is any space at all remaining. This means that the programmer need never be concerned about the details of storage allocation.

LISP is available from Owl Computers for £30 plus £4.50 VAT.

SUPER BASIC

Motorola are developing a 'Super Basic' called BASIC⁶⁹ for the 6809. While compatible with existing interpreters it is intended to implement a large number of extensions, (largely inspired by Pascal with the

IF... THEN... ELSE

WHILE... DO

REPEAT... UNTIL

Procedures and user defined data types).

What makes it particularly interesting is that Motorola plan to burn it into a large ROM mass marketed at a low cost. Using the 6809 relative instructions it is to be position-independent-code with system independent calls for I/O.

With the announcement of 4K x 8 and 8K x 8 pseudo-static RAMS the £100.00 16 Chip PET does not seem far away!

BASIC⁶⁹

Incremental Compiler—fast, syntax errors detected on entry.

Structured Procedure orientated—Named procedures, arbitrary length identifiers, local variables.

CALL Procedure with Parameters

TRACE

User defined data structures

Re-entrant

9 Decimal digits binary representation

Transcendentals accurate to 8 digits

'Print using'

Full set of normal Basic functions.

Basic⁶⁹ is, say Motorola, a system development language (SDL) close to the power of Pascal.

VISICALC

This software package is rapidly making a name for itself. Available on the Apple and Hewlett Packards' new Capricorn micro, it is a 'visual pocket calculator' computer package which while requiring hardly any programming knowledge. Allows the user to handle a wide variety of mathematical problems simply and easily.'

Everyone I know who uses it seems to be totally addicted! Written by Personal Software, of Microchess fame, it costs \$99 (or £99.00 if you buy it in Britain!).

FORTH LANGUAGE FOR THE PET

ACT Petsoft have introduced a modestly priced interactive FORTH Compiler/Interpreter to run on the best-selling PET microcomputer.

FORTH is a unique high level language, which has recently become popular in the United States, where it is widely used in astronomical and general scientific applications.

The principal advantages of the PET FORTH are that it is extremely fast and required only a small amount of memory. The language lends itself naturally to structured programming.

Petsoft's PET FORTH package contains a dictionary of 200 'words', each of which approximates to a subroutine in BASIC. Being vocabulary based the user can tailor the system to resemble the needs and structures of any specific application. Also included are a built-in incremental assembler and text editor.

PET-FORTH is available at £30 plus V.A.T. from ACT Petsoft, Radcliffe House, 66-68 Hagley Road, Edgbaston, Birmingham, B16 8PF.

SURROUND

DAVE STRAKER

SURROUND is a fast moving 'arcade' game for the Apple II or ITT 2020 written in APPLESOFT BASIC

```
10 REM <><><><> SURROUND <><><>
20 REM
30 REM <>THIS VERSION BY DAVE STRAKER<>
40 REM
50 GOSUB 500: REM INITIAL INFO PAGE
60 GOSUB 100: REM GAME INITIALISATION
70 GOSUB 200: REM RUN GAME
80 GOSUB 800: REM FINALISE
85 IF LEFT$(C$,1) = "Y" THEN 60
90 TEXT : HOME : END
100 GR
102 IF D > 0 THEN 115
104 D = 2
110 CR = 2:CB = 9:CF = 14:CE = 0
115 C = GB
120 COLOR= CE: REM BORDER
125 PRINT : PRINT : PRINT S$: PRINT
130 HLIN 0,39 AT 0: HLIN 0,39 AT 39: VLIN 0,39 AT 0: VLIN 0,39
140 COLOR= CF
150 FOR I = 1 TO 38: VLIN 1,38 AT I: NEXT I
170 XA = 17:YA = 17:XB = 22:YB = 22
180 AX = 1:AY = 0:BX = -1:BY = 0
182 IF D = 1 THEN RETURN
184 GOSUB 1000
186 GOTO 120
190 RETURN
200 REM MAIN LOOP
```

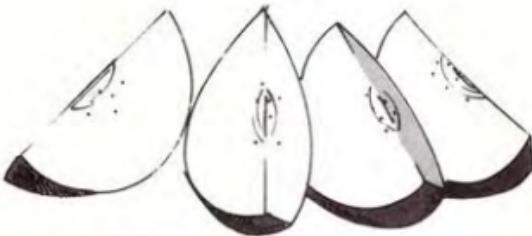


```
535 PRINT S1$: PRINT S2$: PRINT S$: PRINT S2$: PRINT S1$  
540 PRINT : PRINT " <THIS VERSION BY DAVE STRAKER>  
550 PRINT : PRINT "THIS IS A TWO PLAYER, ARCADE-TYPE GAME."  
560 PRINT "EACH PLAYER CONTROLS AN EXTENDING."  
570 PRINT "ORTHOGONALLY MOVING LINE WITH A PADDLE."  
580 PRINT : PRINT " THE WINNER IS THE ONE WHO REMAINS FOR"  
590 PRINT "THE LONGEST WITHOUT HITTING ANYTHING."  
600 PRINT "THE SPEED WILL GRADUALLY GET FASTER"  
610 PRINT "AND FASTER."  
620 PRINT : PRINT "WHAT ARE YOUR NAMES?"  
630 PRINT : INPUT "1ST PLAYER (PADDLE 0) ?"; R$  
640 PRINT : INPUT "2ND PLAYER (PADDLE 1) ?"; B$  
700 PRINT : INPUT "DIFFICULTY LEVEL(1-5) ?"; Z$  
710 Z = VAL(Z$)  
720 IF Z < 1 OR Z > 5 THEN 700  
730 GD = 5  
740 GB = 1055 - Z * 150  
750 GF = GB - 300  
760 RETURN  
800 IF CF < > SA AND CF < > SB THEN 850  
810 IF SA < > CF THEN 830  
820 CW = CA:W$ = R$: GOTO 840  
830 CW = CB:W$ = B$:  
840 W$ = "CONGRATULATIONS," + W$ + ", YOU WIN."  
845 GOTO 860  
850 W$ = "... HOW ABOUT THAT! IT'S A DRAW!"  
855 CW = CE  
860 PRINT : PRINT S$  
870 PRINT : PRINT W$  
880 PRINT " . . . . . ANOTHER GAME? ";  
890 Z = PEEK (- 16384): POKE - 16368, 0  
900 IF Z > 127 THEN 960
```

```
985 FOR I = 1 TO 50: NEXT I
990 COLOR= CW
995 HLIN 0,39 AT 0: HLIN 0,39 AT 39: VLIN 0,39 AT 0: VLIN 0,39
995 FOR I = 1 TO 100: NEXT I
995 COLOR= CE
995 HLIN 0,39 AT 0: HLIN 0,39 AT 39: VLIN 0,39 AT 0: VLIN 0,39
995 GOTO 890
960 Z = Z - 128
970 Z$ = CHR$(Z)
975 D = 2
980 RETURN
1000 REM CHOICE OF COLOURS
1010 COLOR= CR: PLOT XR,YR
1020 COLOR= CB: PLOT XB,YB
1025 D = 1
1030 PRINT : PRINT "THESE COLOURS OK? ";: GET Z$
1040 IF LEFT$(Z$,1) < > "N" THEN RETURN
1045 HOME
1050 GR
1055 D = 2
1060 FOR I = 0 TO 15
1070 COLOR= I
1080 VLIN 0,39 AT I * 2: VLIN 0,39 AT I * 2 + 1
1090 NEXT I
1100 PRINT "0 1 2 3 4 5 6 7 8 9 1 2 3 4 5"
1110 PRINT "0 1 2 3 4 5"
1115 POKE 37, PEEK(37) - 1: POKE 36, 0
1120 PRINT "BACKGROUND COLOUR": INPUT CF
1125 POKE 37, PEEK(37) - 1
1130 POKE 36, 0: CALL - 868: PRINT A$: "/S COLOUR": INPUT CR
1140 POKE 36, 0: CALL - 868: PRINT B$: "/S COLOUR": INPUT CB
1150 RETURN
```

Apple Pips

C.Phillips



Apple II Reference Manual

APPLE have done it again! Available shortly from your local dealer the new reference manual replacing the much loved (?) big red book of January 1978. The new manual is printed in the new standard 5 x 7 spiral bound format of the 'Applesoft', Dos 3.2 manuals and must surely be the last word in Apple documentation. It is divided into six sections:—

- Approaching your Apple*
- Conversation with Apples*
- The System Monitor*
- Memory Organisation*
- Input/Output Structure*
- Hardware Configuration*

3 Appendices deal with:

- The 6502 Instruction Set*
- Special Locations*
- ROM Listings—The Autostart and 'old' F8 ROMS.

Finally a glossary, bibliography and five separate indexes round off the coverage. Highlights include complete schematics of the main board, switching P.S.U. and keyboard, modifications for 'EURAPPLES', the 'mysterious' USER 1 jumper, complete ROM listings of both monitors, a detailed section on the system monitor with useful calls, operation of the different video modes, and using the Apple input/output structure with your own peripherals. Much of the information appears in print for the first time, and leaves little unsaid about the Apple. Eat your hearts out Commodore!

DISK FREE SPACE DOS 3.2 ONLY

This short sub-routine returns the number of free sectors on a disk in F1. As written the machine language code

resides at \$300—\$322 (768—802 Dec) and is poked into memory by lines 10—50. Once resident the routine can be executed repeatedly by CALL 768.

The number of free sectors are in locations 6 and 7.

```

5 REMDISK FREE SPACE 48K DOS 3.2 SYSTEM
10 FOR I = 768 TO 801 : READ B : POKE I,B : NEXT
20 DATA 32, 247, 175, 169, 0, 133, 6, 133, 7, 160, 56
30 DATA 185, 187, 179, 162, 7, 10, 144, 6, 230, 6, 208 2
40 DATA 230, 7, 202, 16, 244, 200, 192, 196, 144
50 DATA 96
60 CALL 768
70 FI = PEEK(6) + PEEK(7)*256
80 PRINT FI; "SECTORS FREE"
90 END

```

For use at assembly level the routine can be executed by a JSR \$300, returning the number of free sectors in locs 6 (low), 7 (high).

For users with less than 48K RAM the two absolute address references have to be altered accordingly.
For 32K systems lines 20 and 30 should read

```

20 DATA 32, 247, 111, 169, 0, 133, 6, 133, 7, 160, 56
30 DATA 185, 187, 115, 162, 7, 10, 144, 6, 230, 6, 208, 2

```

For 16K Systems

```

20 DATA 32, 247, 47, 169, 0, 133, 6, 133, 7, 160, 56
30 DATA 185, 187, 51, 162, 7, 10, 144, 6, 230, 6, 208, 2

```

0300-	20	F7	RF	JSR \$AFF7
0303-	A9	00		LDA \$E#00
0305-	85	06		STR \$06
0307-	85	07		STR \$07
0309-	A0	38		LDY \$E#38
030B-	B9	BB	B3	LDR \$B#3BB, Y
030E-	R2	07		LDX \$E#07
0310-	00			ASL
0311-	90	06		BCC \$E#319
0313-	E6	06		INC \$E#06

0315-	D0	02	DNE	\$#0319
0317-	E6	07	INC	\$#07
0319-	CR		DEX	
031A-	10	F4	BPL	\$#0310
031C-	C8		INV	
031D-	C8	C4	CPY	\$#FC4
031F-	90	ER	BCC	\$#030B
0321-	60		RTS	
0322-	3F		???	
0323-	R9	SD	LDA	\$#45D

20 POKE 768,169:POKE 769,50:POKE 770,162:POKE 771,0:POKE 772,138.
 21 POKE 773,24:POKE 774,233:POKE 775,1:POKE 776,208:POKE 777,252
 22 POKE 778,77:POKE 779,48:POKE 780,192:POKE 781,232:POKE 782,224
 23 POKE 783,3:POKE 784,208:POKE 785,242:POKE 786,136.
 24 POKE 787,208:POKE 788,237:POKE 789,96:POKE 790,200

This, when saved, creates a 'stand alone' program. Remember that the machine language routine is poked in at \$300—any utilities you may have loaded there will be clobbered!

Relocating the Apple Miniassembler

ONE of the nicest features of the Apple system monitor is the tiny 6502 assembler resident at F500-F668. Unfortunately APPLE-II Plus and 2020 owners no longer have it available—Applesoft needs that memory space. Using an Apple with Integer Basic on ROM type:

0C00<F500 .F63CM Return

0C337:0C
 0CSB:0C
 0CBF:0D
 0CDD:0D
 0CE7:0D
 0D33:0C

To run type 0C92G

To save on disk BSAVE Mini-assembler which can be loaded off disk or cassette when needed.

Programming Quickies

'VAL' Function for integer Basic

```

.DLIST
 5 DIM X$(10),X1$(10)
10 REM : "VAL" FUNCTION FOR INTEGER BASIC
20 CALL -936:VTAB 6
30 INPUT "STRING =? ",X1$:X#=X1$
40 GOSUB 899
50 PRINT "STRING :";X1$;" VALUE IS :";Z
55 PRINT " ";PRINT " "
60 GOTO 30
700 REM
710 REM
890 REM *** SUBROUTINE ***
899 H=0:Z=H:REM SET VALUE & H TO ZERO
900 FOR K=1 TO LEN(X$)
905 REM :LOOP TO GET FIRST NUMERIC CHARACTERS ONLY
906 REM :ONLY LEADING NUMERIC ALLOWED
910 F=ASC(X$(K,K))-176
912 REM :FIND ASCII VALUE
920 IF F>-1 AND F<10 THEN 940
921 REM :BETWEEN 0 AND 9 ?
925 IF K=1 THEN F=99
926 REM : 1ST CHARACTER A LETTER ?
930 K=LEN(X$):GOTO 950
935 REM :EXIT LOOP
940 H=H+1
945 REM :INCREMENT H
950 NEXT K
954 IF F=99 THEN RETURN
955 X#=X$(1,H)
965 H=H-1
1000 REM : X# IS NOW ONLY NUMERIC STRING
1010 FOR K=1 TO LEN(X#)
1015 XX=ASC(X$(K,K))-176

```

```

1040 XOR=HCC+(10 ^ H)
1050 H=H-1:Z=Z+XN
1060 NEXT K
1065 RETURN
9998 REM
9999 REM ** ROUTINE BY G JONES
10000 REM **
10001 REM ** HELP YOURSELF
>
>
>RUN
STRING =? 12345
STRING 12345 VALUE IS 12345

STRING =? ABCDE
STRING ABCDE VALUE IS 0
-
-
STRING =? 123EF
STRING 123EF VALUE IS 123
-
-
STRING =? D4RST
STRING D4RST VALUE IS 0

STRING =? 2E4R
STRING 2E4R VALUE IS 2

```

Look!

```

:SYNTH: error
LIST
10000 REM LOOK ! PAUL FULLHOD
10010 REM PROGRAM PRINTS INTOR
10020 REM APPLESOFT POINTERS AND
10030 REM THEIR VALUES
10040 PRINT "HIMEM IS SET TO      "; PEEK(115) + C PEEK(116) * 25
   6)
10050 PRINT "STRINGS COME DOWN TO    "; PEEK(111) + C PEEK(112) * 25
   6)
10060 PRINT "NUMERICS GO UP TO      "; PEEK(109) + C PEEK(110) * 25
   6)
10070 PRINT "VARIABLES GO UP TO      "; PEEK(107) + C PEEK
   <108> * 256
10080 PRINT "LOMMEM IS SET TO      "; PEEK(105) + C PEEK(106) * 25
   6)
10090 PRINT "PROGRAM TOP IS SET TO   "; PEEK(175) + C PEEK(176) * 25
   6)
10100 PRINT "PROGRAM BOTTOM IS SET TO"; PEEK(103) + C PEEK(104) * 25
   6)

```

Novel Sort

```

JLDRG-SORT
LIST
10 REM SUPER-SORT ROY STRINGER 1980
15 REM LINEAR SORT
20 CHLL = 956
30 INPUT X
40 DIM R(100): DIM B(100): P = ((X + 2) * (4 / 5000))
50 FOR I = 1 TO X: H = RND(2) * 1000: R(I) = H: PRINT I,R: NEXT I
60 FOR I = 1 TO X: C = INT(R(I) * P): IF B(C) > 0 THEN 140
70 B(C) = I: NEXT
80 PRINT " "; "OLD", "NEW"
90 FOR I = 0 TO X - 1: IF B(I) = 0 THEN 110
100 D = D + 1: PRINT R(B(I)): B(D)=D
110 NEXT: END
120 C = C + 1
130 IF H(B(C)) = 0 THEN 70
140 IF R(B(C)) < R(I) THEN 120
150 F = C
160 F = F + 1: IF B(F) > 0 THEN 160
170 B(F) = B(F) - 1: F = F - 1: IF F < C THEN 170
180 GOTO 70

```

6502 Instructions Sorted by Op-Code			26 00	ROL	\$00, X
00	BRK		27	???	
01 00	ORA	<(\$00, X)>	28	SEC	
02	???		29 00 00	RND	\$0000, Y
03	???		2A	???	
04	???		2B	???	
05 00	ORA	\$00	2C	???	
06 00	ASL	\$00	2D 00 00	RND	\$0000, X
07	???		2E 00 00	ROL	\$0000, X
08	PHP		2F	???	
09 00	ORA	\$#000	10	RTI	
0A	ASL		11 00	EOR	<(\$00, X)>
0B	???		12	???	
0C	???		13	???	
0D 00 00	ORA	\$0000	14	???	
0E 00 00	ASL	\$0000	15 00	EOR	\$00
0F	???		16 00	LSR	\$00
10 00	BPL	\$0002	17	???	
11 00	ORA	<(\$00), Y	18	PHA	
12	???		19	LSR	
13	???		1A	???	
14	???		1B	???	
15 00	ORA	\$00, X	1C 00 00	JMP	\$0000
16 00	ASL	\$00, X	1D 00 00	EOR	\$0000
17	???		1E 00 00	LSR	\$0000
18	CLC		1F	???	
19 00 00	ORA	\$0000, Y	20 00	BYC	\$0002
1A	???		00	EOR	<(\$00), Y
1B	???		21	???	
1C	???		00	CLI	
1D 00 00	ORA	\$0000, X	22	???	
1E 00 00	ASL	\$0000, X	00	EOR	\$00, X
1F	???		23	???	
20 00 00	JSR	\$0000	00	CLI	
21 00	AND	<(\$00, X)>	24 00	???	
22	???		00	EOR	\$0000, Y
23	???		25 00	???	
24 00	BIT	\$00	00	EOR	\$0000, X
25 00	AND	\$00	26 00	LSR	\$0000, X
26 00	ROL	\$00	27	???	
27	???		28	???	
28	PLP		29 00	RTS	
29 00	AND	\$#00	00	ADC	<(\$00, X)>
2A	ROL		2A	???	
2B	???		2B	???	
2C 00 00	BIT	\$0000	2C 00	???	
2D 00 00	AND	\$0000	00	ADC	\$00
2E 00 00	ROL	\$0000	2D 00	ROR	\$00
2F	???		2E 00	???	
30 00	BMI	\$0002	2F 00	???	
31 00	AND	<(\$00), Y	00	PLR	
32	???		2A	???	
33	???		2B	???	
34	???		2C 00 00	JMP	<(\$0000)>
35 00	AND	\$00, X	00	ADC	\$0000

6E 00 00	ROR	\$0000	A6 00	LDX	\$00
6F	???		A7	???	
70 00	BVS	\$0002	A8	TRY	
71 00	RDC	(\$00), Y	A9 00	LDA	£\$00
72	???		AA	TRX	
73	???		AB	???	
74	???		AC 00 00	LDY	\$0000
75 00	ADC	\$00, X	AD 00 00	LDA	\$0000
76 00	ROR	\$00, X	AE 00 00	LDX	\$0000
77	???		AF	???	
78	SEI		B0 00	BCS	\$0002
79 00 00	RDC	\$0000, Y	B1 00	LDA	(\$00), Y
7A	???		B2	???	
7B	???		B3	???	
7C	???		B4 00	LDY	\$00, X
7D 00 00	RDC	\$0000, X	B5 00	LDA	\$00, X
7E 00 00	ROR	\$0000, X	B6 00	LDX	\$00, Y
7F	???		B7	???	
80	???		B8	CLV	
81 00	STA	(\$00, X)	B9 00 00	LDA	\$0000, Y
82	???		BA	TSX	
83	???		BB	???	
84 00	STY	\$00	BC 00 00	LDY	\$0000, X
85 00	STA	\$00	BD 00 00	LDA	\$0000, X
86 00	STX	\$00	BE 00 00	LDX	\$0000, Y
87	???		BF	???	
88	DEY		C0 00	CPY	£\$00
89	???		C1 00	CMP	(\$00, X)
8A	TMA		C2	???	
8B	???		C3	???	
8C 00 00	STY	\$0000	C4 00	CPY	\$00
8D 00 00	STA	\$0000	C5 00	CMP	\$00
8E 00 00	STX	\$0000	C6 00	DEC	\$00
8F	???		C7	???	
90 00	BCC	\$0002	C8	INY	
91 00	STA	(\$00), Y	C9 00	CMP	£\$00
92	???		CA	DEX	
93	???		CB	???	
94 00	STY	\$00, X	CC 00 00	CPY	\$0000
95 00	STA	\$00, X	CD 00 00	CMP	\$0000
96 00	STX	\$00, Y	CE 00 00	DEC	\$0000
97	???		CF	???	
98	TYA		D0 00	BNE	\$0002
99 00 00	STA	\$0000, Y	D1 00	CMP	(\$00), Y
9A	TXS		D2	???	
9B	???		D3	???	
9C	???		D4	???	
9D 00 00	STA	\$0000, X	D5 00	CMP	\$00, X
9E	???		D6 00	DEC	\$00, X
9F	???		D7	???	
A0 00	LDY	£\$00	D8	CLD	
A1 00	LDA	(\$00, X)	D9 00 00	CMP	\$0000, Y
A2 00	LDX	£\$00	DA	???	
A3	???		DB	???	
A4 00	LDY	\$00	DC	???	
A5 00	LDA	\$00	DD 00 00	CMP	\$0000, X

DE 00 00	DEC	\$0000, X	EF	???
DF	???		F0 00	BEO \$0002
E0 00	CPX	£\$00	F1 00	SBC (\$00), Y
E1 00	SBC	<(\$00, Y)	F2	???
E2	???		F3	???
E3	???		F4	???
E4 00	CPX	\$00	F5 00	SBC \$00, X
E5 00	SBC	\$00	F6 00	INC \$00, X
E6 00	INC	\$00	F7	???
E7	???		F8	SED
E8	INX		F9 00 00	SBC \$0000, Y
E9 00	SBC	£\$00	FA	???
EA	NOP		FB	???
EB	???		FC	???
EC 00 00	CPX	\$0000	FD 00 00	SBC \$0000, X
ED 00 00	SBC	\$0000	FE 00 00	INC \$0000, X
EE 00 00	INC	\$0000	FF	???

APPLE II INTEGER BASIC SYSTEM 48K MEMORY MAP

FFFF	SYSTEM MONITOR	ON BOARD ROM
F800	SWEET 16	
F689	MINI ASSEMBLER	
F500	FLOATING POINT ROUTINES	
F425	INTEGER BASIC INTERPRETER	
E000	UNUSED	
D800	PROGRAMMERS AID #1 GRAPHICS, MUSIC, RELOCATE, MEMORY TEST	
D000	EXPANSION PROM SPACE	IO RESERVED LOCATIONS
C800	I/O RESERVED LOCATIONS	
C000	DOS (if booked)	HIMEM
95FF	USER RAM	
5FFF	HI-RESOLUTION GRAPHICS 2 (if used)	
3FFF	HI-RESOLUTION GRAPHICS 1 (if used)	
1FFF	USER RAM	
0BFF	SECONDARY TEXT PAGE (if used) LOW RESOLUTION GRAPHICS SCREEN 2	
07FF	PRIMARY TEXT PAGE LOW RESOLUTION GRAPHICS SCREEN 1	RAM
03FF	SYSTEM MONITOR, DOS VECTORS	
03D0	UNUSED AVAILABLE FOR SHORT MACHINE LANGUAGE ROUTINES	
02FF	GETLN INPUT BUFFER	
01FF	6502 STACK	
00FF	SYSTEMS PROGRAMS—ZERO PAGE	
0000		

A HEX ON YOU



HEX-DECIMAL

H1	DECIMAL	H2	DECIMAL	H3	DECIMAL	H4	DECIMAL
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49153	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

HEX ADDITION/SUBTRACTION

H2 H1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

HEX MULTIPLICATION/DIVISION

H2 H1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

HEX ASCII

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	Ø	@	P	-	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	/	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	:	K	▀	k	▀
C	FF	FS	+	<	L	\	l	!
D	CR	GS	-	=	M	□	m	}
E	SO	RS	.	>	N	^	n	-
F	SI	US	/	?	Ø	-	o	DEL

HEX-HEX-BINARY-DECIMAL

HEX	HEX	BIN.	DEC
F	Ø	0000	0
E	1	0001	1
D	2	0010	2
C	3	0011	3
B	4	0100	4
A	5	0101	5
9	6	0110	6
8	7	0111	7
7	8	1000	8
6	9	1001	9
5	A	1010	10
4	B	1011	11
3	C	1100	12
2	D	1101	13
1	E	1110	14
Ø	F	1111	15

SUPPORT MEMBERS OF THE COMPUTER RETAILERS ASSOCIATION ...



THEY WILL SUPPORT YOU.

For further details on the associations aims, membership, code of conduct etc.

Please contact: Ms. Heather Hodgson,
47, Creswell Road, Newbury, Berkshire.
Tel. (0635) 42486

SUPPORT MEMBERS OF THE COMPUTER RETAILERS ASSOCIATION ...



THEY WILL SUPPORT YOU.

For further details on the associations aims, membership, code of conduct etc.

Please contact: Ms. Heather Hodgson,
47, Creswell Road, Newbury, Berkshire.
Tel. (0635) 42486

Programming Practices and Technics

MICROCOMPUTER PROGRAMMING TECHNIQUES

COMPILERS AND INTERPRETERS

Martin D. Beer,
Computer Laboratory
University of Liverpool.

Part II



THE first article of this series looked briefly at the operation of assemblers and loaders. An assembler allows you to write your program as a series of machine instructions which are translated one-for-one into machine-code. This method produces programs which execute very rapidly, and, so long as reasonable care is taken, use the absolute minimum amount of store.

Whilst it is relatively easy to write the program once the problem has been redefined in a form which is suitable for computer solution, this redefinition is often a considerable effort. Much of the work required is of a systematic nature, and from the earliest days of computing it has been found possible to allow the computer to perform at least part of the task for us.

Early attempts (notably FORTRAN developed by IBM engineers in the 1950's to help them solve complicated mathematical formulæ using computers, and still one of the most popular computer languages today) were aimed simply at reducing the amount of work required to code the programs required. The computer can be programmed to handle memory management and to generate code to do sequences of integer or floating-point arithmetic operations. More recently considerable thought has been given to easing other parts of the program designer's task, such as the organisation of the Data required, and simplifying the flow of control (notable recent attempts in this area being PASCAL and ALGOL 68).

Two different methods of translating programs written in a high-level, or algorithmic language have been developed. A compiler works in much the same way as an assembler. The program text is translated by a special program which generates the machine code which can be executed by the microprocessor and places it in a file.

The difference is whereas an assembler generates only one operation code for each instruction, a compiler will generate the many operation codes needed to execute one high-level language statement. The programmer is relieved of the problems of allocating storage, and much

repetitive coding, since they can be dealt with automatically perfectly adequately.

A very competent assembler programmer would be able either to save storage, or to shorten execution times, (or both), but it would be at the expense of considerable effort. Once the code has been generated it must be loaded into the computer, ready for execution. It is possible to delay storage allocation to this late stage, and so merge a series of program segments, possibly written in different languages. A special program called a loader does this for you. The ability to develop program modules independently is very important, since they can be reused in other programs when required, so long as there is adequate documentation as to their use, and the interface to the main program.

Interpreters work somewhat differently. An interpreter scans a line of text, and executes it immediately. This means that the program is stored in the computer as it was written by the programmer (or more usually in a slightly condensed form). The interpreter usually operates in two distinct ways, editing mode, when the program can be entered, or altered, and execution mode, when the program is actually executed. This dispenses with the need for a separate editor, but the facilities offered must be very limited if the size and complexity of the interpreter are to be kept within reason.

The program must be stored in the limited memory available within the microcomputer so the interpreter eliminates unnecessary characters, such as spaces, from each line before adding it to the program. More space may be saved if further processing is undertaken. Keywords can be encoded in single bytes, so long as they can be recreated later.

If, in addition, numbers are stored in floating-point format, instead of as typed, further considerable savings result. Since the program is executed directly from the programmer's code monitoring and debugging are particularly easy. Errors can be associated with individual lines of code very easily, and variables can be associated

with the symbolic names associated with them by the programmer at every stage.

Sections of the program can be checked independently by executing them after initialising any variables with trial values since program execution can be controlled so intimately.

There are considerable advantages in using both an interpreter and a compiler, depending on the stage of development of the program. Since a compiler must produce object code for the target microprocessor, it has a far more difficult task than an interpreter. It is therefore much larger. Also some form of backing storage is required to store the code once it is produced. The advantage is that if the program is to be run many times without change, the translation is performed only once, saving considerable computer time.

The linking in of pre-compiled program segments is another considerable attraction in using a compiler. Once a program has been fully developed, and is in continuous use the continued use of an interpreter can be a considerable disadvantage since the immediate tracing and control facilities so useful at the development stage can lead to confusing and unpredictable error messages when used by others. Very clear documentation is required to cover events such as this.

Whilst BASIC is usually implemented with an interpreter and most other languages in common use with compilers, this need not necessarily be so. Several very good compilers have been written for BASIC and

interpreters have been written for other languages. An interesting possibility is the use of an interpreter to develop and test the program, and then compiling the final version for repeated use.

It would save a lot of work if a compiler could be written in such a way that it would generate code for many microprocessors. This is of course impossible since each microprocessor has its own instruction set. What can be done, however, is to write a Compiler which generates an 'intermediate' code, which does not necessarily correspond to the code needed for any particular microprocessor, but which can be run by using a program called a machine-code interpreter. This program takes the intermediate code and executes it, simulating the intermediate machine. The intermediate code can be designed to minimise the resources required by the compiler, which can run on the microcomputer system, if it is large enough, or on any other suitable computer.

The only reprogramming that is required to transfer programs to a new microcomputer system is the machine-code interpreter, a much simpler task than rewriting the whole compiler. If the machine-code interpreter is written carefully it is possible to achieve program execution times very close to those obtained when the compiler has been designed to generate directly executable machine-code. It is often possible to write the compiler in its own language and to run it using the machine-code interpreter, making the whole system transferable.

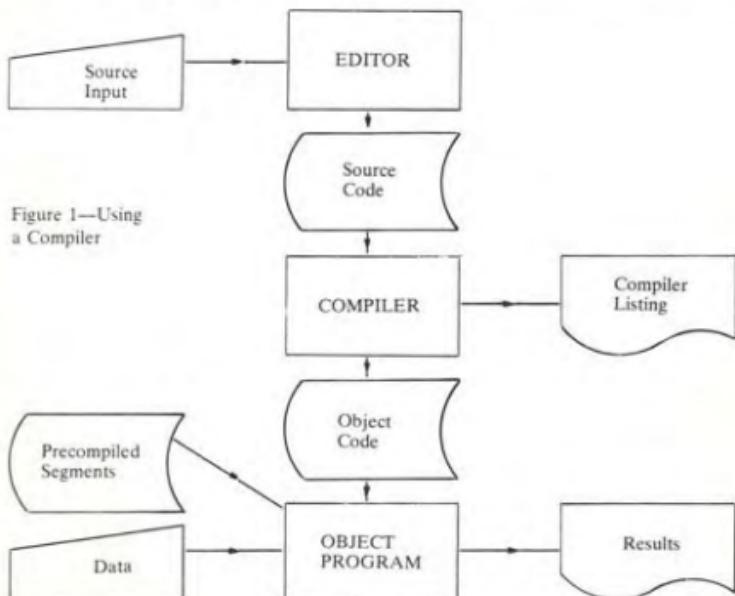
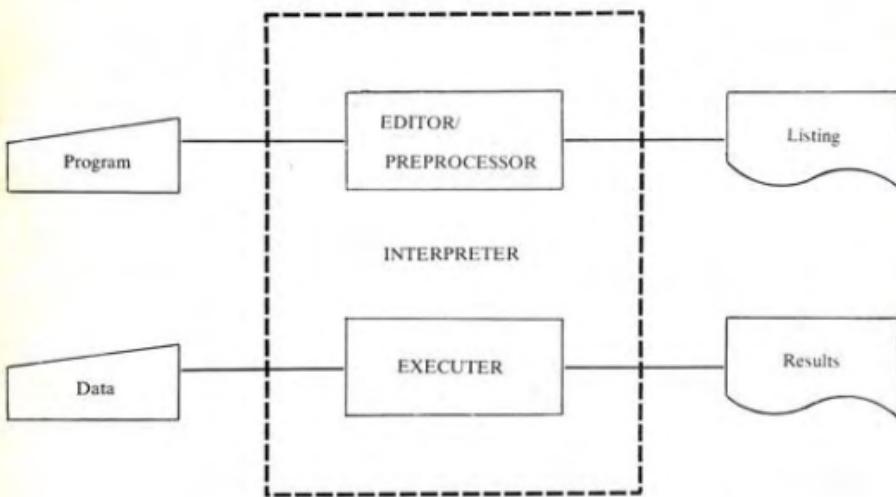


Figure 1—Using a Compiler

Figure 2: Using an Interpreter.



TREKKING BY 'JTK'



WITHIN 10 nanoseconds of coming out of warp the Klingon battle cruiser had fired its phasers. In that time Spatialcomp had informed Tacomp of the Federation freighter 5 miles ahead busily ferrying cargo to the planet surface below them both and Tacomp in turn had instructed Weapcomp to fire the phaser banks. It was traditional to open an encounter with speed of light weapons to inflict damage before the enemy could respond, the slower more powerful weapons being reserved till later in the engagement.

The attack had been inevitable for ten years now, ever since a crippled Federation freighter had drifted into Klingon territory giving the Klingons access to Federation tactical software.

Aboard the Federation freighter the sensors detected the Klingon cruiser and set off a course of events of galactic importance, determining whether the Klingons had finally overcome Federation supremacy. The freighters Execomp, busily supervising the unloading, switched out and Tacomp switched in, taking over command of the ship. Tacomp immediately directed the entire power of the warp drives into the rear deflector shields and started evasive action.

After checking a register to ensure there were no life forms aboard, the freighter accelerated under impulse power at 50 G and pulled a 20 G turn away from the planet surface. Simultaneously Comcomp sent a subspace signal to Federation H.Q. and a more important MASER encoder message to the planetary defence computer below.

Once the Federation were aware that the Klingons had their tactical software it started a race to ensure that the Federation would be ready when the Klingons attacked. On Earth two Hyper Intelligent Machines were run for a year and a half. One HIM simulated what the Klingons would do to overcome the Federation software advantage the other HIM simulated all the Federation alternatives to the simulated Klingon action.

The machines were locked in silent electronic conflict. The periodic reports of the simulation to Federation command were initially disappointing then disconcerting and finally frightening. They showed that no one side

had the advantage, that encounters would be inconclusive, that anarchy would once again prevail.

The phasers struck as the freighter nosed into its turn. The shields absorbing the bulk of the blow, yet the damage was still considerable—the remaining cargo was all destroyed, the communications antennae blown off and impulse power down by 50%.

Federation command had set up a committee of people to find the solutions to their problem. The committee was carefully chosen, people with degrees in lateral thinking, alien psychology, temporal mechanics and a dozen other disciplines. The recommendations of the committee were devastating, federation command took a week to regain their composure but finally they saw the logic, they saw that it would work and they saw that they had no option. Klingon and Federation ships were virtually identical, centuries of plagiarising the better features of each others designs had resulted in an optimum piece of engineering.

The fourteen on board computers worked at the maximum theoretical speed, restrained only by the speed electrons can travel at, they were crammed together in a two inch sphere so as to talk to each other more rapidly and they were exposed to the absolute zero temperature of space so that their Josephson junction technology would work. The only hardware change the committee made was to reduce the number of computers to twelve, this gave the Federation a speed advantage over the Klingons but resulted in a loss of flexibility. The new software didn't need flexibility.

On board the freighter Navcomp made its last calculations very precisely and accurately and quadruple checked them. The freighter jumped into Hyperspace and re materialised at the same co-ordinates as the Klingon cruiser, so they both occupied the same volume in space, their molecules interlocked, both ships destroyed. It took three seconds for the plasma bolt to reach the ships from the planets surface, and afterwards the debris was so small, it didn't even register on radar.

At Federation command they had a party to celebrate, supremacy had been regained.

NASCOM NOTES

AN unclassifiable collection of hints for Nascom 1 and 2 owners, that have appeared in the vicinity of our desks.

RAM Board and the Nascom 2

Nascom-2 constructors please, please check that you have cut the 4K EPROM select P5-link 12 on your 16K RAM Board. As standard these are enabled for addresses.

F000-FFFF causing an address conflict with the Basic ROM which sits at E000 to FFFF.

8K Basic Bug?

Test for correct printing of 'ESC' Nas-2.

```
10 FOR I = 1 TO 1000
20 PRINT I;
30 PRINT CHR$(27);
40 NEXT
```

The above routine should print I (1-1000) then print 'ESC', which clears the line. At various values of I, i.e.

14, 26, 34, etc., the 'ESC' character is not printed and the display scrolls.

I have tried an equivalent routine in machine code, which is O.K. I have also tried it with the print statements in machine code, called by the 'USR' function this is also O.K.

This problem was discovered when trying to control a transtel printer which uses 'ESC' and a number to change its printing format.

HUMBLE PIE

'DEF Function name (variable name) = expression where the function name must be FN followed by a legal variable name and a 'dummy' variable name. The dummy variable represents the argument variable or value in the function call. *Only one argument* is allowed for a user defined function

Examples of valid functions are:

```
10 DEF FNAVE (V, W) = (V + W)/2
12 DEF FNRRAD (DEG) = 3.14159/180*DEG
From Nascom 8K Basic documentation
Over twenty phone calls at the last count.
```

ERRATA

Sargon meets the Nascom 1

Line 30 of the listing should read:—
#9 ADD HL, BC; Next upper line

Our thanks to D. B. Newell for pointing this out. For those interested in graphics to go with SARGON, BITS and P.C.'S, 18 Rye Garth, Wetherby, West Yorks (Tel 0937-63744) sell a very nice Graphics Board, a set of Chess characters is an option.

M5

Page 20 §.2 Introduction

Commands which may be entered now are:

I (not 1) INPUT

1.1.2 The Current Valve

2) On encountering an identifier A-Z X takes the valve stored there.

A.1.3. Variables

As in most other languages, M5 has variables A-Z and a special one @.

Page 24 3.1 Commands

Line 13 should read

Use: RN > >>>

Page 25 Line 9 should read

E:R > >>>>>

4.8 Error Messages

Line 5

ID ERR X ...

Apple Pips Page 30

Integer Basic to Applesoft conversion should read

```
10 DS = " " : DIM TITLES(30)
20 INPUT "PROGRAM TITLE": TITLES$
30% POKE 76, PEEK(202) : POKE 77, PEEK(203)
40 PRINT DS$; "LOAD", TITLES$
50 PRINT DS$; "OPEN", TITLES$; ".TEX"
60 PRINT DS$; "WRITE", TITLES$; ".TEX"
70 PRINT "FP"
80 LIST
90 PRINT DS$; "CLOSE", TITLES$; ".TEX"
100 PRINT DS$; "EXEC", TITLES$; ".TEX"
110 END
```

An alternative is to add the following lines to your program at unused line numbers.

```
1 DS = " " : REM CTRL-D
2 PRINT DS$; "OPEN TEXT" : PRINT DS$; "WRITE TEXT"
: LIST: PRINT DS$; "CLOSE TEXT" : END
RUN this then in direct mode type
FP return
EXEC TEXT return
Our apologies for these errors.
```



Byting more off your Disk

ONE of the fundamental laws of microcomputing states that, no matter how much disk capacity you have, it is not quite enough for the latest project; 'conventional' methods of using a disk on a microcomputer system make for a lot of overheads—in particular a large amount of available space is 'wasted' on the DOS (usually on the first few tracks of a disk) and directory/Track sector lists. This space is all necessary—supporting the operation of serial or Random Access, text, numeric, binary, files etc. and their use from BASIC. To take a specific example the Apple mini-floppy has 35 tracks, made up of 13 sectors each of 256 bytes. A total formatted capacity of 113.75K Bytes. However on each disk tracks \$4-\$2 are reserved for the DOS. Portions of \$11 are normally reserved for the directory etc. All in all about 15% of available space is used by 'system' functions. In many applications the extensive support provided by the DOS is unnecessary but more capacity would be greatly appreciated.

Disk systems are normally classified as 'block-structured devices' data is not written on a character by character basis but instead written into a 'buffer' and where necessary the entire buffer written out or read from the disk. A buffer for disk is normally the same size as a disk sector. The overall control of reading and writing a sector to a specified part of the disk is handled by a low level set of sub-routines in DOS known as RWTS (Read/Write Track/Sector) in the Apple and various other designations on other systems, and their associated Device Control Block or Input Output Byte. (Most of this 'parlance' comes to the micro from mainframe practice.) These routines are fundamental to the operation of the disk and higher level DOS routines for creating and opening files, reading and writing files etc. will call them with appropriate parameters, to perform the menial tasks of interacting with the disk hardware. By setting up the appropriate parameters in the Input Output Byte and calling RWTS from a user program we can circumvent most of the intelligence of a disk system. On the Apple, RWTS provides four basic services.

Null Command—do nothing except position the head to the specified track/sector.

Read—Read the specified track/sector into a buffer (256 bytes long in the Apple case)

Write—Write from the specified buffer to the Track/Sector

Format : Formats a minifloppy into 35 tracks of 13 sectors each of 256 bytes.

RWTS returns any error as an error code. 4 are normally used.

Write Protect error : Attempt was made to write to a write-protected disk.

Read error: After 48 repeated attempts the system was unable to read the specified sector.

Drive error : Something funny is happening!

Volume number mis-match : The actual disk volume number does not match the expected one.

Similar routines available in most computer operating systems. If we set up the appropriate parameter in the Input/Output Byte with a format command and call RWTS with an unused disk in the drive (note: for all these operations DOS must be booted off another, conventional disk) we will have a neatly formatted disk with all 455 sectors available for our use. By calling RWTS we can read and write to any sector. Any file structures created will have to be maintained by the user program. Since our disks do not have any DOS stored the user will have to boot off another disk. What we have gained is capacity—116480 bytes to be precise and speed, as we are not going through any complex calculations before performing the operation.

Examples of applications that spring immediately to mind are stock control, mailing list, database management, graphics picture storage etc. Any application in fact where more speed and capacity would come in handy. To use RWTS we set up an Input/Output Byte in memory, a device characteristics table—Within the IOB we can specify slot, drive number, track/sector number, command (Null, Read, Write, Format), address of device characteristics table, expected volume number etc. In addition the address of a 256 byte data buffer. In the demonstration program I moved HIMEM down below a 256 byte area.

The program below demonstrates use of RWTS, it reads a special sector called the Volume Table of Contents off the disk and returns with the DOS version number, the volume number of the disk and where the disk directory begins. (See DOS 3.2 manual appendix C for further details.)

It uses Page 3 extensively:

\$300 - 310 Input/Output
\$311 - 314 Device Characteristics table
\$315 - 31C Machine Language Driver

With small changes it could be used as a support program for any of the previously mentioned applications.

5 Sets Himem below buffer) Change for smaller
55 Buff is the base address of the data) memory size
buffer	
60-170 Initialise various fields of I.O.B.	
180-290 Pokes I.O.B. into memory	
295-310 Pokes device characteristics table into memory	
320-330 Pokes machine language drive into memory	

340 Calls RWTS via driver

350-380 Print various statistics on the disk? Ends.

The program was written to demonstrate use of RWTS, it is not a particularly elegant way of calculating the DOS version etc. Similar programs would be used on

most other micros. For Apple Users who are expanding the routine pages 94-98 of the DOS 3.2 manual has further details of RWTS.

One addition needed are error handling routines (the easiest way would be to use the machine language drive to set a flag in page 0 that can then be peeked).

```

30
31.LST
5  HIMEM=38143
10 REM VTOC ACCESS PROGRAM
20 REM C. PHILLIPS
30 REM COPYLEFT 1980
40 REM ALL RIGHTS RESERVED
50 CALL - 936
55 BUFF = 08144: REM FOR 40K SYS
      TEH
60 TYPE = 01: REM 10B TYPE CHUS
      T BE 01 FOR DISK 11)
70 INPUT "SLOT NUMBER": S
80 IF S < 1 OR S > 6 THEN 70
95 S = S + 16: REM SLOT NUMBER + 16
96 INPUT "DRIVE NUMBER": D
100 IF D < 1 OR D > 2 THEN 90
110 VOL = 0: REM VOLUME NUMBER
120 T = 17: REM TRK# NUMBER
130 SEC = 0: REM SECTOR NUMBER
140 CMD = 1: REM READ COMMAND
150 PSN = S: REM SLOT ACCESSED RECENTLY
160 PDN = 0
170 PRINT
180 REM *****
190 REM POKE 1,0,B
200 REM *****
210 POKE 260,TYPE
220 POKE 229,D
240 POKE 273,VOL
250 POKE 225,T
260 POKE 212,SEC
265 POKE 274,17: POKE 275,3
267 POKE 276,H: POKE 277,149
270 POKE 278,B: POKE 279,B: REM
      UNISED
280 POKE 798,CMD
290 POKE 794,124: POKE 795,PDN
295 REM *****
296 REM POKE D,I,T
297 REM *****
298 POKE 795,09: POKE 796,01
310 POKE 795,219: POKE 796,216
320 POKE 795,169: POKE 796,2: POKE
      791,160
330 POKE 792,0: POKE 793,22: POKE
      794,217: POKE 795,2: POKE 79
      6,96
340 CALL 799
350 PRINT "DOS VERSION 3": PEAK
      (BUFF + 3)
355 PRINT
360 PRINT "VOLUME NUMBER": PEAK
      (BUFF + 6)
365 PRINT
370 PRINT "DIRECTORY BEGINS AT T
      RACK": PEAK (BUFF + 1): SE
      CTOK": PEAK (BUFF + 2)
380 END

```



STOP PRESS

BYTE SHOP

THE Byte Shop chain of 6 computer stores in London, Birmingham, Manchester, Nottingham, Glasgow and Ilford, which recently went into receivership has been purchased by Comart Ltd. of St. Neots.

TANDY

Tandy are to introduce a colour graphics computer system, to plug into a domestic television, using a number of custom integrated circuits produced by Motorola—including a version of the 6847 Video Display Generator.

SINCLAIR

Clive Sinclair is to launch a £100.00 Z-80 based micro by the end of February.

APPLE PRICES DOWN!

From 1st February Apples are getting cheaper.

The 16K Europlus Apple with video output is now £695.00. A disk drive with controller £349.00. With 4116 RAM seemingly dropping in price daily a 48K Apple II with disk has now virtually halved in price since its introduction 2 years ago! All this in these inflationary times!

PET

The PET appears to have a considerable lead in software and sales in the U.K. More PETs are being sold here than on the American West Coast! With the tremendous number in use some really good software is becoming available—the PET Programmers Toolkit, the Commodore Assembly Language development system (that's almost as good as those available for the Apple—biased editor's note), PETAID etc. Latest rumours are of a full Pascal implementation for a 32K PET with the Run-time utilities and a P-Machine in ROM.

How to get your **LIVERPOOL SOFTWARE GAZETTE** regularly

Imagine the disastrous effect on your life style if you missed a single issue. The possibility of trauma is easily eliminated by the simple expedient of acquiring a regular subscription at the all time bargain price of £6.00 for the next twelve scintillating issues.

Don't miss the chance of a lifetime
Fill in the form below.

Commercial and educational organisations requiring a large number of copies can buy in bulk at advantageous rates from:

**Computer Bookshop,
43-45 Temple Street,
Birmingham.**

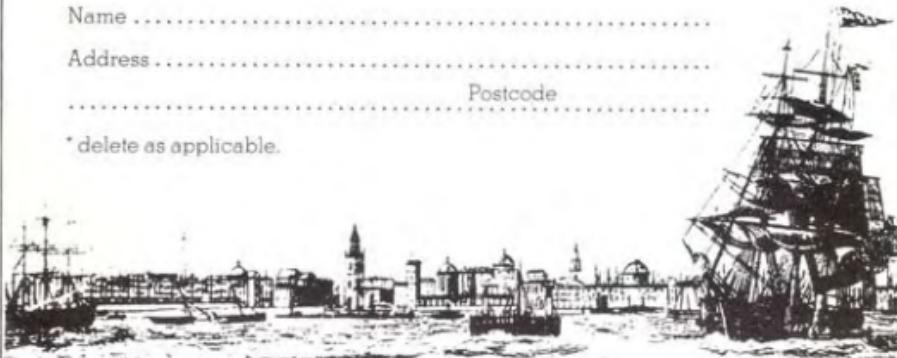
Please send me twelve issues of the Liverpool Software Gazette starting with the first/second/third* issue. Cheques and PO's should be made payable to Liverpool Software Gazette and sent to us at 14 Castle Street, Liverpool.

Name

Address

..... Postcode

* delete as applicable.





LIVERPOOL SOFTWARE GAZETTE

Printed by MERSEY MIRROR LTD., Media House, 34 Stafford Street, Liverpool L3 8LX, Tel: 051-207 7113