THE BEST OF
THE TORPET

PLUS

MORE

FOR THE

COMMODORE 64*

AND THE

VIC-20*

C64/VIC
Program
Disk
Included

EDITED BY
BRUCE BEACH

* Commodore 64 and Vic-20 are Registered Trademarks of Commodore Business Machines, Inc.

320
INFORMATIVE
PAGES

The Best of The Torpet

plus

More For The Commodore 64*

The Vic-20*

edited by

Bruce Beach

Copp Clark Pitman Ltd.
Toronto

Please note that **The Best of the TORPET Plus More for the Commodore 64 and the VIC-20**, and **The TORPET** are completely unrelated to Commodore Business Machines, Inc. CBM, PET and VIC are Commodore's registered trademarks.

ISBN 0-7730-4080-3

Copp Clark Pitman Ltd.
495 Wellington Street West
Toronto, Ontario M5V 1E9
Printed and bound in Canada

Canadian Cataloguing in Publication Data

Main entry under title:
The Best of the Torpet Plus More for the Commodore and the VIC-20

ISBN 0-7730-4080-3

1. Commodore 64 (Computer) — Programming — Catalogs.
2. VIC-20 (Computer) — Programming — Catalogs.
3. Computer programs — Catalogs.

I. Beach, Bruce. II. The Torpet

QA76.8C55B47 1984 001.64'2 C84-098754-4

FOREWORD

The Best of The TORPET was difficult to edit. Not because the material was difficult to understand, or poorly organized, or of poor quality, but simply because there was so much good stuff to choose from and squeeze into 320 pages. I started out with nearly 1500 pages of material and squeezed, squeezed and squeezed.

There are many things in the end that I had to leave out that, in the beginning, I felt just had to go in. Sometimes, it really hurt, but other times we were able to make some compromises by making some of the programs available on disks without putting the long listings in the book.

Undoubtedly, any reader or other editor would have made other choices. While I realized I could not please everyone, I was concerned that, in trying to please every level of experience, I would please no-one. Nevertheless, I hope I have come up with a book that will be very useful to every level of experience and especially to the beginners as they progress through the various levels.

The editor's intention has been to make this book a comprehensive resource for all those important things for all users, both new and old alike, that cannot be found in a single place elsewhere. For example:

1. Those important utility programs like BYTEDS, VIC MICROMON, WEDGE 64, etc.

2. And, most importantly, documentation for these programs and many others.

3. Catalogues, descriptions and sources for over 1000 free programs, like Prof. Peter Ponzo's programming tutorials.

4. Large, easy-to-read memory maps.

The real credit for this book, of course, goes to The TORPET readers and writers who have always supported The TORPET. I wish to also take this opportunity to thank the following persons for their significant contributions.

First of all, I must of course thank my wife Jean. She has put in twelve- and fourteen-hour days right along beside me in assembling and pasting up during the several months it has taken to put it together.

Darrin McGugan was also with us "for the duration", and he is the one who did most of the work of assembling and documenting "The Best Disks". Sue Spires did the typesetting along with help on the set-up by Mary Cornfield and Marie Collins. Paul Trachsler and anyone else (especially my daughter Bahai'a), who was on our local scene, could not avoid being looped into all sorts of arduous tasks such as the indexing. Everyone proofread and proofread, but proofreading was done especially by Marie Arnold, Ruth Beeley and Marie Matthews. Ed Niejadlik was our graphics consultant. Gottfried R. Walter was our programming technical consultant.

The Best of The TORPET team

— Front row, left to right:

Jean Beach,

Sue Spires;

Back row: Gottfried Walter,

Darrin McGugan,

Bruce Beach.

AUTHOR INDEX

Harold Anderson 97
Bruce Beach 2,9,175,214,215,289
T.J. Bos 154
David Bradley 42,144
Richard Bradley 217
Alfred J. Bruey 38
Jim Butterfield 15,79,82,292
Ron Byers 228
Hal Chamberlin 188
Robert A. Chandler 222
Dr. Frank Covitz 202
Steven Darnold 30
Tony Davidson 53
Elizabeth Deal 67,83,142,143
Rick Dendá 120
Doug Drake 135
Robert Dray 61,134
William R. Frenchu 24,36
Steve Garmon 46,131
Larry Goldstein 86,89,98
Mel Granick 166
Gary Greenberg 178
Herbert Gross 60
Colin A. Haig 226
Dr. Efraim Halfon 28,105,109
Greg Halley 40
Todd Hamilton 130
Velda Hardman 223
Mayland Harriman 52,137
Gary Hayes 149
Terry Herckenrath 47,48,93
Paul Higginbottom 196
T.H. Holmer 77
D. Howell 159
Michael Kelinert 153
Wm. Kendall 124
Garry Kiziak 18
Charles Kluepfel 62
Robert Kobenter 218
Stan Koma 148
Ron Kushnier 12
Ed Mansfield 174,176
Darcy Mason 264
Paul McClay 209
Theodore McDowell 219
Darrin McGugan 272,274,278
Margaret McRitchie 69
Joel Meers 145
Howard M. Mesick 155
T.C. Meyer 169
Hank Mrockowski 162
Mark Niggemann 101,307
Tony Ning 120
M. Petersmeyer 74
Dr. Piasecki 128
Steve Punter 212
Ron Radko 32
Mike Richardson 16,44,58,72,84,102,126,132,146,164,184
Steve Rimmer 116
Michael Ross 181
Neil Salkind 11
John Scott 171
Robert Scott 10
John Seitz 11
Dennis Sievers 55,57
Vince Sorenson 92,95,100
Terry Taller 179
Peter Tattersall 75
Paul Thompson 32
E. Toussaint 154
Paul Trachsler 23
Lee Urbanski 51
Tom Van Flandern 139
Fred Wallace 122
G.R. Walter 104,158,178,236
Don White 300
Gene Wilburn 150
Daryl E. Williams 153
David Williams 78,80
Leslie Wood 4,8
Wes R.D. Wraggett 186
Ian Wright 138
Alan Wunsche 264
Bill Yee 262,267

HUMOUR

Adams 131,160,224
Chaput 14,22,46,50,172,183,237,262,266,271
Patrick Corrigan 41,121,143,145,170,177
Joe Gravel 277
McWade 35
L. Poon 91,96,108,156
K. Pruner 9,81
Quillan 52,88,130,137,142,218,306
M. Richardson 60
Dan Sloan 66,94,129,152
C. Strutton 141
Ylimaki 7,38,55,57,77,83,99,125,144,149,194,213,216,235,308

TABLE OF CONTENTS

	PAGE
Introduction	1
C64	17
VIC	45
General Programming	59
BASIC Programming	73
Machine Language Programming	85
Other Languages Programming	103
General Hardware	127
Disk/Tape Drives Hardware	133
Printers Hardware	147
Data Base	157
Spreadsheets	165
Word Processing	173
Music	185
Telecommunications	211
Education	225
Best Programs	261
Best Disks	273
Maps	291

Note: *Approximate prices are given in some articles.*
Please check with your distributor for accurate prices.

Table of Contents
CHIPP Cartoon Strips

CHIPP Explains BASIC

by Mike Richardson, Orangeville, Ont.

	PAGE
GOTO statements for the beginning programmer	16
COLONS for the beginning programmer	44
How to put CURSOR CONTROLS into a program	58
How the STOP statement helps the programmer to debug a program	72
How to use the DIRECT MODE	84
How the READ statement interacts with the DATA statement	102
How the RESTORE statement interacts with the DATA statement	126
How the ON statement works	132
How the FOR/NEXT statements work	146
More on FOR/NEXT Loops	164
Still More on FOR/NEXT Loops	184

Introduction

PAGE

History of The TORPET 2
Bruce Beach, Horning's Mills, Ont.
The TORPET has been here since the early beginning of Commodore Computers. It started as four pages and has grown. Its growing pains are described for clubs and others starting similar magazines.

History of Commodore 4
Leslie Wood, Toronto, Ont.
The history of Commodore from small Typewriter repair shop to giant worldwide computer manufacturer.

Origins of the Originator 8
Leslie Wood, Toronto, Ont.
The founder of Commodore. The man and his philosophy.

Why Everyone Needs to Understand Computers 9
Bruce Beach, Horning's Mills, Ont.
Here are some good arguments for the kid who wants to convince his parents to get him a computer.

How to Survive The Price War Games 10
Robert Scott, Brantford, Ont.
Some more sound advice for the new computer buyer.

Buying through the Mail 11
Neil Salkind & John Seitz, Lawrence, Kan.
Both the pluses and minuses are examined.

Planning for Obsolescence 12
Ron Kushnier
Some real insights into the problems that face the new computer buyer.

Swapping and Sharing 15
Jim Butterfield, Toronto, Ont.
Our respected leader sets down some of the rules and guidelines for copying programs.

History of The TORPET

By Bruce M. Beach, Horning's Mills, Ont.

SMALL BEGINNINGS

The TORPET has reached its third anniversary (in Dec. 1983), and we are now printing 32,000 copies. For the next three months the TORPET will be distributed through all the independent (non-chain store) Commodore dealers in Canada.

The TORPET continues to support new publication efforts on the part of clubs throughout the world. We only ask that those making reprints from the TORPET to send us copies of their magazines and if they have a policy of paying their contributors to send the remuneration made out to the author in care of the TORPET so that we may forward it to the author.

Each month we receive a number of new magazines who are using TORPET reprints and we are often amazed by the growth and size exhibited by them. It may interest you to know the stages that the TORPET has gone through in its growth and for this reason we are publishing the following history.

When the TORPET began three years ago as a Commodore 'only' magazine there had been only one predecessor of note and there were two contemporary publications. The predecessor was called The Paper and it has merged with The Midnite Gazette which began the same month as The TORPET. The other contemporary publication was The Transactor published by Commodore Computers and now published by a private company in Canada.

Today there are dozens of club magazines that are larger than The TORPET was during its first year and there are six Commodore 'only' publications that are larger in circulation than the present TORPET. The larger circulation magazines (or planned magazines) are Compute's Gazette, Commander, RUN and POKE (first issues of the latter two to be published in November), Commodore Magazine, and Power Play (these last two are published by Commodore itself).

The TORPET doesn't expect to ever become a large-circulation slick magazine like the last six mentioned. It intends to keep its own inimitable style. We will never be as technical as The Transactor nor as free form as the Midnite Gazette. We will continue to address the new audience of computer users who are wanting to move away from the games-only aspect of their machines and onto a beginning understanding of the operation of the computer and its capabilities. We will probably move away from parochial types of club information that we have published in the past but we will continue to try to innovate and improve.

We hope that you have enjoyed The TORPET in the past and that you will enjoy it even more in the future, and if you are publishing your own magazine we still want to help you in every way we can to distribute information about the world of Commodore microcomputers.

IMPROVEMENT WITH EVERY ISSUE

Issue No. 1, Nov. 1980

The first TORPET was issued in November 1980. It was called the TPUG News and consisted of four pages run on a sheetfed offset press. It included Issue No. 1 of the Midnite Software Gazette, and a machine language checklist from Jim Butterfield's machine language course.

Issue No. 2, Jan. 1981

Carried The TORPET name for the first time.

Issue No. 3, Feb. 1981

Had first graphics, first ad (from 2001) and first ad rate schedule ($85 for a full page).

Issue No. 4, March 1981

First appearance of stylized masthead. First use of letterspacing in the typesetting. Over 300 subscribers. Increase to 8 pages. Changed from 4 column to three column format.

Issue No. 5, April 1981

The big jump to web offset. 16 pages. Now had 403 subscribers. Better stylized heads. First boxes for calendar, executive list, Bennett Box, etc. Change from 3 column to 2 column format. First listing of TPUG monthly disk release. First assistant editor (Barbara Bennett). Established policy of twenty-five percent or less of advertising.

Issue No. 6, July 1981

First cover photo. First half-tones. First color. 32 pages. First table of contents. First Butterfield Box. First listing of executive's phone numbers.

First cover price of $1 for computer store stands.

First classified ads (5 cents per word $1 minimum).

Issue No. 7, Oct. 1981

First web issue on bond paper. First 48 page double page spread ad (RTC). Included first extensive program documentation.

Issue No. 8, Jan. 1981

First man of the year issue (Chris Bennett). First use of hyphenation in typesetting. First of the dealer of the month series (Electronics 2001).

Issue No. 9, April 1982

First issue of regular monthly schedule. First schematics published. First insert section (BMB Compuscience). First advertising manager (Michael Hyszka). $1.50 cover price.

Issue No. 10, May 1982

First two color issue. First maps. First cover scoops. (CBM II and PET II). First $2.00 cover price.

Issue No. 11, June 1982

First Reader's corner. First registered overprinting (RTC ad). First extensive photo story (by John Easton).

Issue No. 12, Aug. 1982

First alphabetic listing of library (by David Hook). First sustaining member listings. First extensive artwork (Butterfield's PET family tree). First three side trim.

Issue No. 13, Sept. 1982

First pasted cover. First issue without major outside editorial content.

Issue No. 14, Oct. 1982

First horizontal printing of listings. (Had to reprint run to accomplish it.)

Issue No. 15, Dec. 1982

First full time editorial effort. First separate cover. First publication by separate publisher. First 96 page issue. First inside front table of contents.

Issue No. 16, Jan. 1983

The big jump to first four color cover (Man of the Year - Michael Bonnycastle). First Canadian mailing under second class registration pending. First ad for writers.

Issue No. 17, Feb. 1983

First collage cover. First insertion of return card. First front cover ear titles. First ad for cartoonists. First Hardware Hacker box. First magazine rather than newspaper press scheduling. First use of premium 70 paper.

Issue No. 18, March-April 1983

First professional photo cover. First use of bipad. First inside cover 4 color. First cover credits. First newsstand distribution. First cartoons. First strip. First page top borders. First theme issue. First blue inserts.

Issue No. 19, May 1983

First Canadian mailing on second class permit. First full page regular strip. First paid series. First editorial board listing. First associate editor. First This and That column.

Issue No. 20, June 1983

First U.S. mailing on second class pending. First listing of ISSN number.

Issue No. 21, July 1983

First use of separate bindery. First full time advertising manager. First Canadian mailing without envelopes.

Issue No. 22, Aug. 1983

The big jump to first regular 96 page issue. First printing in two sections. First use of filled reverse printing (Mirage Concepts ad). First mailing on U.S. second class mailing permit. First insertion of editorial address. First advertiser's index. First printing of List Me files. First U.S. mailing without envelopes. First inside cover advertising printing with bleed.

Issue No. 23, Sept. 1983

First 96 page issue with full editorial content. First use of end of story indicators. First use of in-house headliner. Most extensive use of story dividers. First issue with all registration numbers inside. First use of composite half-tones in advertising.

Issue No. 24, Oct. 1983

First commissioned comic strip.

Issue No. 25, Nov.-Dec. 1983

Third anniversary issue with first internal four color sheets (Richvale Telecommunications). First complete distribution to Canadian Commodore independent dealers.

History of Commodore

By Leslie Wood, Toronto, Ont.

In just 25 years, a small typewriter sales and repair shop tucked away in downtown Toronto, Canada, has been transformed into one of the hottest personal computer companies in the world -- Commodore International Limited.

Shipping more units world-wide than any other computer company, Commodore has grown from sales of $46 million (U.S.) in 1977 to over $680 million (U.S.) in fiscal 1983 (year ended June 30). And much of that success is due to the entrepreneurial instincts of Commodore's founder and present vice-chairman, Jack Tramiel.

The Polish-born Tramiel survived Nazi concentration camps to immigrate to North America and, in 1958, open his own typewriter shop in Toronto. Tramiel has always had a gift for anticipating future home and business electronic needs -- and the ability to move quickly to fill them. Commodore's progress is a testimonial to that trait.

Over the past quarter-century, Tramiel has led Commodore on a heady ride through adding machines, electronic calculators, digital watches and the introduction of the personal computer age. Together with his skilled management team around the world, he is still considering: what's next? Commodore, in fact, is widely acknowledged as a company that puts into action a smart but simple rule: hold onto the old for as long as it is good and change to the new the moment it becomes better.

IN THE EARLY YEARS

During those early years, Commodore grew from typewriter repairs and sales to typewriter manufacturing, with the acquisition of a factory in Berlin, West Germany. Early in the 1960s, Tramiel began selling and servicing a wide range of office equipment, and distributing nationally for an office furniture company.

1965

In 1965, Commodore acquired the furniture manufacturer, and moved his operation to what is now Commodore's present Canadian headquarters. Commodore still manufactures office furniture (mainly filing cabinets and desks, plus metal housings for the CBM 8032 and SuperPET) at this plant in Scarborough, Ontario, and has expanded operations to three offices and two manufacturing plants in the Toronto vicinity.

Also in 1965, Tramiel met Canadian lawyer and financier Irving Gould, who later became Commodore's chairman. These two formed the head of the team that built the Commodore we know today. One of the first things this team did was to sell Commodore's adding machine plant and find a company in Japan to make adding machines for Commodore to distribute. While in Japan, Tramiel got his first look at an electronic calculator, and he quickly deduced that this product would mean the death of the mechanical adding machine. With the Commodore philosophy that, "if we are not our own competition, then someone else will be", Tramiel moved quickly and found manufacturers to produce electronic calculators under the Commodore name. Thus, the company was right there in the market when it began to take off.

The company began manufacturing its own electronic calculators in 1969 using Texas Instruments chips. In fact, Commodore was the first company to bring out a "hand-held" calculator, the C108, an example of what has become a long history of Commodore "industry firsts" in marketing value, innovation and performance in new products. It is interesting to note that this product was sold at much the same price, through similar distribution channels and to similar customers, as is the popular VIC-20 today.

1974

Up to 1974, Commodore expanded its lines of calculators from simple four-function machines to memory machines, scientific machines, and keyboard programmable models. Commodore was largely dependent on third parties for the chips and displays that went into the products it was making.

1975

In 1975, Texas Instruments decided to go into business against its own customers by manufacturing calculators. At the same time, chip prices dropped to $1 from $12, and Commodore was caught with a big inventory of chips and calculators while market prices plunged. It was this incident which led to Tramiel's decision that Commodore would be a company that controlled its own destiny, and not be at the mercy of other manufacturers.

1976

Commodore purchased MOS Technology, one of its semi-conductor chip suppliers, in 1976, and

worked its way to become vertically integrated. This vertical integration allows Commodore to supply its own needs, and it gives the company significant lead time in new product development, which means manufacturing cost advantages — and that, in turn, translates into price performance benefits for consumers.

The acquisition of MOS Technology was followed in the next 18 months by two further key investments: the purchase of Frontier, a Los Angeles chip manufacturer complementary to those produced by MOS, and the acquisition of Dallas-based Micro Display Systems Inc., a manufacturer of liquid crystal displays. As a result of these acquisitions, Commodore had in-house expertise and production in more key technologies than most electronics companies several times its size.

Also in 1976, Commodore reorganized its corporate structure as Commodore International Ltd. and moved its financial headquarters to the Bahamas and the operations headquarters to Wayne, Pennsylvania (it has since re-established in West Chester, Pa.).

1977 — PET INTRODUCED

The next year was the watershed for Commodore when, in 1977 — still anticipating the future in true Commodore style — the company introduced its first personal computer: the PET.

The PET (Personal Electronic Transactor) uses the MOS-designed 6502 microprocessor, which is also used by some of the competition. It was the original machine launched at the Hanover Fair in Germany and the Consumer Electronics Show in the U.S.A., that helped give birth to the personal computer market of today.

The PET sparked another period of rapid growth which is still underway today. It was marketed world-wide and really took hold in the European market because of the widespread, loyal dealer network Commodore had developed in its distribution of calculators. Commodore dominates the personal computer market in Europe today with more than 50 per cent of the market in many countries. In fiscal 1983 (year ended June 30), European sales reached $155.6 million (U.S.), almost 23 per cent of Commodore's total sales.

After the PET line was completed with the 4000 and later the CBM 8000 series micros, the next major product from Commodore was the very popular VIC-20. The prototype of the VIC-20 was previewed at the National Computer Convention in Chicago in 1980, and it was first launched in the Seibu Department Store in Tokyo, Japan, because, as Jack Tramiel said about the threat of competition from Japan, "The Japanese are coming; therefore, we must become the Japanese."

Commodore sold 800,000 VIC-20s world-wide in 1982, reached the one million mark early in 1983, and they are now being shipped at the rate of 100,000 units per month.

Commodore didn't stop with that success either, but continued research and development and, in August 1982, shipped the first Commodore 64. By the end of that year, aided by the single biggest advertising campaign in Commodore's history, the 64 had already passed the Apple II in monthly unit sales. And, by March 1983, the 64 was being shipped at the rate of 25,000 machines a month.

Both the VIC-20 and the 64 are sold through mass merchandise retail outlets, as well as computer dealers and selected electronics stores, a successful marketing technique that has since been emulated by other companies.

Commodore has now become the largest unit seller of microcomputers in the world. And, according to a Dataquest study published in Electronic News recently, Commodore is No. 1 in computers priced under $1,000, with an estimated 43 percent dollar share in the U.S. Maybe this is one reason why the 'Commodore 64 Programs Reference Guide' is currently the top-selling computer book in the U.S.

As well as the obvious success the company has achieved in the home market, the Commodore name is familiar in both the business and education markets for personal computers. Commodore is one of the leaders in small business computers with its SuperPET and CBM lines, and the 64 is also being used for a number of functions in small business.

EDUCATION MARKET

The education market is another area in which Commodore is a front-runner. In Canada, for instance, Commodore holds about 65 per cent of the national market for computers in education. Penetration is also significant in U.S., British and European schools and universities.

Commodore has become an international company, with manufacturing facilities in Japan, Hong Kong, West Germany, the U.K., Pennsylvania and California in the United States, and Scarborough, a city within Metropolitan Toronto,

To the left: The Commodore portable Executive 64, weighing 27.6 pounds, travels easily. It has 64K RAM, a built-in, five-inch monitor, and floppy disk drives with 170K capacity.

Below: The advanced Commodore "B" series business microcomputer with a minimum RAM of 128K, expandable to 896K.

Canada. In fiscal 1983, world-wide sales increased 44.7 per cent over 1982's $304.5 million (U.S.) to reach over $680 million (U.S.). By the end of fiscal 1984, Commodore will be a billion-dollar-plus company.

Wall Street financial analysts who follow Commodore (shares have been traded on the New York Stock Exchange for three years, and on the American Exchange several years prior to that) state that much of the company's success is due to its flexibility and willingness to adapt quickly to -- and even lead -- changes in technology and in the marketplace. Jack Tramiel puts it more simply: 'The minute you're through changing, you're through.'

Commodore International has the most complete line of products of any microcomputer manufacturer, with models and software specifically geared to the education, business and home markets. The company's track record of tradition and steady growth have resulted in an organization whose sophistication in research and development and in product engineering are second to none.

This commitment and dedication to research and development -- over $37 million was invested in R & D last year -- will lead to advances in technology and product application from Commodore in the years ahead. The company is driven by technology, and prides itself not only on giving its customers the products they want, but on introducing products the public didn't even know were available.

Commodore has programmers, systems designers and engineers working full-time to develop improved microprocessors, more efficient manufacturing techniques, enhanced quality control procedures, improved product design and engineering and, perhaps most importantly, an accelerated software development program.

Commodore is further expanding its software development in the United States and Canada with both in-house and external programming teams. The results of this program will certainly be evident to users of Commodore computers throughout 1984.

Commodore remains a firm believer in the adage that, if you just stand and watch the world go by, it will. So, the company continues to advance with a planned series of new proprietary systems, including a family of advanced microprocessors and peripheral integrated circuits for high-speed, low-power battery-operated computer systems, and improved video graphics. In addition, investigation into advanced microprocessor architecture is well underway that could lead to even lower-cost, 16-bit Commodore computers.

RECENT RESULTS

The most recent results of Commodore's high-level quality and value approach are the advanced "B" series business microcomputer and the portable Executive 64. The "B" series has a minimum RAM configuration of 128K, expandable to 896K. It is ideal for variable work situations, especially where high output levels are demanded. The Exec 64, weighing only 27.6 pounds, can go anywhere with no difficulty. It has 64K RAM, a built-in five-inch monitor and floppy disk drive with 170K capacity.

Another recent step has been the development of a sophisticated new voice synthesizer for the Commodore 64. The Commodore speech module plugs directly into the Commodore 64, and at present has a vocabulary of 235 words. This is the first voice I/O product to be developed at the company's Speech Technology Division in Dallas, Texas.

Also, Commodore's first consumer robot will soon be announced. Robotics is a challenging field of consumer electronics which has not yet been fully explored, and the company is excited about the potential in this area.

Commodore celebrated its 25th year with an international extravaganza held in Toronto, Canada, early in December. The "World of Commodore" Show was the first truly international computer show to be orchestrated by a single microcomputer company.

In fact, looking at the history of Commodore at the close of its first quarter century, it is easy to see that the company has consistently been a leader in recognizing change and leading the electronics industry into the changes. But, more than studying history, Commodore is a company that creates the history. Just watch.

With all of the micro chips going into modern weaponry, some future Julius Caesar will likely say: I come, I.C., I conquer. Oh well, when in ROM do as the ROMans do.

Ylimaki

Origins of the Originator

By Leslie Wood, Toronto, Ont.

The story of Commodore is completely entwined with Jack Tramiel, founder and architect of the company, and no story can be complete without a little bit of the flavour of the man behind the company.

Tramiel was born in Poland, and survived the Nazi concentration camps to immigrate to the United States after the war. His first association with the industry that evolved into the computer era was in his army days at Fort Dix where he became involved with the repair of typewriters. Once into civilian life, this was the business he pursued, and he even drove a taxi in New York City to help establish himself.

It was the typewriter experience that led to the start of Commodore a few years later when Jack moved his wife and two sons to Toronto in 1958 and started his own typewriter repair business at 2 Toronto Street, in the city's downtown core. Only the likes of Tramiel could have envisioned the Commodore International of today from that small repair shop.

Later in 1958, having grown to a strength of five employees, the company moved to more spacious quarters at 1905 Davenport Road. Two more moves for expansion purposes brought Commodore to 501 Yonge Street and then 630 King St. West at Bathurst in 1959, where it continued in sales and repair. The number of expansion moves in those early years attest to the hustle and hard work Tramiel put into his company, and the successful results he achieved.

Commodore is very much a result of Tramiel's anticipation of the future, and it's corporate philosophy is purely Tramiel's. Some of Jack's own thoughts are undoubtedly the best way to describe him:

ON CUSTOMERS

We produce for the masses, not the classes. Quality and service is our commitment because, if we don't give our customers the best, they will know it.

ON TECHNOLOGY

Commodore is driven by technology. We don't only introduce the products the customer wants, we introduce products the customer didn't even know were available.

ON MUTUAL IMPROVEMENT

Never settle for doing things the way they were done in the past. Always find new ways to do things better, cheaper and more efficiently.

ON SUPPLIERS

Never buy a product if you don't know what it costs to make it. Never buy below the supplier's cost to drive him out of business. Give him a profit, but never more than our company makes.

ON FINANCIAL RESPONSIBILITY

Treat evey penny as your own.

ON BUSINESS

Business to us is not a sport, it's war. We are not here to play the sport but to win the battles. We will win because we work harder and smarter, and serve our customers better.

ON THE FUTURE

We're always looking at the future because we're helping to create that future...but the work is always done in the present.

JACK TRAMIEL

WHY EVERYONE NEEDS TO UNDERSTAND COMPUTERS

By Bruce Beach, Horning's Mills, Ont.

I predict that in the next few months we will begin to see a reaction against the current enthusiasm for widespread computer literacy. The push will come largely from those who feel left out and threatened by the technology which they do not understand but there will be some very visible and very qualified experts who will also voice their reservations.

The question will be raised as to why in this age of specialization everyone needs to understand computers anymore than they need to understand automobile mechanics or any other technology in order to be able to use it. Why should everyone learn BASIC or any other form of programming? One does not need to know how to program a computer in order to use it any more than one needs to be an auto mechanic in order to drive a car. This will be the line of reasoning.

But there is a difference. Digital computers are logic machines and have a close kinship to man and his reasoning faculties. In my day (I am telling you my age) the classical education required one to learn Latin and Greek, not that one expected to ever meet any living Romans or Greeks. It was the intellectual discipline itself that was valued.

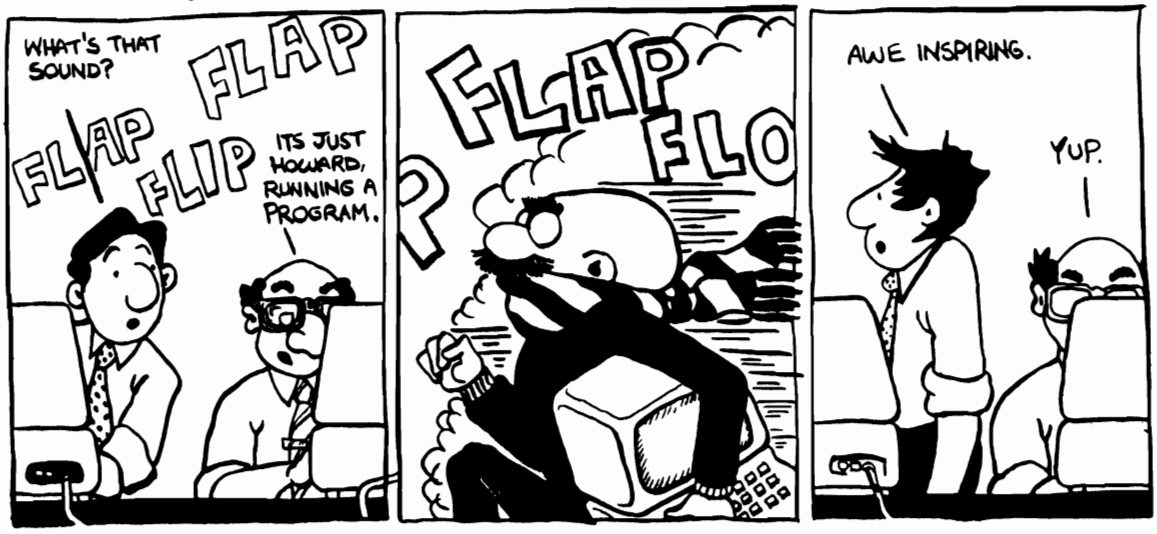
Socrates proposed that students learn the discipline of Euclid's geometry before tackling philosophy. Today's universities have similar requirements and yet surprisingly (an early indicator of the reaction) some will not give credit for high school computer courses. I, on the other extreme, feel that computer courses should be **required**.

The computer certainly requires as much intellectual and logical discipline as Euclid, Latin, or Greek and the teacher (the computer itself) is an infinitely patient teacher that never tires and **always** remains perfectly logical. When one adds to this the benefit of individual instruction and the ability to progress at one's own rate, how can there be caveat?

But there still remain objectors. "Many of the students will have no practical use of the skills they are learning", they will say. Wrong. The skill they are learning is not programming but logic and they have great need to learn logic. A society that places such stress on physical gymnastics will do well to place an equal emphasis on mental gymnastics, one more part of the Greek triad.

The fact that one has nothing to program does not mean they should not know how to program. Most students who learn to write English have nothing to say either. In fact, I know a fellow who speaks seven languages and has nothing to say in any one of them. In our educational systems we try to teach many to write in the hope that at least a few will have something to write about. Let us do the same with computers.

HOWARD by K. Pruner



How to Survive the Price War Games

By Robert J. Scott, Brantford, Ont.

Are you one of those people who spent $100 on a calculator that could add, subtract, multiply and divide, but wished you had spent the extra $50 to get one that did square roots? Perhaps you own a $129 digital watch. If the cost of microcomputers continues on the same slide, we would all be well advised to hold off on purchasing until they become available as prizes in popcorn boxes.

I am writing this using a $100 word processor that now sells for $39, and a $900 Commodore 64 that now can be purchased for $300 or less. I know it sounds a little crazy, but I'm not alone. I'm really not even that upset; after all, if it wasn't for us consumers, the prices would never have dropped. I would not have been able to replace my calculator for $19 with one that will do 10 times as many functions, or buy a $25 watch that is also a stopwatch, a calculator (that does square roots), and even wakes me up in the morning. What the heck, I will probably replace the computer someday.

I actually felt intelligent after having learned from the calculator. I didn't buy a $3000 computer three years ago, or even a $2000 computer two years ago. I waited until they reached $1000, and then jumped in thinking they couldn't go any lower. Wrong!! I'm not even going to mention printers, but I have gone through two in six months.

Where does it end? With my 64, though they are now cheaper, I assumed they couldn't get better, at least not for the same money. Wrong again!! Now I hear rumours that eight-bit machines will soon be obsolete and megarams are on the way.

Finally, we come to the reason I am writing this article, other than to make more money to help fund some manufacturer's research and development programs. When should one buy a computer? How can we survive the price war?

There are essentially two questions to be answered. The first is: what do I want the computer do to for me The second is: how much am I willing to pay, or should I say how much can I afford to pay?

When assessing what you want a computer to do for you, it is good to keep in mind that there are going to be things you didn't think you wanted until you find your machine will not do them. This is a strong argument for expandability. Due to the nature of the industry, expandability is almost universal, but, nevertheless, it is a consideration.

This way, you will feel great when you go to expand. The price of peripherals is sure to be much lower than when you first bought your machine. Some of the things you should consider, though, are word-processing, games (why not?), spread sheet and other business applications, data base accessibility (which is becoming more and more useful), and programming features such as languages and editing characteristics. I have found full screen editing extremely useful, and again it is almost unversal in the world of micros.

Then there is keeping up with the Jones's, actually not a bad bet in the computer field. It is nice to be compatible, not just friendly, with your neighbours. Often, in fact, it helps to see what your friends use their micros for when you are deciding what you want to do with one.

The other major way of finding out how you might use a computer is by reading magazines like this one. The articles may be interesting but the ads are also quite useful. They give you lots of ideas on what's available for different machines in both hardware and software.

If, after all this, you are still not able to determine exactly why you need a computer, but you know you do, it may be necessary to pull out all of the stops and use the same reasons (excuses) that I did. They are educational, and it's my hobby. After all, lots of people spend thousands of dollars on golf. If the latter is your reason, then you probably won't have to wait for the companies to develop the computer you need. This leaves us with the second question: how much to spend?

The answer: whatever it costs to get what you need. Right? Well, unless you are Herman Hollerith, money is going to be tight. If you can't afford a computer to do what you want, it's not a bad idea to wait a little while until the computer you need reaches the price you're willing to pay. If you become impatient waiting, it may be necessary to join a USER group where other users will discuss their habits and perhaps ease your tension. The last approach to the problem is back to the popcorn box or, in our terms, the entry level computer. These are usually very cheap (-$50), and often equally limited. At this level, though, you can always use it for a doorstop when it has outlived its usefulness.

There now, I'm finished. Is it clear to you what you need to know before you buy? I didn't think so, but I hope you have at least had a chance to reconsider your position. I also hope that, whatever you decide, your personal computer fits you personally, and the price doesn't drop through the floor the day after you buy it (Murphy's Law No. 473).

Buying Through The Mail

By Neil J. Salkind & John K. Seitz, Lawrence, Kansas

Customer: I really like some of the features of that new computer. What did you say the cost was?

Dealer: $695.00

Customer: Hmm. That's not a bad price. Let me see how I do this month, and I'll get back to you.

A few weeks later:

Customer: Listen, how do I program my word processor so that I can get correspondence quality output on that printer you're selling? I just couldn't resist the mail order price of $499.00.

Dealer: Hmm.....

This kind of interaction between computer dealers and customers represents a dilemma that more and more dealers are facing. What do I do about customers who seek advice about equipment that they purchased from someone else? Do I charge them for assistance by the hour? By the question? Can I afford to ignore them at the cost of losing further business? Should I spend time with them rather than devote it to the retail end of my business and other customers?

The question we would like to explore here is what buying through the mail can mean to you, your dealer, and the quality of your experience with your computer system after your purchases have arrived.

While mail order firms sell computers, printers, disk drives, software and practically anything else at a great savings to the buyer, the local dealer usually has available a smaller selection plus (we hope) knowledge, experience, and maintenance service.

This knowledge and experience doesn't come as a "software package", but needs to be developed.

When a dealer sells a line of computers or peripherals, a great deal more time and money goes into the retailing of those products than just inventory and overhead costs.

They usually have the following kinds of costs and time commitments associated with any line of hardware or software.

HIDDEN COSTS

First, most dealers attend national conventions such as COMDEX incurring various types of expenses, to maintain a level of expertise their customers should expect.

A second major "hidden" cost, is the payment of consultants to help modify new hardware or software to fit a particular system that is already established and running and has a large consumer following. People like to have new things modified to existing systems rather than have to relearn a new system.

When local people cannot be of assistance long distance calls to the service department of the manufacturer or their suppliers become necessary for answers to technical questions about the operation and capabilities of equipment. One dealer we know regularly has phone costs of over $200. per month just to cover inquiries on one system.

Fourth, usually there is sales training required by the vendor or manufacturer of the line before the dealer can sell the product. Although this training is not at the cost of the individual dealer, the time away from the store and travel expenses represent additional cost.

Fifth, maintenance training is often required by the vendor before the dealer can be certified to repair the particular hardware product. For example, one large printer manufacurer requires 10 full days of training before a dealer can become a repair center. Until the training is completed, they will not ship parts or repair manuals.

All the costs associated with these activities are besides the regular overhead involved in operating any retail business such as rent, utilities, advertising, salaries, and so forth.

These points are not plugs for why you should buy from your local dealer, rather than from the mail order firms that advertise in all the popular magazines. Rather, it is an explanation of some of the hidden costs associated with bringing equipment "on line", so that the dealer can make the system available and reliable.

WHAT ARE THE ALTERNATIVES?

What are the alternatives for the dealer, when faced with a situation like the one which opened this article?

One alternative is to answer all questions that any customer might have about equipment, regardless of whether they purchased it from the

dealer. A clear plus to this strategy is that people keep coming back for help, and perhaps new business. It's important to remember however, that most people will buy big items through the mail so that the $20. software package purchased from the dealer might not be worth his or her time and commitment in answering hours of questions and providing "free" instructions (for you — not the dealer).

A second alternative is to help only those people who purchased from you. Needless to say, this can become sticky. To begin with, many of the retail sales in any business operate as a result of referrals. If a dealer chooses not to help someone, that person might very well not mention that this or that dealer was helpful (especially when the consumer *really* needed it!)

Worse yet, the message about the dealer might be derogatory in nature. In addition, what if a consumer buys a computer, disk drive, modem, and monitor from the same dealer, but not the printer? Does the dealer answer questions about the disk drive, but not about the interface between the drive and the printer?

There really is no clear solution. What the consumer needs to remember, is that with the introduction of any line into a retail establishment there are costs associated with the provision of full service. If local stores are not supported, it is often questionable whether needed services will continue to be available.

What the dealer needs to keep in mind is that even if people do buy through the mail, they may very well depend upon you for future service, purchases, and that all important referral at the cost of some dealer time now.

* * * * * *

Neil J. Salkind is an occasional mail order buyer, and John K. Seitz is an occasional giver of free advice and a computer dealer. Both live in Lawrence, Kansas.

Planning for Obsolescence

By Ron Kushnier

In a recent Compute article, Jim Butterfield matter-of-factly stated that, in his opinion, the VIC-20 will "fade away" in a few years. I have no argument with that statement. I too believe it. Yet, seeing it in print made my mind gasp. Things are moving so fast, products are going on and off the market at such a rapid rate, that it is not surprising the following event occurred at a recent computer club meeting.

I had brought in my original 8K PET which, I must say, is still in mint condition. A young member of the club came running up and exclaimed, "Boy, another new model! Commodore is really something! Look at that, a built-in cassette unit! What will they think of next?"

It broke my heart to tell him that what he was seeing was the great-grandfather of the present-day CBM computer.

But that's the way things are in this "Future Shock" world of micros.

How can we live with such goings on?

How can we decide when to buy and when to wait?

And, more important, how can we plan for obsolescence?

In the world of computers, obsolescence is a subjective term. My single board "KIM" (vintage 1976) would be considered by many as obsolete. Yet it still performs the same functions as it did back then. It has all the bells and whistles, all the options that were ever made, and has never had a failure. But try to find a buyer for it — impossible!

My experience with "KIM" brings out three areas to consider when dealing with obsolescence.

The first question which must be asked is "For whom is the product obsolete?"

WE CAN CATEGORIZE BUYERS INTO THREE TYPES

There is the ***"Applications Buyer"***

This is a person who buys a computer with a particular application in mind, and who satisfactorily solves his problem with that computer. He certainly does not complain that his computer is obsolete.

There is the ***"Ubiquitous Computer Buyer"***.

This person expects his computer to do everything from high density color graphics to 80-column word processing, all at super speed and precision. This type of person is apt to be disappointed and dissatatisfied with any computer he buys. He will constantly be on the lookout for something newer or better.

Finally, there is the ***"Computer Experimenter"***.

The "Computer Experimenter" is more fascinated by the idea of what a computer can do than actual applications. He is the guy involved in advancing the technology and probably accounts for most of the published computer articles.

The experimenter finds himself in an unfortunate situation. Unless he is independently wealthy, he can never keep up with the rapid changeover in equipment. To him, machines become obsolete before they've even had a chance to be fully explored.

Perhaps I should mention a fourth category of buyer, the ***"New Educational Buyer"***.

This person is just breaking into the computer field and is not sure what his needs will be. He usually settles for a low-end micro such as the VIC-20 or the Sinclair ZX-81. The small initial cost can be written off as an educational expense.

At first, the newcomer is usually ecstatic with his purchase. However, once the novelty and educational value have diminished, the "New Educational" buyer is reduced to one of the three previously-defined categories, and is faced with the same decisions.

Each of our three buyers has his own view and definition of "obsolete".

An example of how each would view the purchase of a VIC-20 might be enlightening.

The "Applications Buyer" probably saw the VIC as one of three possible "Games" machines, the others being the ATARI and INTELLIVISION. The VIC provided more flexibility at only a slightly higher price. So the purchase was made. He is satisfied with his machine because it does everything that he expected it to do.

The other two buyers are not happy. They *are* satisfied with what the VIC is, but they are not satisfied with what it is not. They complain about slow tape speeds, lack of a "proper" amount of memory, and only a 22-column screen.

This leads us to the second area of concern and to another question.

WHERE ARE WE HEADED?

In the "KIM" example, it was not until I had amassed a large amount of memory, an ASCII keyboard, and a huge assortment of other hardware and software, that I asked myself, "Where am I headed?".The answer was that I was heading toward a system that spoke "BASIC". Unfortunately, by the time I achieved that end, my "computer" covered an entire table, and had to be turned on through a complex procedure by three separate power supplies. My "KIM" was obsolete by now, at least to the buyers market, and all the money I had spent on "add-ons" was lost.

What I am proposing, then, is that you ask yourself that all-important question, now. If you are not happy with your system as it is, wouldn't it be better to trade up *now* while your present computer still has value It seems foolish to me to start with the "add-ons" only to produce a bigger "obsolete" system a few years from now.

The third area of concern affects all of us buyers. This is the area of product discontinuance. Even our "Applicatations Buyer", snug and secure with his programs and machine, is shaken by this one.

Every year, new car models come on the scene. Yet, we can still get parts for a '57 Chevy or, for that matter, even a "Model T". But it seems that, once a computer model has been discontinued, it stands alone and unsupported. Resale values crash and it finally lands up in the back row of some computer Flea Market.

This should not be. Computer manufacturers have a responsibility to support their product for more than just the one year of its sales life. As I have tried to point out, obsolescence is a relative term. Old computers are not "dead".If they can do your job and meet your needs, then they are just as good as the new machines.

Now, this brings up another area dealing with "Software Obsolescence". Does the quality of a computer consist solely of its hardware? Or, is the merit of a computer system only dependent on the number of programs available to it? Obviously, it must be a combination of both. But, when the pendulum swings to one extreme or the other, it may mean the death of a particular micro.

The original PET had its hardware and firmware bugs. However, the users found ways around virtually all of them. Commodore's decision to throw

out compatability with their new operating system sounded the death knell for the 8K PET.

When the KIM was in its golden age, software abounded. Yet, when new systems became available, the amount of software flow decreased and finally trickled to zilch. The KIM ceased to be.

COMPUTER MAINSTREAMING

The "Computer Experimenter" can become involved in a concept I call "Computer Mainstreaming". This is a negative feedback mechanism. If, over a several-month period, he sees a decrease in the number of published articles dealing with his particular computer, he immediately panics. He feels that he and his machine are no longer in the mainstream. It is, therefore, time to purchase the "new" leader. This, of course, does lead to fewer articles and the cycle continues.

Therefore, the computer magazines themselves have a hand in shaping product obsolescence.

Once upon a time, there was a company called Data General which, from the beginning of the mini-computer era, produced a hardware product that never changed. Oh sure, there were mods to the system and improvements, but software compatibility was strictly maintained. After many years, their hardware was considered obsolete by many. But a strange thing was happening. People continued to buy Data General. Why? Well, throughout the years, they had amassed such an overwhelming abundance of software that it seemed stupid to use anyone else.

In the Micro world, if one looks closely, one can see two philosophies emerging. Some companies put out a new product what seems to be once a month. Others stay with the old hardware as long as possible. An example of "Rapid Hardware Inc."is, of course, Commodore. Some slow movers are Apple and the AIM-65. Radio Shack, I would consider, is somewhere in the middle. It is a little too early to tell about ATARI, although, if one uses their game product as a basis, then they seem to be very stable.

Texas Instruments is an interesting and unique example. They introduced their system early in the game. Yet, because of a lack of advertising, their high price and lack of software, their computer sat for years on the dealers' shelves. Then, T.I. made its move. The price dropped dramatically, software and firmware became available, and Bill Cosby loved his Pudding Pops and the T.I. Computer. The hardware never changed, but the support made the product respectable, until it suddenly disappeared from the market.

What can we conclude then, when we must plan for obsolescence?

IN SUMMARY

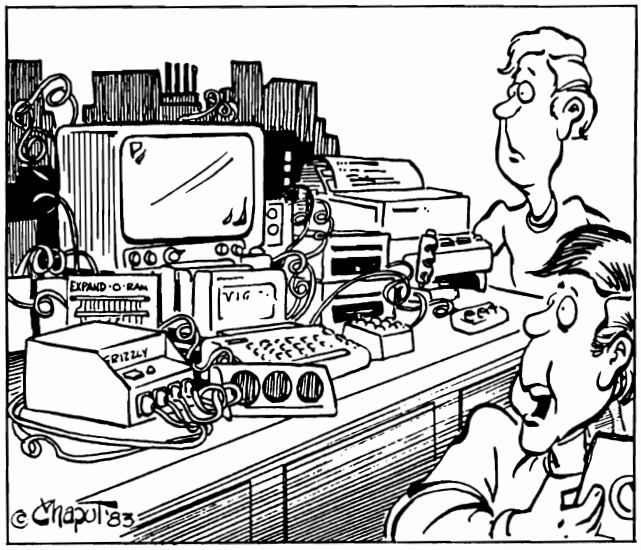
Obsolescence is relative. It exists in the eye and the mind of the buyer.

If you are dissatisfied with the features of your present computer, trade up now. Don't spend money on add-ons which can never make your computer the machine you want it to be.

Don't buy a new computer just because it's new. Examine your needs and your computer's capability to see if they match.

To the "Computer Experimenter" — Means must be found to fulfill your infinite curiosity without breaking your bank account. Writing articles and programs for profit is one way. Another avenue is to review new hardware and software for stores, customers or others who are willing to lend you the new systems. For that end, you get to play with the goodies and they receive valuable information.

The computer manufacturers must retain a parts supply for products they have produced for at least as long a period as other products on the market. This will ensure that both old and new computers can co-exist and provide years of valuable service to their users.



IT USED TO BE A VIC-20.

Swapping and Sharing

By Jim Butterfield, Toronto, Ont.

I must confess that I can't understand the logic of swapping programs.

Sure: you have a spare cat you don't need, and your friend has a shoe polishing kit... go ahead and swap, you'll both benefit. But programs are different.

I can see the situation where each of the two parties have written a program. You've written a telephone list, and I've written a simple game... why not swap?

But even then, it flies in the face of good sense.

You can give away a program — and still have it. If it's yours — or if it's public domain — you incur no loss. Maybe, as the saying goes, he who steals my purse steals trash... but I'm out one purse. On the other hand, he (or she) who gets a copy of my program may also get trash... but I have lost nothing.

Occasionally, I run across someone who has an attractive program. And when I ask, "Is that public domain? May I have a copy?", I get the reply, "What can you swap me for it?" My answer: "Nothing. All my programs are in the TPUG library" So I don't get a copy of the program.

This amazes me. The other person may have dozens — or hundreds — of my programs. But I'm not going to get the new program, because I have nothing to swap.

A few years ago, I received a letter from Oregon, asking if I had any music programs. The writer had bought a commercial package and interface, but didn't have much music. I put together a cassette of all the music I had... a dozen programs or so.

About a month later, a letter came from northern California. It said, "I got a copy of your music programs from XYZ in Oregon. I have some music programs of my own. What programs do you have to swap me for them?" Again, I had to reply, "None — I sent them all to Oregon, you have them all now".

The whole swapping thing makes no sense to me. The name of the game is sharing, not swapping.

Let's look back at the origins of the club. Suppose I — and several other programmers — had said to TPUG, "You don't get programs from us unless you can swap us something equally good". Suppose that TPUG said to its members, "You don't get a program until you submit a program of equal quality". We'd have a pretty weak operation. User groups don't work that way. Thank heavens.

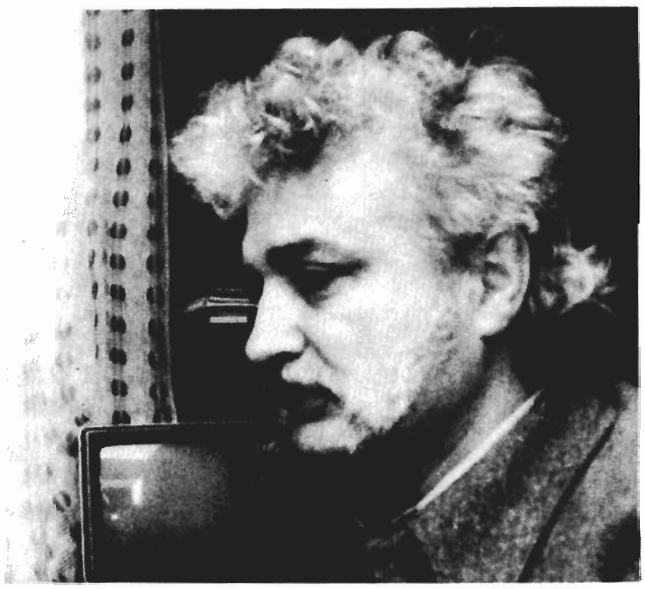
I fear that the swap syndrome encourages program theft.

Some poor beginner who isn't skilled in program writing is coerced by swappers into giving a program as a swap. What is he or she going to give? The pressure is to buy a program and give away a copy. And that's wrong, wrong, wrong.

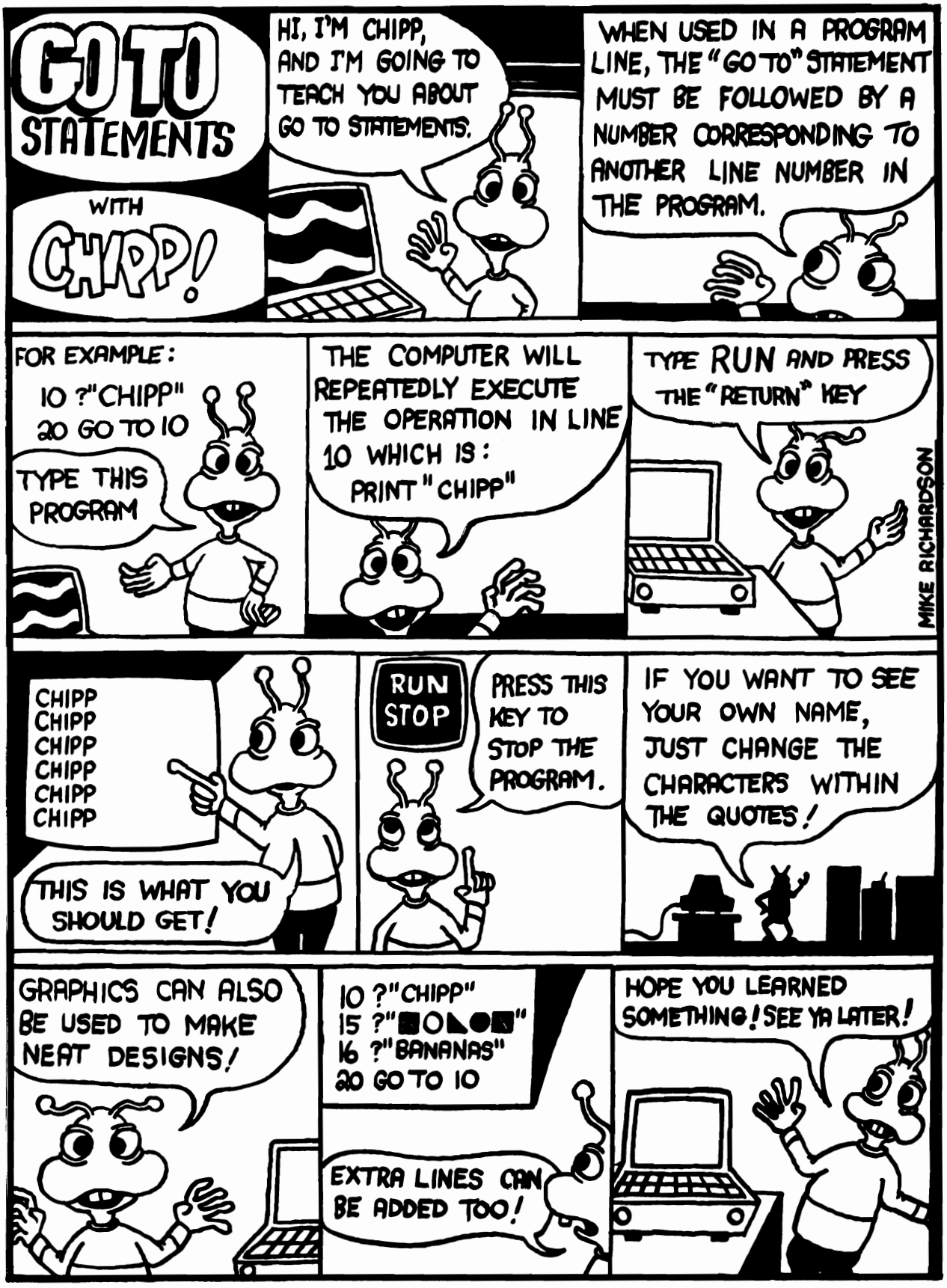
Sometimes I send people programs. I usually refer them to the club, but occasionally I need to send a program or two directly. I don't expect anything in return; in fact, sometimes my return address is not on the package. Some people reply and say, "Thank you", which is OK. On a couple of occasions, people have replied by sending me bootleg copies of commercial programs. They shouldn't do that. I have a feeling that these people have been brainwashed into the 'swapping'thing. They think that they must give something in return ... even if it's illegal. They shouldn't.

Let's get off this swapping bandwagon.

Any programs I have, provided they are not copyrighted or commercial, are freely available to anyone who wants them. They are in the club library, for that matter.



JIM BUTTERFIELD


GO TO STATEMENTS
WITH CHIPP!
HI, I'M CHIPP, AND I'M GOING TO TEACH YOU ABOUT GO TO STATEMENTS.
WHEN USED IN A PROGRAM LINE, THE "GO TO" STATEMENT MUST BE FOLLOWED BY A NUMBER CORRESPONDING TO ANOTHER LINE NUMBER IN THE PROGRAM.
FOR EXAMPLE:
10 ?"CHIPP"
20 GO TO 10
TYPE THIS PROGRAM
THE COMPUTER WILL REPEATEDLY EXECUTE THE OPERATION IN LINE 10 WHICH IS:
PRINT "CHIPP"
TYPE RUN AND PRESS THE "RETURN" KEY
MIKE RICHARDSON
CHIPP
CHIPP
CHIPP
CHIPP
CHIPP
CHIPP
THIS IS WHAT YOU SHOULD GET!
RUN STOP
PRESS THIS KEY TO STOP THE PROGRAM.
IF YOU WANT TO SEE YOUR OWN NAME, JUST CHANGE THE CHARACTERS WITHIN THE QUOTES!
GRAPHICS CAN ALSO BE USED TO MAKE NEAT DESIGNS!
10 ?"CHIPP"
15 ?"■O◣●■"
16 ?"BANANAS"
20 GO TO 10
EXTRA LINES CAN BE ADDED TOO!
HOPE YOU LEARNED SOMETHING! SEE YA LATER!

C64

PAGE

Converting Programs from PET to 64 18
Garry Kiziak, Burlington, Ont.
Only the pure vanilla BASIC old PET programs will work on a 64 without converting. This tells you how to convert the rest.

Those Crazy Screen Control Codes 23
Paul Trachsler, Flesherton, Ont.
How to handle those strange symbols that you find in BASIC program listings.

Some Mixed Mode Graphics Subroutines in BASIC 24
William R. Frenchu, Princeton, N.J.
These BASIC routines allow beginning programmers to mix lower-case, upper-case, graphic characters and bit mapped graphics on their screens.

Painting 28
Dr. Efraim Halfon, Burlington, Ont.
The example of how to fill a circle with color is used to demonstrate how to fill any geometric figure with color.

Programmable Characters 30
Steven Darnold, Alexandra, New Zealand
How to change any or all of the characters on the 64 into any special characters that you desire.

Programming the Commodore 64 Function Keys 32
Paul Thompson & Ron Radko, Toronto, Ont.
This article gives the directions for using the program that allows you to assign any string of characters that you wish to the C-64 function keys.

Dvorak Keyboard 36
William R. Frenchu, Princeton, N.J.
In the 1920's Dvorak designed a keyboard that is quicker and easier to use than the standard QWERTY. Here it is if you would like to have it on your 64.

Menu Selection With A Joystick 38
Alfred J. Bruey, Jackson, MI
This article will not only tell you how to do a menu selection with a joystick; it will give you insight into programming for a joystick itself.

Speech Synthesis of C64 40
Greg Halley, Silver Spring, MD
Yes, your computer really can be made to talk. It is easy. And it is fun! This is a review of a relatively inexpensive software package that does the job.

Creating Sprites on the C64 42
David Bradley, Toronto, Ont.
It is not as difficult as you might think if you follow these step-by-step procedures.

Converting Programs from PET to 64

by Garry Kiziak, Burlington, Ont.

This program is on The Best Programs Disk

Many owners of the new Commodore 64 will have access to a large number of programs written originally for the PET computer. It is natural for these people to ask, "What is involved in converting these programs so that they will run on the 64?" This article will attempt to detail some of the steps involved, and hopefully make the conversion somewhat easier. I will only be discussing conversions involving 2.0 and 4.0 ROM PETS. Those interested in converting programs from 1.0 ROM PETS should be able to make the additional changes necessary.

In many cases, a PET program will run immediately on a 64. In some cases, a few minor changes will make the program workable. In a few cases, major surgery will be required, and in some instances, unless you are heavily into machine language, the conversion will be impossible. The type of conversion required will depend on the make-up of the original program.

As I said above, some programs will run immediately on the 64. These programs will be written entirely in BASIC and will not make use of the commands POKE, PEEK, WAIT, SYS and USR. The easiest way to determine if a program falls in this category is to simply load the program into the 64 and run it. If it works, great. Otherwise, read on.

Note: All BASIC programs for the PET will load into the 64 correctly. This may seem surprising since a PET program is stored in memory starting at location 1025 while 64 programs normally start at 2049. Such loads are successful because of a relocation feature incorporated into the Commodore 64 (and also the VIC 20) computer. These computers will automatically load a program at the START OF BASIC (wherever that happens to be), unless told to do otherwise (see your manual to see how to tell it to do otherwise).

Editor's Note: Details for loading C64 programs to a PET are on page 23.

I should point out that I have had some difficulty loading programs that were saved on a PET with 1.0 ROMS. Such programs do list but the first line is usually mangled. This can be fixed up by deleting that first line and re-typing it or by a few simple pokes.

THE SIMPLEST CONVERSION

Of the programs that do require conversion, the simplest to fix are the ones that do not use the SYS or USR commands. They may use the POKE, PEEK or WAIT commands, but these can usually be fixed up by changing an appropriate address and possibly a corresponding numeric value.

For example, POKE 59468,14 is a command frequently found in PET programs to convert the screen display to lower case. If this command is executed on the 64, nothing drastic will happen but lower case is definitely not displayed. The correct command on the 64 is POKE 53272,23. Similarly, all POKE 59468,12 statements will have to be changed to POKE 53272,21. (This converts the screen display to upper case and graphics.)

The majority of "fixes" can be achieved in this manner, i.e.:

i) Find the address on the PET that is causing a problem;
ii) Find the corresponding address on the 64;
iii) Make all changes involving that address.

What is needed, then, is a list of addresses for the PET that can cause problems and a list of the corresponding addresses for the 64.

Actually, with a little more work, we can even do better. Ideally, a program should be able to run on any machine — PET with 2.0 ROMS, PET with 4.0 ROMS, and the Commodore 64.

This can be achieved for the upper/lower case conversion above in the following way.

Assume first that the program is running on a PET. Somehow have the computer execute the following commands:

```
3000 TEXT = 59468: REM Address to be poked for upper/lower case
3010 UC = 12: REM Value to be poked for upper case
3020 LC = 14: REM Value to be poked for lower case
```

On the other hand, if the program is running on a 64, have it execute the following:

```
3100 TEXT = 53272
3110 UC = 21
3120 LC = 23
```

Now change all POKE 59468,12 statements to POKE TEXT, UC and all POKE 59468,14 statements to POKE TEXT, LC. After these

changes are made, the correct case will be displayed regardless of which computer the program is running on. If all other problem addresses can be fixed up in this manner, then we are well on our way to converting the program to work on all three computers.

WHICH COMPUTER ARE YOU?

The first task, then, is to somehow identify what type of computer a program is running on.

There already is a standard technique for identifying whether a PET has 2.0 ROMS or 4.0 ROMS; namely,

```
110 IF PEEK (50003)  =  160 THEN ... : REM 4.0 ROMS
120 IF PEEK (50003)  =  1 THEN ... : REM 2.0 ROMS
```

PEEKing location 50003 on a 64 will usually yield zero. I say "usually" because 50003 is a RAM location on the 64 and is normally unused. However, machine language routines can be placed in that area and so you cannot be 100% sure what location 50003 will contain. The sequence below will get around this problem, and will identify the type of computer correctly without destroying any machine code already there.

```
100X = PEEK (50003): POKE 50003 , 0: Y = PEEK (50003)
110 IF Y = 160 THEN COMP$ = "4.0" : REM 4.0 ROMS
120 IF Y = 1 THEN COMP$ = "2.0": REM 2.0 ROMS
130 IF Y = 0 THEN POKE 50003,X: COMP$ = "64": REM COMMODORE 64
```

The statement POKE 50003,0 in line 100 has absolutely no effect on 2.0 PETS or 4.0 PETS since location 50003 is in ROM. On the 64, however, it puts a zero into that RAM location. Notice that the original value in location 50003 is saved by the statement X = PEEK (50003) and restored again in line 130 if the computer is identified as being a 64. Note the use of the variable COMP$ to identify the type of computer just in case it is needed again later in the program.

Now the conversion process should be clear. It should include the following:

1. At the beginning of the program, jump to a sub-routine that identifies the type of computer that the program is running on.

2. In that sub-routine, initialize a set of standard variables (such as TEXT, LC, UC, etc.) to the correct values for that computer.

3. Change all references to numerical addresses or values to the corresponding variables.

Here is a sample initialization routine.

```
10 GOSUB 60000
20 REM MAIN PROGRAM
21 END
60000 X = PEEK(50003): POKE 50003,0:Y = PEEK(50003)
60010 REM INITIALIZE VARIABLES COMMON TO 2.0 & 4.0 PETS
60020 TEXT = 59468:UC = 12:LC = 14: SCREEN = 32768:HIV = 144
60030 NUMCHAR = 158:KEY = 151: NOKEY = 255
60040 IF Y< >1 THEN 60100
60050 REM INITIALIZE VARIABLES PECULIAR TO 2.0 PETS
60060 COMP$ = "2.0" :ENA = 46:DIS = 49
60070 RETURN
60100 IF Y< >160 THEN 60200
60110 REM INITIALIZE VARIABLES PECULIAR TO 4.0 PETS
60120 COMP$ = "4.0" :ENA = 85:DIS = 88
60130 RETURN
60200 IF Y< >0 THEN 60300
60210 REM INITIALIZE VARIABLES PECULIAR TO THE 64
60220 COMP$ = "64" :TEXT = 53272:UC = 21:LC = 23:SCREEN = 1024:HIV = 788
60230 NUMCHAR = 198:KEY = 203:NOKEY = 64:ENA = 49:DIS = 52
60240 POKE 50003,X:RETURN
60300 PRINT "I DON'T RECOGNIZE THIS COMPUTER" :END
```

The variables SCREEN, NUMCHAR, etc., will be explained shortly.

MORE PROBLEM AREAS

Upper/lower case conversion is certainly not the only problem area. Another potential one is the screen.

1. THE SCREEN

On the PET, the screen is found in memory locations 32768-33767. On the 64, it is found in locations 1024-2023.

If all output to the screen is obtained through the use of PRINT statements, then absolutely no problem will arise. If, however, the output is POKEd to the screen, then changes will be required.

These changes are best achieved by assigning a value to the base address of the screen and then using an appropriate offset from that base.

For example, the base address of the screen on the PET is 32768, while on the 64 it is 1024. Therefore, the first thing to do is to assign values to the standard variable SCREEN as follows:

```
SCREEN = 32768 if on a PET
SCREEN = 1024 if on a Commodore 64
```

i) **Poking a single value onto the screen.** A statement of the form POKE 32956,61 on a PET has to be changed as follows:

First, calculate the offset.

```
Offset = 32956 - 32768 = 188
```

Then, change POKE 32956,61 to: POKE SCREEN + 188,61

The resulting statement will work on either a PET or a 64 (assuming SCREEN has been properly initialized).

Notice that the 61 does not have to be changed as these values are the same for both PETS and the 64.

ii) Poking within a loop. The following is a typical PET routine that POKES a border of reversed diamonds around the screen.

```
100 FOR I =32768 TO 32807:POKE I,218 :NEXT
110 FOR I =32847 TO 33767 STEP 40:POKE I,218:NEXT
120 FOR I =33766 TO 33328 STEP -1:POKE I,218: NEXT
130 FOR I=33688 TO 32768 STEP -40:POKE I, 218: NEXT
```

This can be changed to work on both PETS and 64 by changing each screen address as above.

```
100 FOR I =SCREEN TO SCREEN +39:POKE 1,218 : NEXT
110 FOR I = SCREEN +79 TO SCREEN + 999 STEP 40: POKE
I,218 : NEXT
120 FOR I = SCREEN + 998 TO SCREEN + 990 STEP -1: POKE
I,218 : NEXT
130 FOR I = SCREEN +920 TO SCREEN STEP -40: POKE
I,218 : NEXT
```

Or better yet:

```
100 FOR I = 0 TO 39: POKE SCREEN + I,218 : NEXT
110 FOR I = 1 TO 24: POKE SCREEN + 39 +I * 40,218 : NEXT
120 FOR I = 38 TO 0 STEP -1: POKE SCREEN +960 +1,218 :
NEXT
130 FOR I = 23 TO 1 STEP -1: POKE SCREEN + I *40,218 :
NEXT
```

2. CLEARING THE KEYBOARD BUFFER

The PET is able to retain up to 10 keystrokes in a buffer, enabling you touch typists to type as fast as you can without losing any keystrokes. This can sometimes add extra unwanted characters to the beginning of an input, so a common technique in PET programming is to clear the keyboard buffer before each input is requested. This can be accomplished in a couple of ways.

```
1) 100 FOR I = 1 TO 10 : GET A$ : NEXT
or
2) 100 POKE 158,0
```

The first method will work as is on the 64. The second method must be changed.

On 2.0 and 4.0 PETS, location 158 always contains the number of characters in the keyboard buffer. On the 64, this value is stored in location 198. Thus, if we assign values to the standard variable NUMCHAR as follows:

```
NUMCHAR = 158 if on a PET
NUMCHAR = 198 if on a 64
```

and change all references to POKE 158,0 to POKE NUMCHAR,0, then the resulting statement will work on both computers.

3. PAUSING UNTIL ANY KEY IS PRESSED

Here again, two techniques are commonly used.

1) 100 GET A$:IFA$ = ""THEN 100 is certainly the simplest and will work on both computers.

2) 100POKE158,0:WAIT158,1:POKE158,0 is another technique and will have to be changed to 100 POKE NUMCHAR,0 : WAIT NUMCHAR,1 : POKE NUMCHAR,0

4. WHICH KEY IS PRESSED

A common technique used on the PET, especially in games, is to PEEK at location 151 to see if a key is being pressed and, if so, which one. Depending on which key is pressed, a certain action is performed. This technique is frequently used in games that use the numeric keypad as a joystick. A sample sequence might be:

```
500 X = PEEK (151)
510 IF X = 255 THEN 1000: REM NO KEYPRESS
520 IF X = 18 THEN 2000: REM 2 KEY IS PRESSED
530 IF X = 50 THEN 3000: REM 8 KEY IS PRESSED
etc.
```

The conversion here is a little more complicated, but is still possible. First, we need to know that location 151 on the PET corresponds to location 203 on the 64. Then assign the following values to the standard variable KEY:

KEY = 151 if on a PET
KEY = 203 if on a 64

Replacing line 500 with 500 X = PEEK(KEY) gives us a start with the conversion.

Another problem occurs with the values stored in location 151 (or 203) when a key is not being pressed. Location 151 on the PET contains 255 while location 203 on the 64 contains 64. This time, we will use the standard variable NOKEY and initialize it as follows:

NOKEY = 255 if on a PET
NOKEY = 64 if on a 64

Line 510 is then replaced with 510 IF X = NOKEY THEN 1000

There are two problems associated with the other keys. First, location 151 will contain a certain value on the 2.0 machines, the same value on the Skinny 40 (i.e., the 9-inch screen) machines, but a different value on the Fat 40 machines. There is no standard way that I am aware of, for distinguishing between a Skinny 40 and a Fat 40 machine. But PEEKing at location 57344 will do as well as any other. On a Skinny 40 you will get a value of 169, while on a Fat 40 you will get a value of 76.

The easiest way to see what value is stored in location 151 is to run the following program segment and press any key that you wish to test.

```
100 KEY = 151 : REM = 203 ON THE 64
110 PRINT PEEK (KEY) : GOTO 110
```

The second problem arises from the fact that the 64 does not have a numeric keypad and using the numbers 2, 4, 6, and 8 to simulate a joystick is unacceptable. It would be much better to use the keys I, J, K, and M or some other suitable arranged set of keys.

My suggestion for getting around this is to first settle on the keys that you wish to use on each machine (they don't have to be the same). Find the values corresponding to these keys by running the short program above and store these values in corresponding standard variables which I like to designate K1, K2, K3, etc. (for KEY 1, KEY 2, KEY 3, etc.) then lines 520 and 530 can be replaced by the following:

520 IF X = K1 THEN 2000
530 IF X = K2 THEN 3000
ect.

5. DISABLING THE STOP KEY

The stop key on the PET can be disabled by altering the Hardware Interrupt Vector. For example:

POKE 144,49 for 2.0 PETS
POKE 144,88 for 4.0 PETS

will disable the stop key (and the time clock as well).

The corresponding command on the 64 is POKE 788,52.

To enable the stop key again:

POKE 144,46 for 2.0 PETS
POKE 144,85 for 4.0 PETS
and POKE 788,49 for the 64

These can be replaced by POKE HIV,DIS to disable the stop key and POKE HIV,ENA to enable the stop key, after appropriately initializing the variables HIV, DIS and ENA. On the 64, the program can still be stopped by pressing the RUN/STOP and RESTORE keys simultaneously, but this will prevent stoppage of a program due to accidentally pressing the STOP key.

A good question to ask is "How do you know what value is to be stored in these locations?" The PET program actually tells you the location to poke as well as the value, but the value to be poked on the 64 is usually different (i.e., disabling the stop key above or converting to upper/lower case). A memory map will tell you what location to poke on the 64, but it will not tell you what value to poke it with. A good start is to PEEK that location from direct mode and make note of the value. Do this for all three machines and it will tell you the "normal" state of that location. For example, PEEKing location 144 on 2.0 PETS and 4.0 PETS yields 46 and 85 respectively. PEEKing at 788 on the 64 yields 49.

Observing that the values to disable the stop key on the 2.0 and 4.0 PETS are each three more than the "normal" value. A good start to finding the correct value on the 64 is to add 3 to the normal value of 49, obtaining 52. This process will work for more than 90% of the problem values. It is that last five to 10% that makes the conversion challenging.

It would be impossible to list all problem locations and their "fixes" here. (I will list what I feel are the more common ones below.) Instead, I have attempted to give you a feeling for how the conversion should proceed. The proper tools that are required are the excellent memory maps (both zero page and ROM routines) published for all three computers in COMPUTE by Jim Butterfield. Another excellent source is the book "Programming The PET/CBM" by Raeto Collin West, and I am sure there are others. (Would you believe The Best Of The Torpet-ed.) See page 292 .

SOME OF THE MORE COMMON PROBLEM LOCATIONS

Location on 2.0 PET	Location on 4.0 PET	Location on 64	Suggested Name	Description
40-41	40-41	43-44	SBAS	Start of BASIC text
42-43	42-43	45-46	SVAR	Start of variables
44-45	44-45	47-48	SARR	Start of arrays
46-47	46-47	49-50	EARR	End of arrays
52-53	52-53	55-56	THEM	Top of memory
144-144	144-145	788-789	HIV	Hardware Interrupt Vector
151	151	203	KEY	Which key is pressed
158	158	198	NUMCHAR	Number of characters in keyboard buffer
159	159	199	RVS	Screen reverse flag
167	167	204	CRSR	Flag for flashing cursor in GET statements
196	196	209	SLO	Pointer to screen
197	197	210	SHI	(low/high format)
198	198	211	CH	Horizontal position of cursor
216	216	214	CV	Vertical position of cursor
623	623	631	BUFF	Start of keyboard buffer
634	634	--	--	Start of first cassette buffer
826	826	828	CAS	Start of second cassette buffer
32768	32768	1024	SCREEN	Start of screen memory
59468	59468	53272	TEXT	Poke location for upper/lower case
64721	64790	64738	--	Simulates power on reset
			SPECIAL VALUES	
12	12	21	UC	Upper case
14	14	23	LC	Lower case
255	255	64	NOKEY	Nokey is pressed
46	85	49	ENA	Enable stop key
49	88	52	DIS	Disable stop key

A COUPLE OF CAUTIONS

1. The PET and the 64 only recognize the first two letters of a variable name. When converting a program, you must make certain that the variables already present in the program do not conflict with the standard variables suggested above. If there is a conflict, change whichever you feel is easier.

2. If a program is to be used both on a PET and a 64, then the changes should be made on and saved with a PET computer. The reason for this is that a program saved on a 64 will not load properly on a PET, due to the lack of a relocation feature in the PET, but.....

If the changes are made on a PET, then a utility such as BASIC AID or POWER will be invaluable since you can type in such things as FIND/POKE/ and all lines that contain a POKE statement will be listed, making it easier for you to make the necessary changes and to make certain that you have found all of them.

Similarly, you can type in FIND/SC/ to see if there are any variables in the program that will conflict with the standard variable SCREEN.

PROGRAMS THAT CONTAIN SYS OR USR COMMANDS

These programs will require that you be somewhat familiar with machine language in order for you to be able to make the necessary conversions. Such changes are beyond the scope of this article. However, let me say that these M/L routines themselves fall into a number of categories.

Well, I'll just see your Z-80 and raise you three 6502's.

1. The routine works on the 64 as is (few routines will likely fall in this category).

2. The routine will work with a simple address change (these are frequently ROM routines such as the routine for resetting the entire stack).

3. The routine will work with some minor changes. an example here could be a routine to reverse a portion of the screen. Chances are the only changes necessary would be for the location that determines the base address of the screen. However, if parameters are passed in the calling statement, then the location of certain ROM routines (such as checking for a comma) might have to be changed as well.

4. The routine will require major surgery before it will work. A program like BASIC AID or MICROMON would fall into this category. Such programs should be left to the experienced users.

Other areas that may require major changes are those programs that make use of the BASIC 4.0 disk commands. Some of these can be fixed up easily, but some are extremely difficult, e.g., those that make use of Relative Record files.

Programs that make use of CB2 sound will still run on the 64, but no sound will be produced. The POKEs that the PET uses to produce these sounds will POKE into the ROMS of the 64 and hence do no harm. Once you are familiar with the sound processes on the 64, here is a good place to make use of the variable COMP$. For example, suppose lines 1000 to 1030 in the PET program are used to produce the sound. Leave these lines exactly as they are and add a similar routine for producing sound on the 64, beginning with:

```
1031 IF COMP$◄►"64" then 1040
```

Your sound routine can be placed in lines 1032 to 1039, and the rest of the program should proceed as normal.

LOADING PROGRAMS SAVED ON THE 64 INTO THE PET

As mentioned above, programs saved on the 64 do not load properly into a PET. It is not a difficult procedure to correct this shortcoming, however. Here are the steps.

1. Type a dummy line 0 into the PET, 0 REM will do.

2. Type in POKE 2048,0.

3. LOAD in the program that was saved on the 64 as you normally would.

4. Type in POKE 1025,1 : POKE 1026,8.

You should now be able to LIST the program including the dummy line 0 that you typed in initially. Delete this line by typing in 0 (Return). The process is now complete. You can save the program to cassette or disk. The next time that you load it in to your PET, it will load normally. If you are using a disk, you will notice that the program is 3 blocks longer that the original even though it is the same program. The reason for this is that the Start of Variables pointer did not get changed properly. An experienced programmer can get into the monitor and make the necessary changes without too much difficulty, but the program will operate correctly without making this change.

Those Crazy Screen Control Codes

By Paul Trachsler, Flesherton, Ont.

For some of us, the hardest part of typing in a program listing from a book or magazine is trying to figure out how to type in some of the control symbols or codes used in the program. Control symbols or codes are like traffic signals for the computer, in that they give the program information about where and how to display information on the screen. Because of this, control symbols or codes are always typed in from the "quote mode" that is from within quotation marks (otherwise a clear screen code will leave you looking at a blank screen!).

Different publishers of books and magazines do not go out of their way to make things difficult for us but the use of different formats for displaying these symbols can be doubly confusing. The best and easiest solution is to obtain a working copy of the program from the publisher or author (see page 272). Failing this, some publications include a table explaining what the symbols stand for and how to type them in. The following general table

will explain what these codes do, what they look like and how to type them.

WHEN YOU SEE	WHICH MEANS	THEN PRESS
clear, clr or	clear the screen	SHIFT CLR/HOME
home or	place cursor at top left corner of screen	HOME
up crsr, up or	cursor up	SHIFT CURSOR (up/down)
down crsr, down or	cursor down	CURSOR (up/down)
left crsr, left or	cursor left	SHIFT CURSOR (right/left)
right crsr, right, or	cursor right	CURSOR (right/left)
reverse, rvs or	reverse lettering on	CTRL 9
reverse off, off or	reverse lettering off	CTRL 0

Colors may be obtained by using the Control key and the appropriate color key.

Some Mixed Mode Graphics Subroutines in BASIC

By William R. Frenchu, Princeton, N.J.

This program is on The Best Programs Disk

The C-64 Programmer's Reference Guide mentions (but doesn't give examples of) using the raster scan interrupt to mix bit-mapped and text modes on the screen. For machine language programmers this presents few problems, but for users with only BASIC experience it would often be advantageous to be able to present high resolution plots with text mode labels quickly and easily. This method would also side-step another difficulty of using the raster interrupt method to mix graphics, that is, the inability to mix graphic types on the same raster. For applications where a vertical axis must be plotted, this restriction eliminates much of the screen from containing text. The BASIC routines presented here enable beginning programmers to mix upper case (and graphic characters), lower case and bit-mapped graphics on the same line and screen with minimal effort.

ROUTINE 1 (Lines 10000-10290)

Routine 1 prints a string (including cursor, case and reverse control characters) starting at a given position while in the high resolution mode. It's capable of using any character set pointed to by variable B1. This allows the use of user defined character sets in addition to the ROM set defined by the C-64. User sets should be stored in the same order as the ROM set (64 character blocks of upper case, graphics, reversed upper case, reversed graphics, lower case, shifted lower case, reversed lower case, shifted reversed lower case) in order for the case and reverse control keys to retain their expected function. Using the ROM set requires the routine to temporarily disable the interrupts and "switch in" the ROM while getting the character definitions. For this reason the stop key is disabled whenever a string is actually being printed. When a user character set is accessed, this restriction does not apply and the POKEs to locations 1 and 56334 may be eliminated.

Routine 1 makes use of several flags and variables set by the user. The lower case flag (L) is set to "1" when a string is to be printed using the

lower case set and "0" otherwise. The reverse flag (R) performs a similar function for reverse on/off. Case and reverse can also be changed at any time from within a string by including the following special characters:

Change to upper case	(reversed shift "n")
Change to lower case	CTRL-N
Turn reverse on	CTRL-9
Turn reverse off	CTRL-0

Thus, any string may contain characters from both upper and lower case character sets. The programmable function keys could be easily added to the decoding section (lines 10030-10130) to produce frequently used effects such as tabs, super- and sub-scripting.

The position of the printed string is determined by the variables X and Y. These refer to the usual cursor positions, X from 0 to 39 and Y from 0 to 24. B$ is the string to be printed. Strings may be up to 255 characters in length and the subroutine will automatically continue a string that is too long on the next line. Strings printed at the same X,Y positions will be "overstruck" if flag "O" is equal to "1" and replaced if "O" is equal to "0".

ROUTINE 2 (Lines 20000-20080)

This routine supports user input to the program. As it calls Routine 1, the X,Y,R and L variables retain the same functions as above, but now B$ is the prompt string for the "input statement". Input is returned as a string, I$, and must be converted to a numerical value with the VAL function if necessary. The special characters from the decoding section of Routine 1 are returned in I$, but are not echoed on the screen during input. The final flag is BL which, when set to "1", causes the prompt and input strings to be blanked after a return is received. If BL is "0" the prompt and input strings will remain on the screen. This routine also allows use of the DELETE key to correct errors in input. Do not use cursor controls to correct errors as these keys are returned in the I$. The INSERT key is not supported.

ROUTINE 3 (Lines 30000-30400)

Routine 3 draws a line from pixel coordinates X1,Y1 to X2,Y2 where X ranges from 0 to 319 and Y from 0 to 199. The Reference Guide recommends always using points two pixels wide to decrease "Chroma Noise". This could be a simple modification (or two lines could be drawn side by side) but since it decreases resolution it hasn't been implemented here.

ROUTINE 4 (Lines 16 and 1000)

This is a "one line" routine that turns on a given pixel X,Y as above. It is called by Routine 3. Line 16 sets up a table containing the powers of two from 7 to 0 for use by line 1000. This was done because exponentiation on the C-64 is very slow. (Integer exponentiation is even slower than floating point!) It was found that calculating the powers of two for each X,Y pair more than doubled the time necessary for plotting.

ROUTINE 5 (Line 1001)

The final routine turns off a given X,Y pixel. If called instead of Routine 4 (by Routine 3) lines may be "unplotted", too.

```
0 REM ***CHANGE SCREEN COLOR***
1 REM ***PRINT WAIT MESSAGE ***
2 POKE 53280,11:POKE 53281,0
4 PRINT"[GRAY2,CLEAR,RIGHT7,DOWN8]CLEARING HIGH RES SCREEN . . "
6 PRINT"[DOWN2,RIGHT8]PLEASE WAIT 35 SECONDS"
7 PRINT"[DOWN11,RIGHT,RVS]♦♣ COMMODORE-64 HI-RESOLUTION DEMO ♠♥"
8 REM *** CLEAR HI-RES SCREEN ***
10 FOR I=8192 TO 16192:POKE I,0:NEXT
13 REM *** SET UP POWERS OF 2 TABLE ***
14 REM ***    FOR ROUTINES 4 & 5    ***
16 FOR I=0 TO 7:P(I)=2↑(7-I):P1(I)=255-P(I):NEXT
17 REM ***   START HI-RES MODE AND   ***
18 REM *** SET HI-RES SCREEN AT 8192 ***
20 PRINT"[CLEAR]":POKE 53265,PEEK(53265)OR 32:POKE 53272,PEEK(53272)OR 8
25 REM ***    SET HI-RES COLORS    ***
26 REM ***UPPER NYBBLE FOR "1" BITS***
27 REM ***LOWER NYBBLE FOR "0" BITS***
30 FOR I=1024 TO 2023:POKE I,192:NEXT
100 REM ***PRINT STRINGS USING***
101 REM ***      ROUTINE 1      ***
102 L=0:R=0:X=0:Y=13:B1=53248:O=1:B$="-2↓[LEFT2,UP,LEFT]| ":GOSUB 10000
103 X=20:Y=13:B$="0":GOSUB 10000
104 X=37:Y=13:B$="[UP,RIGHT2] [LEFT3,DOWN]+2↓":GOSUB 10000
105 X=19:Y=3:B$="—+1":GOSUB 10000
106 X=19:Y=21:B$="¯-1":GOSUB 10000
107 R=1:X=1:Y=23:B$="♦♣ C[TEXT]OMMODORE-64  I-_ESOLUTION ¯EMO [GRAPHIC]♠♥"
   :GOSUB 10000
108 REM *** PRINT AXIS USING ***
```

```
100 REM ***     ROUTINE 3       ***
110 X1=0:Y1=100:X2=319:Y2=100:GOSUB 30000
113 REM *** PRINT AXIS USING ***
114 REM ***     ROUTINE 4       ***
115 FOR Y=25 TO 174:X=158:GOSUB 1000:X=157:GOSUB 1000:NEXT
116 REM *** GET USER INPUT WITH ***
117 REM ***       ROUTINE 2       ***
120 BL=1:X=0:Y=0:R=0:B$="INPUT PERIOD ? ":GOSUB 20000:J=VAL(I$)
125 REM *** PLOT SINE CURVE ***
126 REM *** USING ROUTINE 4 ***
130 FOR X=0 TO 319:Z=SIN((X-158)/25*J):Y=INT(100-70*Z*Z*Z):GOSUB 1000:NEXT
131 REM *** LABEL PLOT WITH INPUT ***
132 REM ***     USING ROUTINE 1     ***
133 L=1:R=0:X=4:Y=1:B$="Y=[GRAPHIC JS[TEXT JIN[UP JS[DOWN J[ ":GOSUB 10000
134 B$=I$:GOSUB 10000:B$="*X)":GOSUB 10000
137 REM *** PAUSE LOOP: WHEN "A"   ***
138 REM *** IS RECEIVED GO BACK    ***
139 REM ***    TO STANDARD MODE     ***
140 REM ***       AND STOP          ***
145 GET A$:IF A$="" THEN 145
150 POKE 53265,PEEK(53265)AND 223:PRINT"[CLEAR]";:POKE 53272,PEEK(53272)AND 21
   :END
982 :
984 :
990 REM ***      ROUTINES 4 & 5     ***
992 REM ***     FOR PLOTTING AND    ***
994 REM ***    UNPLOTTING POINTS    ***
996 REM *** SEE REF. GUIDE PG 125 ***
997 :
1000 B=INT(Y/8)*320+INT(X/8)*8+(Y AND 7)+8192:POKE B,PEEK(B)OR P(X AND 7)
   :RETURN
1001 B=INT(Y/8)*320+INT(X/8)*8+(Y AND 7)+8192:POKE B,PEEK(B)AND P1(X AND 7)
   :RETURN
9980 :
9982 :
9990 REM *** ROUTINE 1: FOR PRINTING ***
9992 REM ***    STRINGS IN HI- RES     ***
9993 :
9994 REM *** DISABLE INTERRUPTS & ***
9996 REM ***  SWITCH IN CHAR ROM   ***
9998 REM *** CALCULATE CHAR BASE   ***
10000 POKE 56334,PEEK(56334)AND 254:POKE 1,PEEK(1)AND 251:B2=B1+R*1024+L*2048
10010 REM ***  GET A CHARACTER   ***
10012 REM *** FROM INPUT STRING ***
10020 FOR I=1 TO LEN(B$):C=ASC(MID$(B$,I,1))
10026 REM *** SPECIAL CHARACTERS ***
10028 REM ***  DECODING SECTION   ***
10030 IF C=145 THEN Y=Y-1:NEXT:RETURN:REM ** CURSOR UP **
10040 IF C=17 THEN Y=Y+1:NEXT:RETURN:REM ** CURSOR DOWN **
10050 IF C=29 THEN X=X+1:NEXT:RETURN:REM ** CURSOR RIGHT **
10060 IF C=157 THEN X=X-1:NEXT:RETURN:REM ** CURSOR LEFT **
10070 IF C=18 THEN R=1:B2=B1+1024+L*2048:NEXT:RETURN:REM ** REVERSE ON **
10080 IF C=146 THEN R=0:B2=B1+L*2048:NEXT:RETURN:REM ** REVERSE OFF **
10090 IF C=19 THEN X=0:Y=0:NEXT:RETURN:REM ** CURSOR HOME **
10100 IF C=14 THEN L=1:B2=B1+R*1024+2048:NEXT:RETURN
   :REM ** START LOWER CASE **
10120 IF C=142 THEN L=0:B2=B1+R*1024:NEXT:RETURN:REM ** STOP LOWER CASE **
10130 IF C=255 THEN C=126:REM ** "#" IS SPECIAL CASE **
10132 REM ***  TRANSLATE CHR$ CODES   ***
10134 REM *** TO SCREEN CODES: CHARS ***
10136 REM *** PATTERNS IN ROM STORED ***
10138 REM ***     BY SCREEN CODE      ***
10140 ON C/32+1 GOTO 10150,10200,10170,10160,10150,10170,10190,10170
10150 C=32:GOTO 10200
10160 C=C-32:GOTO 10200
10170 C=C-64:GOTO 10200
10180 C=C-96:GOTO 10200
10190 C=C-128
10192 REM *** CALCULATE STARTING POS ***
10194 REM ***   FOR STRING AND CHAR   ***
10196 REM ***       DEFINITION        ***
10200 Z=Y*320+X*8+8192:C=C*8+B2
10220 REM *** POKE DEFINITION INTO ***
10222 REM ***   HI-RES LOCATION     ***
```

```
10240 FOR J=0 TO 7:POKE Z+J,(0*PEEK(Z+J))OR PEEK(C+J):NEXT:X=X+1:NEXT
10260 REM *** RE-ENABLE INTERRUPTS ***
10262 REM ***  AND SWITCH OUT ROM  ***
10290 POKE 1,PEEK(1)OR 4:POKE 56334,PEEK(56334)OR 1:RETURN
19880 :
19882 :
19900 REM *** ROUTINE 2-USER INPUT    ***
19901 :
19902 REM *** INITIALIZE INPUT STRING ***
19904 REM *** SAVE START POSITION AND ***
19906 REM ***    LENGTH OF PROMPT     ***
20000 I$="":HX=X:HY=Y:HB=LEN(B$):GOSUB 10000
20008 REM *** GET A CHAR ***
20010 GET B$:IF B$="" THEN 20010
20014 REM *** CHECK FOR SPECIAL CHARS ***
20016 REM ***    ONLY FIRST TWO ARE   ***
20018 REM ***DIFFERENT FROM ROUTINE 1 ***
20020 IF B$=CHR$(13) THEN 20070:REM *** RETURN ***
20030 IF B$=CHR$(20) THEN 20045:REM *** DELETE ***
20031 IF B$=CHR$(145) THEN 20041
20032 IF B$=CHR$(17) THEN 20041
20033 IF B$=CHR$(29) THEN 20041
20034 IF B$=CHR$(157) THEN 20041
20035 IF B$=CHR$(18) THEN 20041
20036 IF B$=CHR$(146) THEN 20041
20037 IF B$=CHR$(19) THEN 20041
20038 IF B$=CHR$(14) THEN 20041
20039 IF B$=CHR$(142) THEN 20041
20040 REM *** ECHO CHARACTER ***
20041 GOSUB 10000
20042 I$=I$+B$:GOTO 20010
20043 REM *** DELETE KEY: DONT DELETE ***
20044 REM ***     IF NOTHING THERE     ***
20045 IF LEN(I$)=0 THEN 20010
20046 REM *** MOVE BACK AND BLANK ONE ***
20047 REM ***    CHAR; UPDATE INPUT    ***
20050 X=X-1:Z=Y*320+X*8+8192:FOR I=0 TO 7:POKE Z+I,0:NEXT
   :I$=LEFT$(I$,LEN(I$)-1)
20058 REM *** GET NEXT CHAR ***
20060 GOTO 20010
20066 REM *** BLANK INPUT IF DESIRED ***
20068 REM ***       ELSE RETURN       ***
20070 IF BL=0 THEN RETURN
20074 REM *** STARTING ADDRESS FOR  ***
20076 REM *** BLANKING AND BLANKING ***
20078 REM ***          LOOP         ***
20080 Z=320*HY+8*HX+8192:FOR I=0 TO (HB+LEN(I$))*8:POKE Z+I,0:NEXT:RETURN
29880 :
29882 :
29900 REM ***  ROUTINE 3     ***
29902 REM *** DRAW A LINE    ***
29903 :
29904 REM ***  CALCULATE SLOPE AND  ***
29906 REM ***   DECIDE WHETHER TO   ***
29908 REM ***    INCREMENT X OR Y   ***
30000 XD=X1-X2:YD=Y1-Y2
30010 IF XD=0 THEN 30200
30020 IF YD=0 THEN 30300
30030 M=YD/XD:S=Y1-M*X1
30040 IF ABS(M)<=.5 THEN 30400
30050 M=XD/YD:S=X1-M*Y1
30060 REM *** CALCULATE X  ***
30062 REM *** STEP ALONG Y ***
30100 FOR Y=Y1 TO Y2 STEP SGN(Y2-Y1):X=M*Y+S:GOSUB 1000:NEXT:RETURN
30160 REM *** VERTICAL LINE ***
30162 REM ***  STEP ALONG Y ***
30200 X=X1:FOR Y=Y1 TO Y2 STEP SGN(Y2-Y1):GOSUB 1000:NEXT:RETURN
30260 REM *** HORIZONTAL LINE ***
30262 REM ***   STEP ALONG X  ***
30300 Y=Y1:FOR X=X1 TO X2 STEP SGN(X2-X1):GOSUB 1000:NEXT:RETURN
30360 REM *** CALCULATE Y  ***
30362 REM *** STEP ALONG X ***
30400 FOR X=X1 TO X2 STEP SGN(X2-X1):Y=M*X+S:GOSUB 1000:NEXT:RETURN
```

Painting

By Dr. Efraim Halfon, Burlington, Ont.

This program is on The Best Programs Disk

High-resolution graphics in the Commodore 64 have a resolution of 320 pixels horizontally and 200 pixels vertically. This graphics mode, however, only allows two colors, a foreground and a background. Up to four colors can be obtained by losing some horizontal resolution, from 320 to 160 pixels, and using the multi-color bit mapping graphics. In this graphic mode, the colors are determined by the bit pattern in each byte. For example, an 11 will produce the foreground color, 01 a second color, 10 a third, and 00 the screen color.

The C64 BASIC at present does not support any extended graphics commands to draw lines and circles; all drawings in high resolution must be programmed using mathematical equations. In this article, I present a method to draw circles and fill them with color, i.e., " paint" them. This method is then extended to "paint" circles within circles. This example is part of the more general problem of being able to draw a geometrical figure anywhere in the screen and painting it by turning on the bits in the bytes in the appropriate sequence, e.g., 01010101, as required by the multi-color bit mapping graphics mode. In fact, if for example the bits in all rows are not aligned, i.e.

```
010101010101010101010101
010101010101010101010101
101010101010101010101010
010101010101010101010101
```

then, instead of a uniform painting, we have lines of different colors. In the above example row, three have a different color from the others, depending on the bit pattern on that line.

The Algorithm (Table 1) allows this alignment to take place and to draw circles in any of the chosen colors. The algorithm can then be extended to any closed two-dimensional geometrical figure. The principles in the algorithm are to draw the perimeter of the circle and then paint inside horizontally right to left. Since this algorithm in BASIC is fairly slow I suggest saving the high resolution screen on disk or tape for quick retrieval when necessary.

The equation of a circle centered on the origin of a Cartesian graph is

$$x^2 + y^2 = r^2$$

where x and y are the Cartesian coordinates and r is the radius length. Given the allowable resolution, x has a range of 0 to 319 and y a domain of 0 to 199.

The equation of a circle anywhere on the screen is

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

where x and y are the coordinates of the center. The circle circumference can be drawn with the algorithm starting at line 170, i.e.

$$\left.\begin{matrix} x_l \\ x_r \end{matrix}\right\} = x_c \pm \text{SQR}\,(r^2 - (y - y_c)^2)$$

$$\text{for } y = y_c - r \text{ to } y_c + r \ .$$

I make y the independent variable, instead of x, because the bit color mapping alignment needed for the multi color mode is horizontal rather than vertical. Thus we draw the circumference line by line. X is the coordinate of the left semicircle and x the coordinate of the right semicircle.

Painting of the circle is obtained by drawing a line from X to x (line 230), i.e. right to left. As mentioned before, the important point is that all 0 and 1 bits be aligned vertically to obtain a uniform color. Thus, if you choose so, x has to be 01 for all y's. Algorithmically this is fairly simple, the x coordinate must always be even, odd for 10. Then a line is drawn between x and x with STEP 2, i.e., we turn on the screen bits 1 separated by the default bit 0, or the desired 010101 combination. Lines 190-210 provide this alignment. The test of oddness, or even-ness, is made by dividing the integer portion of x by 2 and then multiplying the integer quotient x by 2. If x and x are the same, then the number is even, else it is odd. If x is not even, or odd as we wish, we subtract 1 to make the coordinate odd, or even, then we draw the line between x and x, i.e., we "paint" each line.

The circle is then uniformly painted in the chosen color (line 450 sets the colors to blue and red in this example), and at any chosen coordinate xc and yc for any given radius r.

CIRCLE WITHIN A CIRCLE

To draw a circle within a circle of a different color, the extension is straight forward. The center coordinates are the same x and y but the radius is smaller. A different color is chosen, here red, and the bit pattern is not 0101010101, but 1010101010. The rest of the procedure is the same.

To draw a circle within a circle with screen color, bits 00, a test must be performed (line 220) to see which colors to paint where. The outside and inside circles are drawn together, so as not to turn on the bits in the inside circle. Thus, the inside circle remains in the screen, background color.

Once the algorithm presented here in BASIC is executed, the result is three concentric circles; the outside is blue, the middle white and the internal red. The circles' location and radius are determined in line 400. For faster execution, a smaller radius may be chosen. The high resolution screen is located in memory starting at location 8192 decimal (line 410). As mentioned before execution in BASIC is slow and if redrawing during a program execution is necessary the easiest solution is to save the screen on disk, alternatively the algorithm can be reprogrammed in machine language. A circle plotted in high-resolution multicolor bit mapping mode does not look as round and smooth as one plotted with standard two-color high-resolution graphics, but this compromise was chosen by the designers of the C-64 to have four available colors.

TABLE 1: DESCRIPTION OF THE BASIC PROGRAM LINE BY LINE

Line 100: Go to line 400 to initialize parameters and then start plotting from line 160.
Lines 130-150: Subroutine to turn on bits in high resolution mode. See C-64 programmers reference manual, high resolution section.
Line 160: Main program, set background color to white.
Line 170: Set outside limits of circle.
Lines 180 - 190: Set x coordinates and prepare to test for oddness by dividing right circle limit x by 2 and then multiplying the integer quotient by 2.
Line 200: Test for oddness
Line 210: If even subtract 1 from x.
Line 220: Test whether to plot an inside circle with screen color, if yes go to 250, else go to 230.
Lines 230 - 240: Paint circle blue.
Lines 250 - 320: Do not paint inside circle but leave it in background color, white. This is done by painting only the area between XX and XS and X to XQ. Test for oddness is performed on X.
Lines 330 - 380: Paint third inside circle red.
Line 390: End of program.
Line 400: Set location of circle center and length of the three radii, R1 inside circle, R2 middle circle, R3 outside circle.
Lines 410 - 430: Set high resolution multi color bit mapping.
Line 440: Clear high resolution area.
Line 450: Set high resolution color models to blue and red, 98 01100010. The high byte is blue, color 6 0110, and the low byte is 0010, color 2 or red.

```
100 GOSUB400:GOTO160
110 REM THIS PRGRAM PLOTS 3 CONCENTRIC
    CIRCLES IN MULTI-COLOR HI-RES MODE
130 CH=INT(XR/8):RO=INT(Y/8):LN=YAND7
140 BY=BASE+RO*320+8*CH+LN:BI=7-(XRAND7)
150 POKEBY,PEEK(BY)OR(2↑BI):RETURN
160 V=53280:POKEV,1:POKEV+1,1:REM SET WHITE
    BACKGROUND
170 FOR Y=C2-R3TOC2+R3STEP1:REM OUTSIDE BLUE
    CIRCLE
180 SQ=SQR((R3*R3)-(Y-C2)*(Y-C2))
190 XQ=C1-SQ:XP%=C1+SQ:XD=INT(XP%/2):XA%=XD*2
200 XX=XP%:IFXA%<>XP%THEN220
210 XX=XP%-1
220 IFY>=C2-R2ANDY<=C2+R2THEN250
230 FORXR=XXTOXQSTEP-2:GOSUB130:NEXTXR
240 GOTO320
250 SQ=SQR((R2*R2)-(Y-C2)*(Y-C2))
260 XT%=C1-SQ:XS=C1+SQ
270 FORXR=XXTOXSSTEP-2:GOSUB130:NEXTXR
280 XD%=INT(XT%/2):XA%=XD%*2
290 IFXA%<>XT%THEN310
300 X=XT%-1
310 FORXR=XTOXQSTEP-2:GOSUB130:NEXTXR
320 NEXT Y
330 FOR Y=C2-R1TOC2+R1STEP1:REM INTERNAL RED CIRCLE
340 SQ=SQR((R1*R1)-(Y-C2)*(Y-C2))
350 X=C1-SQ:XQ=X:X=C1+SQ:XP%=X:XD=INT(XP%/2):XA%=XD*2
360 X=XP%:IFXA%=XP%THEN380
370 X=XP%-1
380 FORXR=XTOXQSTEP-2:GOSUB130:NEXTXR,Y
390 POKE 1024,16:GOTO390
400 C1=160:C2=100:R1=30:R2=60:R3=80
410 BASE=8192:POKE53272,PEEK(53272)OR8
420 POKE53265,PEEK(53265)OR32:REM ENTER BIT MAP MODE
430 POKE53270,PEEK(53270)OR16
440 FORI=BASETOBASE+7999:POKEI,0:NEXT
450 FORI=1024TO2023:POKEI,98:NEXT:RETURN:REM SET COLOR
    TO RED AND BLUE
```

Editor's Note: The following is part of a letter regarding Dr. Halfon's article on "Painting" Circles.

Gentlemen:

I cannot get a color display (only black and white figures) with this program. In line 450, the 98 or color has no effect. Omitting line 430 (multi-color bit map turn-on) gives a predicted blue and red display. I am puzzled why 98 doesn't give a blue-white-red multi-color bit map display.

In any case, there is a simpler and faster way. I used Simons' Basic:

Note the correction of 1.6 for the Y radius because the screen is rectangular. Lines 75-80 is a little color pizzaz.

Regards,
Harry Metz.

```
10 HIRES  0,1 : MULTI 2,1,6
20 CIRCLE 80,100,50,50*1.6,3
30 PAINT 80,100,3
40 CIRCLE 80,100,30,30*1.6,2
50 PAINT 80,100,2
60 CIRCLE 80,100,15,15*1.6,1
70 PAINT 80,100,1
75 FOR X=1TO500:NEXT
80 MULTI INT(RND(0)*14),INT(RND(0)*14),INT(RND(0)*14)
85 GOTO75
```

Best of The TORPET C64

Programmable Characters

By Steven Darnold, Alexandra, New Zealand

The Commodore 64 has a wide variety of graphics modes. You can use PET graphics, sprites, multi-colour sprites, a bit map, a multi-colour bit map, programmable characters, multi-colour programmable characters or extended background colours. You can also use combinations of these modes. This rich selection of modes permits the 64 to produce extremely sophisticated graphics. However, there is a lot to learn before you can fully utilize the 64's capabilities. Programmable character definition is a good place to start.

When I am writing a program, I often find that I need a character which does not appear on the keyboard. For example, I was once working on an educational program to teach angles and I needed a degree sign. However, since I was using a PET, I had no way of producing one. The Commodore 64, on the other hand, is quite capable of producing a degree sign, or any other character which can be defined in an 8 X 8 block of dots.

The key to programmable characters on the Commodore 64 lies in the fact that any of its 256 characters can be redesigned. This means that you can change the spade sign (for example) into a degree sign. Then every time you hit shifted-A, you get a degree sign on the screen. PRINTs and POKEs will also produce the degree sign. However, before you can redesign characters, you have to put your 64 into the right frame of mind.

First reset your 64 and remove any cartridges. Then PRINT CHR$(142); CHR$(8). This locks the computer into the upper-case/graphics character set. The lower-case/upper-case character set can also be redesigned, but it makes this discussion easier if we avoid switching character sets. Push the shift key and the Commodore key simultaneously, and you'll see that no switching occurs.

Now POKE 792,116 : POKE 793,164. This alters the RESTORE routine to keep it from destroying the new characters we are going to build. Press RUN-STOP/RESTORE a few times. You should get a READY without the screen being cleared.

Now POKE 56,127 : CLR. This lowers the top of memory to give us some RAM to use. Enter PRINT FRE(0) and the result should be 30461. If it isn't, then you probably forgot to enter CLR.

NOW THE HARD PART

The next bit is the most difficult, so be careful. At present the character set is stored in ROM. In order to redesign the characters, we have to shift them to RAM. There are five steps: disable the interrupt, connect the ROM, transfer the characters, disconnect the ROM, and re-enable the interrupt. This must be done as one operation. Enter all of the following before pressing RETURN. In order to squeeze it in, it will be necessary to leave out the spaces and abbreviate POKE by P shift-O. POKE 56334,0 : POKE 1,51 : FOR I=0 TO 2047 : POKE 40960+I, PEEK(53248+I) : NEXT : POKE 1,55 : POKE 56334,1. This will take about 30 seconds to execute.

You now have a copy of the character set in RAM, but the computer is still using the set in ROM. The final step is to tell the computer to use the new character set. POKE 56576,149 : POKE 53272,8 : POKE 648, 128. Since the screen must be in the same block of memory as the character set, it shifts at the same time. The screen now starts at 32768 (just like the PET) and the character set starts at 40960.

Clear the screen and type ABC. The characters should look normal (if not, you have a problem). Now enter POKE 40971,0. Look at the A in ABC, look at the A in READY. The zero you put in 40971 wiped out the fourth line of the A. Try putting zeros in 40972 and 40973. What happens? See if you can make all of the A disappear. Now POKE

40976,0. Can you make all of the B disappear? Can you make the C disappear, too?

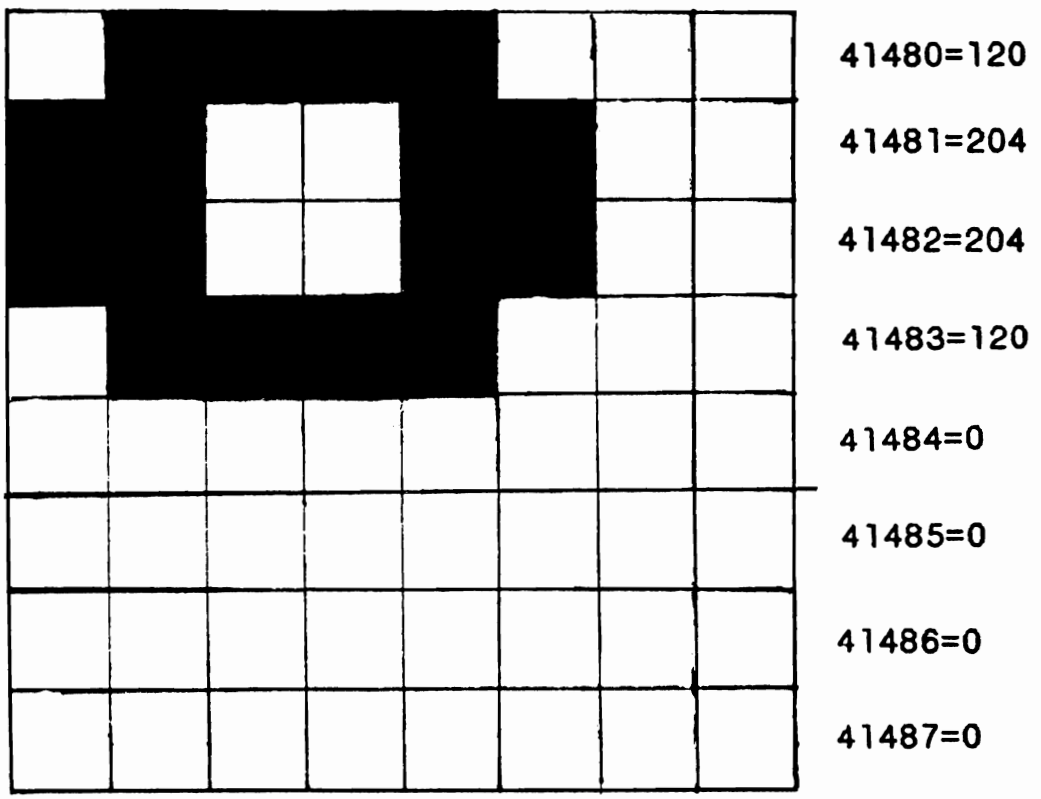
Each character is made up of eight lines. Each line is stored in a separate memory location. If the contents of a memory location is zero, then the corresponding line is blank. If the contents is 255, then the line is solid (try putting 255 into 40968). Different numbers between 0 and 255 give different types of lines. This is based on the binary representation of the number. A value of 255 gives a solid line because in binary it is 11111111. Similarly, a value of 0 is represented as 00000000. Each binary digit corresponds to a dot on the line. If the digit is 1, the dot is lit; if the digit is 0, the dot is off. Thus, if you want the left half of a line to be lit, the number to poke is 240 (= 11110000). Experiment with different numbers. See pages 77-78 in the 64 User Manual for details.

The character images are arranged in order, with each character taking 8 bytes. If you want to know where a particular character starts, use this formula: screen display code times 8 plus 40960. The screen display codes are listed on pages 132-134 of the User Manual. Codes for the reverse field letters run from 128 to 255. Thus, although we have mangled A, B, C (codes 1, 2, 3), we have not touched their reverse field representations (codes 129, 130, 131). Check this by pressing CTRL/RVS-ON and typing ABC.

Now we are ready to turn the spade sign into a degree sign. First look up the code for spade (=65), multiply it by 8 (=520) and add 40960 (=41480). This gives us the first line of the spade. POKE 120 (=01111000) into 41480 and 41483. POKE 204 (=11001100) into 41481 and 41482. POKE 0 into the bottom three lines (41484-41486). Now press shifted-A for a lovely degree sign —(see figure).

That's all there is to it. Now you can design your own characters. You'll have to spend a bit of time with an 8 X 8 grid working out the numbers, but once you get used to it, it isn't too difficult.

FIGURE 1.



Programming the Commodore 64 Function Keys

By Paul Thompson & Ron Radko, Toronto, Ont.

This program is on The Best Programs Disk

This program is designed to allow the programming of the FUNCTION keys. Each FUNCTION key may be programmed with up to 10 characters including multiple RETURN characters. The RETURN character(s) will function as a typed carriage return and can be used more than once for each FUNCTION key, e.g., you can type LIST (cr) RUN (cr)* and that FUNCTION key, when pressed, will LIST and then RUN the program in memory.

To program the FUNCTION keys, RUN program 2. There will be a slight pause while the machine language is POKEd into place, and then the computer will prompt you with:

F1 = ?

You can now program the F1 key with a maximum of 10 characters, and press RETURN. The computer will prompt you with:

F2 = ?

This cycle will repeat until you have programmed all eight of the FUNCTION keys (or defaulted by pressing RETURN). The program will then NEW itself to give the user full memory capacity.

While this program is totally user transparent, by hitting the RUN/STOP RESTORE keys, you must type:

SYS 12*4096 (cr)

This will restore the FUNCTION keys to their previous values.

* — carriage return is a back arrow.

(cr) is a carriage return.

Program 1 is the disassembly of the machine language routine found in program 2 on page 35, and is provided here for the interested assembly language programmers.

PROGRAM 1
DISASSEMBLED MACHINE CODE

```
2
20:     C000                        .OPT P4,00
30:     C000                        *=   $C000
40:     C000            NOKEYS      =    $C6      ;NO OF KEYS IN BUFFER
50:     C000            INKEY       =    $D7      ;LAST KEY PRESSED
60:     C000            IRQVEC      =    $0314    ;IRQ VECTOR
70:     C000            IRQRTN      =    $EA31    ;NORMAL IRQ ROUTINE
71:     C000            BUFFER      =    $0277    ;BEGINNING OF KEYBOARD BUFFER
80:     C000 78                     SEI
90:     C001 A2 0D                  LDX  #<NEWIRQ ;SET THE IRQ ROUTINES
100:    C003 A0 C0                  LDY  #>NEWIRQ ;TO START AT THIS
110:    C005 8E 14 03               STX  IRQVEC   ;PROGRAM
120:    C008 8C 15 03               STY  IRQVEC+1
130:    C00B 58                     CLI
140:    C00C 60                     RTS           ;RETURN TO BASIC
150:    C00D 48         NEWIRQ      PHA           ;BEGINNING OF NEW IRQ ROUTINE
150:    C00E 8A                     TXA           ;SAVE ALL REGISTERS
150:    C00F 48                     PHA
150:    C010 98                     TYA
150:    C011 48                     PHA
160:    C012 A5 D7                  LDA  INKEY    ;GET THE LAST KEY PRESSED
170:    C014 C9 85                  CMP  #133     ;CHECK IF IT IS ONE OF
180:    C016 F0 25                  BEQ  F1START  ;THE FUNCTION KEYS
```

```
190:   C018 C9 89          CMP  #137
200:   C01A F0 38          BEQ  F2START
210:   C01C C9 86          CMP  #134
220:   C01E F0 4B          BEQ  F3START
230:   C020 C9 8A          CMP  #138
235:   C022 F0 5B          BEQ  F4START
240:   C024 C9 87          CMP  #135
250:   C026 F0 6B          BEQ  F5START
260:   C028 C9 8B          CMP  #139
270:   C02A F0 7B          BEQ  F6START
280:   C02C C9 88          CMP  #136
290:   C02E D0 03          BNE  NEXT1
295:   C030 4C BB C0       JMP  F7START
300:   C033 C9 8C  NEXT1   CMP  #140
310:   C035 D0 03          BNE  NEXT2
315:   C037 4C CF C0       JMP  F8START
320:   C03A 4C E2 C0 NEXT2 JMP  MOVEON1 ;IF NOT GO TO END OF ROUTINE
330:   C03D A2 00  F1START LDX  #0      ;CLEAR THE X REGISTER
340:   C03F BD EA C0 LOOP1 LDA  F1,X    ;GET FIRST STORED CHARACTER
350:   C042 C9 00          CMP  #0      ;CHECK IF IT IS THE LAST ONE
350:   C044 D0 03          BNE  NEXT3
350:   C046 4C E0 C0       JMP  MOVEON  ;IF SO GOTO TO END
355:   C049 9D 77 02 NEXT3 STA  BUFFER,X ;PUT IT IN THE KEYBOARD BUFFER
360:   C04C E8             INX
360:   C04D E0 0A          CPX  #10     ;CHECK IF THE MAXIMUM
360:   C04F D0 EE          BNE  LOOP1   ;HAS BENN REACHED
365:   C051 4C E0 C0       JMP  MOVEON
370:   C054 A2 00  F2START LDX  #0
380:   C056 BD F4 C0 LOOP2 LDA  F2,X    ;SEE ABOVE
390:   C059 C9 00          CMP  #0
390:   C05B D0 03          BNE  NEXT4
390:   C05D 4C E0 C0       JMP  MOVEON
400:   C060 9D 77 02 NEXT4 STA  BUFFER,X
405:   C063 E8             INX
405:   C064 E0 0A          CPX  #10
405:   C066 D0 EE          BNE  LOOP2
410:   C068 4C E0 C0       JMP  MOVEON
420:   C06B A2 00  F3START LDX  #0
430:   C06D BD FE C0 LOOP3 LDA  F3,X
440:   C070 C9 00          CMP  #0
440:   C072 F0 6C          BEQ  MOVEON  ;SEE ABOVE
450:   C074 9D 77 02       STA  BUFFER,X
455:   C077 E8             INX
455:   C078 E0 0A          CPX  #10
455:   C07A D0 F1          BNE  LOOP3
460:   C07C 4C E0 C0       JMP  MOVEON
470:   C07F A2 00  F4START LDX  #0
480:   C081 BD 08 C1 LOOP4 LDA  F4,X    ;SEE ABOVE
490:   C084 C9 00          CMP  #0
490:   C086 F0 58          BEQ  MOVEON
500:   C088 9D 77 02       STA  BUFFER,X
505:   C08B E8             INX
505:   C08C E0 0A          CPX  #10
505:   C08E D0 F1          BNE  LOOP4
510:   C090 4C E0 C0       JMP  MOVEON
520:   C093 A2 00  F5START LDX  #0
530:   C095 BD 12 C1 LOOP5 LDA  F5,X    ;SEE ABOVE
540:   C098 C9 00          CMP  #0
540:   C09A F0 44          BEQ  MOVEON
550:   C09C 9D 77 02       STA  BUFFER,X
560:   C09F E8             INX
560:   C0A0 E0 0A          CPX  #10
```

```
560:   C0A2 D0 F1              BNE  LOOP5
570:   C0A4 4C E0 C0           JMP  MOVEON
580:   C0A7 A2 00     F6START  LDX  #0
590:   C0A9 BD 1C C1  LOOP6    LDA  F6,X     ;SEE ABOVE
600:   C0AC C9 00              CMP  #0
600:   C0AE F0 30              BEQ  MOVEON
610:   C0B0 9D 77 02           STA  BUFFER,X
620:   C0B3 E8                 INX
620:   C0B4 E0 0A              CPX  #10
620:   C0B6 D0 F1              BNE  LOOP6
630:   C0B8 4C E0 C0           JMP  MOVEON
640:   C0BB A2 00     F7START  LDX  #0
650:   C0BD BD 26 C1  LOOP7    LDA  F7,X     ;SEE ABOVE
660:   C0C0 C9 00              CMP  #0
660:   C0C2 F0 1C              BEQ  MOVEON
670:   C0C4 9D 77 02           STA  BUFFER,X
680:   C0C7 E8                 INX
680:   C0C8 E0 0A              CPX  #10
680:   C0CA D0 F1              BNE  LOOP7
690:   C0CC 4C E0 C0           JMP  MOVEON
700:   C0CF A2 00     F8START  LDX  #0
710:   C0D1 BD 30 C1  LOOP8    LDA  F8,X     ;SEE ABOVE
720:   C0D4 C9 00              CMP  #0
720:   C0D6 F0 08              BEQ  MOVEON
730:   C0D8 9D 77 02           STA  BUFFER,X
740:   C0DB E8                 INX
740:   C0DC E0 0A              CPX  #10
740:   C0DE D0 F1              BNE  LOOP8
1000:  C0E0 86 C6     MOVEON   STX  NOKEYS   ;TELL THE MACHINE # OF KEYS IN BUFFER
1010:  C0E2 68        MOVEON1  PLA           ;RESTORE ALL REGISTERS
1010:  C0E3 A8                 TAY
1010:  C0E4 68                 PLA
1010:  C0E5 AA                 TAX
1010:  C0E6 68                 PLA
1020:  C0E7 4C 31 EA           JMP  IRQRTN   ;RETURN TO NORMAL IRQ ROUTINES
1030:  C0EA 20 20 20 F1        .ASC "    "   ;SPACE FOR THE F1 KEY
1040:  C0F4 20 20 20 F2        .ASC "    "   ;SPACE FOR THE F2 KEY
1050:  C0FE 20 20 20 F3        .ASC "    "   ;SPACE FOR THE F3 KEY
1060:  C108 20 20 20 F4        .ASC "    "   ;SPACE FOR THE F4 KEY
1070:  C112 20 20 20 F5        .ASC "    "   ;SPACE FOR THE F5 KEY
1080:  C11C 20 20 20 F6        .ASC "    "   ;SPACE FOR THE F6 KEY
1090:  C126 20 20 20 F7        .ASC "    "   ;SPACE FOR THE F7 KEY
1095:  C130 20 20 20 F8        .ASC "    "   ;SPACE FOR THE F8 KEY
```

Program 2 is what you type in. Save it before you run it (in case it crashes).

PROGRAM 2
BASIC LISTING

```
5 FORN=49152TO49385:READA:POKEN,A:NEXT
10 FORN=0TO7
20 PRINT"F"N+1"=";:INPUTF$
30 IF LEN(F$)>10THENX$=LEFT$(X$,10)
40 FORX=1TOLEN(F$)
50 X$=MID$(F$,X,1)
60 IFX$="←"THENX$=CHR$(13)
65 IFX$=""THEN85
70 POKE49385+N*10+X,ASC(X$)
```

```
80 NEXT
85 FORR=XTO10:POKE49385+N*10+R,0:NEXT
90 NEXT
100 SYS12*4096
110 NEW
200 DATA 120, 162, 13, 160, 192, 142, 20, 3, 140, 21, 3, 88, 96, 72
201 DATA 138, 72, 152, 72, 165, 215, 201, 133, 240, 37, 201, 137, 240, 56
202 DATA 201, 134, 240, 75, 201, 138, 240, 91, 201, 135, 240, 107, 201,139
203 DATA 240, 123, 201, 136, 208, 3, 76, 187, 192, 201, 140, 208, 3, 76
204 DATA 207, 192, 76, 226, 192, 162, 0, 189, 234, 192, 201, 0, 208, 3
205 DATA 76, 224, 192, 157, 119, 2, 232, 224, 10, 208, 238, 76, 224, 192
206 DATA 162, 0, 189, 244, 192, 201, 0, 208, 3, 76, 224, 192, 157, 119
207 DATA 2, 232, 224, 10, 208, 238, 76, 224, 192, 162, 0, 189, 254, 192
208 DATA 201, 0, 240, 108, 157, 119, 2, 232, 224, 10, 208, 241, 76, 224
209 DATA 192, 162, 0, 189, 8, 193, 201, 0, 240, 88, 157, 119, 2, 232
210 DATA 224, 10, 208, 241, 76, 224, 192, 162, 0, 189, 18, 193, 201, 0
211 DATA 240, 68, 157, 119, 2, 232, 224, 10, 208, 241, 76, 224, 192, 162
212 DATA 0, 189, 28, 193, 201, 0, 240, 48, 157, 119, 2, 232, 224, 10
213 DATA 208, 241, 76, 224, 192, 162, 0, 189, 38, 193, 201, 0, 240, 28
214 DATA 157, 119, 2, 232, 224, 10, 208, 241, 76, 224, 192, 162, 0, 189
215 DATA 48, 193, 201, 0, 240, 8, 157, 119, 2, 232, 224, 10, 208, 241
216 DATA 134, 198, 104, 168, 104, 170, 104, 76, 49, 234

READY.
```

I'LL HAVE A BEER BUT JUST CHIPS FOR MY FRIEND...ANYWAY, THE WIFE STARTS AGAIN THIS MORNING: "YOU SPEND TOO MUCH TIME WITH THAT STUPID MACHINE," SHE SAYS....

Dvorak Keyboard

By William R. Frenchu, Princeton, N.J.

This program is on The Best Programs Disk

In the past few months there have been several articles comparing the Dvorak and the standard or "QWERTY" typewriter keyboards (Figure 1). Many feel that the Dvorak is a much easier board to use and that people trained on it can reach higher typing speeds with less fatigue than those using the QWERTY system. The QWERTY board gets its name from the first row of letter keys and is practically as old as the typewriter itself. When typewriters were in their infancy (and all mechanical) the letters were arranged on the keyboard in this manner to keep typists from typing faster than their machines could operate, and eliminate some of the key jamming that occured when two or more keys were pressed too quickly. The most commonly used keys were spread out among all three rows so the typist would be slowed down in reaching for them.

The Dvorak keyboard started to become popular with the advent of electronic typewriters and word processors with few (or quicker) mechanical parts. It returned the most commonly used keys to the "home row", the row on which the typist's fingers rest. With less reaching, the typist was able to type faster and with less fatigue.

The program presented here will enable the user to experiment with the Dvorak keyboard by reassigning the Commodore 64 keyboard.

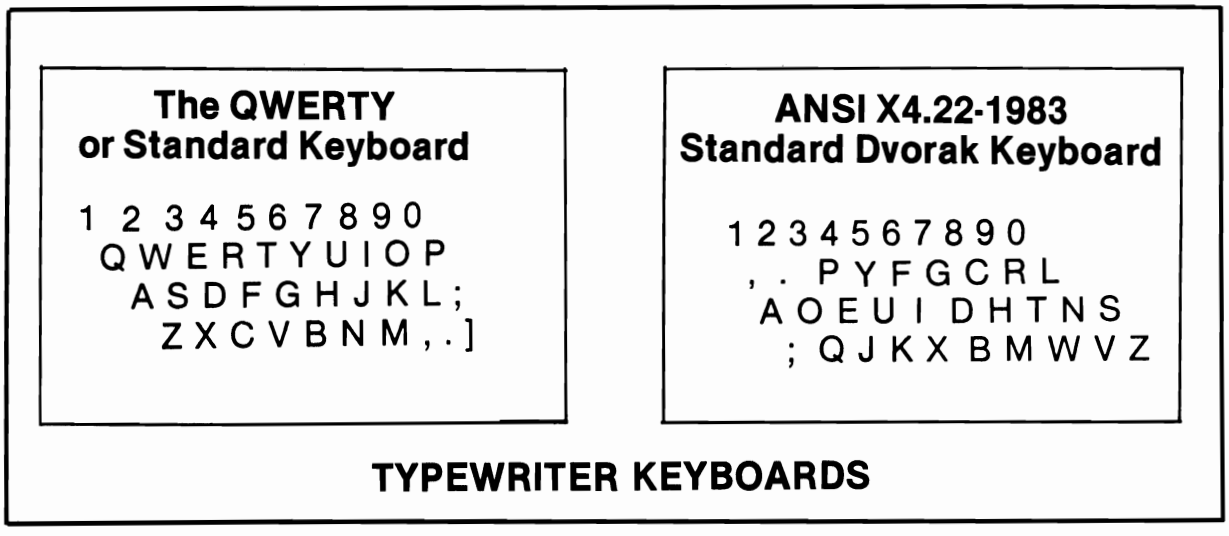
THE KEY TABLE

The Commodore 64 uses a table in the kernal to determine which character is assigned to a particular key. The table is arranged by "key number" a number different from both the screen and CHR$ codes. The key number of any key can be displayed with the following one line program:

```
10 PRINT "[clr]";PEEK(197) :GOTO 10
```

where [clr] is entered by pushing the "SHIFT" and "CLR/HOME" keys.

The easiest way to redefine the keyboard would probably be to change the vector, that points to this table, to point to a new table somewhere in RAM. This program for the Commodore 64 uses a slightly different approach, made possible by the ability of the 6510 processor to "bank" certain areas of memory in and out of its address space. All ROM on the 64 has associated with it a section of RAM occupying the same address. Which type of memory the processor sees at any one time is determined by an I/O port and data direction register at locations 0 and 1. A machine language subroutine first copies both Basic and the kernal into their underlying RAM areas. The ROM areas are then banked out and the original keyboard decode table is modified without changing the pointer. To restore the original keyset, the ROM is simply banked back in. This method was chosen in the hopes that the new keyboards could then be used with some of the existing word processor and typing tutor software. Unfortunately, the only word processor it has been tried with, the disk version of Commodore's own EasyScript, uses a system restart (SYS 64738) to start the program. This banks the ROM back in and the new keyboard can't be used.



TYPEWRITER KEYBOARDS

PROGRAM TO CHANGE THE COMMODORE 64 STANDARD QWERTY KEYBOARD TO A DVORAK KEYBOARD

```
5 :
6 :
10 REM *** MAIN PROGRAM ***
20 :
30 GOSUB 4000:REM *** LOAD AND RUN SWAP ROUTINE ***
40 DIM D(194):GOSUB 5400:REM *** LOAD DVORAK DATA ***
50 GOSUB 5050:END:REM *** LOAD DATA INTO KEY TABLE, END ***
3994 :
3995 :
3996 REM *** LOAD AND RUN SWAP ROUTINE ***
3997 :
4000 T=0:FOR I=0 TO 62:READ J:T=T+J:POKE 49152+I,J:NEXT
4010 IF T=10067 GOTO 4050:REM *** TOTAL FOR ML ROUTINE ***
4020 PRINT "(2crsr d]      (rvs on]DATA ERROR IN SWAP SUBROUTINE"
4040 END
4050 SYS 49152:RETURN:REM *** SWAP OUT BASIC AND KERNAL ***
4095 :
4096 :
4097 REM *** DATA FOR ML ROUTINE THAT COPIES BASIC AND KERNAL INTO RAM ***
4098 REM *** CODE USES NO ABSOLUTE ADDRESSES SO MAY BE LOCATED ANYWHERE ***
4099 :
4100 DATA 169,0,133,251,169,160,133,252,162,32,160,0,177,251,145,251,136
4110 DATA 240,2,208,247,202,240,4,230,252,208,238,169,0,133,251,169,224,133,252
4120 DATA 162,32,169,0,177,251,145,251,136,240,2,208,247,202,240,4,230,252,208
4130 DATA 238,165,1,41,253,133,1,96
4994 :
4995 :
4996 REM *** ROUTINES TO READ AND LOAD DVORAK KEY DATA ***
4997 :
4998 REM *** SUBROUTINE FOR POKING DVORAK DATA INTO KEY TABLE ***
4999 :
5050 FOR I=0 TO 194
5070 POKE I+60289,D(I)
5090 NEXT
5100 RETURN
5396 :
5397 :
5398 REM *** READ AND CHECK DVORAK DATA ***
5399 :
5400 T=0:FOR I=0 TO 194:READ D(I):T=T+D(I):NEXT
5405 IF T=22987 THEN 5410
5407 PRINT "(2crsr d]      (rvs on]DATA ERROR IN DVORAK KEYSET"
5408 END
5410 RETURN
5496 :
5497 :
5498 REM *** DVORAK DATA ARRANGED BY KEYNUMBER ***
5499 :
5500 DATA 20,13,29,136,133,134,135,17,51,44,65,52,59,79,46,1,53,80,69,54,74,85
5510 DATA 89,81,55,70,73,56,88,68,71,75,57,67,72,48,77,84,82,66,43,76,78,45
5520 DATA 86,83,64,87,92,42,58,19,1,61,94,90,49,95,4,50,32,2,47,3,255
5530 DATA 148,141,157,140,137,138,139,145,35,60,193,36,93,207,62,1,37,208,197
5540 DATA 38,202,213,217,209,39,198,201,40,216,196,199,203,41,195,200,48
5550 DATA 205,212,210,194,219,204,206,221,214,211,186,215,169,192,91,147,1
5560 DATA 61,222,218,33,95,4,34,160,2,63,131,255
5570 DATA 148,141,157,140,137,138,139,145,150,60,176,151,93,185,62,1,152
5580 DATA 175,177,153,181,184,183,171,154,187,162,155,189,172,165,161,41
5590 DATA 188,180,48,167,163,178,191,166,182,170,220,190,174,164,179,168,223,91
5600 DATA 147,1,61,222,173,129,95,4,149,160,2,63,131,255
```

The author has also provided a lengthier program that permits one to redefine any keys in the keyboard so that one may design their own keyboard. Available on the Best Programs Disk.

Menu Selection With A Joystick

By Alfred J. Bruey, Jackson, MI

This program is on The Best Programs Disk

There are hundreds, perhaps thousands, of programs that use the joystick for operations. Most of these are game programs. The introduction of LISA by Apple demonstrated that it is possible to write a business program which provides for a non-keyboard interface between the user and the applications software. The article describes one approach that might be used to allow joystick control of the Commodore 64.

INTRODUCTION

This program is for demonstration purposes only. It will demonstrate:

1. How to program for the Commodore 64 joystick.
2. How to select items from a menu using a joystick.

The program will *not* perform the functions that you select with the joystick. I have indicated where you need to add coding if you want to continue building on this program.

PROGRAMMING THE JOYSTICK

Although there are two joystick ports on the 64, labelled Control Port 1 and Control Port 2, the following discussion and the program will assume that a joystick is plugged into Control Port 1.

When a joystick is plugged into Control Port 1 and the joystick handle is moved around or the fire button pressed, specific values are placed in memory location 56321. Location 56321 is one byte (8 bits) long. (See Figure 1) We will only be interested in the 5 rightmost bits of this location.

To tell whether the fire button has been pressed, we only need to look at bit 4 of location 56321. If that bit is a 1, it means the button has *not* been pressed. If it is a 0, the button *has* been pressed. To look at location 56321, we have to use the PEEK instruction. To look at bit 4 of this location, we can AND the value in location 56321 with the value 16, which in binary form is 00010000. For example, if 56321 contains the binary value 10110111, then

```
10110111 AND 00010000 = 00010000
```

and since the new value is 00010000, which is the binary representation of the decimal number 16, we know that the button has not been pressed. If the result of this operation had been 0, we would have known that the fire button had been pressed.

The same principle applies in determining the joystick position. As figure 1 shows, the low order four bits determine whether or not the joystick has been moved and, if it has, which position it has been moved to. To zero out the first (high-order) four bits and keep the last (low-order) four bits unchanged, all we have to do is AND the value of location 56321 with 15, which is 00001111 in binary notation. The new value, which I'll abbreviate as JD (for *J*oystick *D*irection) can only take on part of the values from 0 to 15. Each 1 in the value of JD represents a joystick direction that was *not* selected. I'll subtract the value of JD from 15 to reverse the bit values in JD so that a 1 will represent a chosen direction.

Listing 1 shows how to zero out the high order four bits and then convert the value remaining to one where the binary representation contains a 1 if the joystick position is chosen and a 0 if it is not.

In this program, we will only be interested in moving the joystick up or down. Listing 1 gets a value of FB (0 = fired, 16 = not fired) and JD (JD = 1 if stick pushed up, 2 if pushed down). As you can see by the REM statements in Listing 1, JD can take on other values, which we will ignore in this program.

THE PROGRAM

Listing 2 shows the complete program. Notice that the joystick routine that was shown in Listing 1 is included here as a subroutine. The first thing you should do is enter the program and then run it. I haven't included any operating instructions as part of the program because I wanted to keep it easy to enter. The operation is simple: just type RUN and then press the RETURN key. Then move the joystick ahead or back. When you've got the cursor on the selection you want, press the joystick button. Remember, I warned you at the start that the program doesn't do any of the things listed on the menu (except for the END OF RUN selection); the program simply demonstrates how to use the joystick to make selections from a menu.

If a PET is DOWN, it's out of SORTS. — Ylimaki

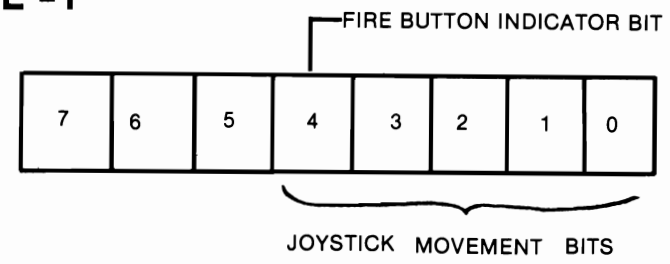
CONCLUSION

There's no limit to where you can go from here. You might want to put menu selections all over the screen. Then you'll have to check for other joystick directions. Maybe you'd like to write a program that requires some data from the keyboard and some from the joystick. As a final test of your understanding, you might try writing a joystick-controlled game program like some of the arcade games. I should warn you, however, that if you write a game that's too complicated, you'll have to program it in machine language or it will run so slowly that it won't be any challenge to the player.

LISTING 1

```
1000 REM CHECK FOR FIRE BUTTON PRESSED
1010 REM JV IS JOYSTICK VALUE
1020 JV = PEEK(56321)
1030 REM GET VALUE OF FIRE BUTTON
1040 REM 0 IF FIRED, 16 IF NOT FIRED - PUT IN FB
1050 FB = JV and 16
1060 REM GET JD, JOYSTICK DIRECTION VALUE
1070 REM = 1 UP,2 = DOWN
1080 REM JD CAN HAVE OTHER VALUES THAT WE WON'T
USE
1090 JD = 15 - (JV AND 15)
```

FIGURE -1



LISTING 2

```
100 REM MENU SELECTION WITH A JOYSTICK
110 REM DISPLAY MENU WITH CURSOR
120 L=1510:V=0
130 PRINT"■":POKE L,81
140 PRINT"SAMPLE MENU: USE JOYSTICK TO CHOOSE LINE"
150 PRINT"THEN PRESS BUTTON TO SELECT"
160 PRINT"■■■■■■■■■■■■■PAYROLL PROGRAM"
170 PRINT"■■■■■■■■■■■■■ACCOUNTS PAYABLE PROGRAM"
180 PRINT"■■■■■■■■■■■■■ACCOUNTS RECEIVABLE PROGRAM"
190 PRINT"■■■■■■■■■■■■■INVENTORY PROGRAM"
200 PRINT"■■■■■■■■■■■■■STOCKHOLDER RECORDS PROGRAM"
210 PRINT"■■■■■■■■■■■■■STOP RUN"
220 GOSUB 350:REM CHECK JOYSTICK AND BUTTON
230 REM SEE IF BUTTON WAS PRESSED
240 IF FB=0 THEN 460
250 REM IF JOYSTICK NOT UP OR DOWN
260 REM AND FIRE BUTTON NOT PRESSED,
270 REM GO BACK TO SCAN KEYBOARD
280 REM CHECK FOR JOYSTICK UP
290 IF JD=1 THEN IF V>4  THEN 220
300 IF JD=1 THEN V=V+3:POKE L,96:L=L-120:POKE L,81:FORI=1TO250:NEXTI:GOTO 220
310 REM CHECK FOR JOYSTICK DOWN
320 IF JD=2 THEN IF V<-7  THEN 220
330 IF JD=2 THEN V=V-3:POKE L,96:L=L+120:POKE L,81:FORI=1TO250:NEXTI:GOTO 220
```

```
340 GOTO 220:REM GO BACK TO CHECK FOR JOYSTICK
350 REM CHECK FOR FIRE BUTTON PRESSED
360 REM AND TO FIND JOYSTICK DIRECTION
370 REM FB IS FIREBUTTON VALUE
380 REM FB=0 IF PRESSED
390 REM FB=16 IF NOT PRESSED
400 JV=PEEK(56321)
410 FB=JV AND 16
420 JD=15 - (JV AND 15)
430 REM JD=1 IF JOYSTICK UP
440 REM JD=2 IF JOYSTICK DOWN
450 RETURN
460 REM HERE IS WHERE YOU GO IF BUTTON IS PRESSED
470 FOR I=1 TO 250:NEXT I
480 IF V=6 THEN PRINT"■■■■■■■■■■PAYROLL PROGRAM SELECTED":GOTO 540
490 IF V=3 THEN PRINT"■■■■■■■■■■ACCOUNTS PAYABLE SELECTED":GOTO 540
500 IF V=0 THEN PRINT"■■■■■■■■■■ACCOUNTS RECEIVABLE SELECTED":GOTO 540
510 IF V=-3 THEN PRINT"■■■■■■■■■■INVENTORY PROGRAM SELECTED":GOTO 540
520 IF V=-6 THEN PRINT"■■■■■■■■■■STOCKHOLDER RECORD PROGRAM":GOTO 540
530 IF V=-9 THEN PRINT"■■■■■■■■■■END OF RUN":STOP
540 PRINT"■■PRESS BUTTON TO RETURN TO MAIN MENU"
550 REM WAIT HERE UNTIL BUTTON IS PRESSED
560 A=PEEK(56321) AND 16
570 IF A<>0 THEN 560
580 FORI=1 TO 250:NEXT I:GOTO 130
READY.
```


Speech Synthesis of C64

By Greg L. Halley, Silver Spring, MD

Have you been wondering when you would finally have the capability of providing your Commodore 64 computer with the gift of speech? Well, Tronix has recently released the Commodore 64 version of its Software Automatic Mouth (S.A.M.) disk based speech system (1). This popular software package has already been available for the Apple, Atari and VIC-20 for some time now (2).

The system itself is very easy to access from the resident BASIC provided with the Commodore 64 computer by using a software "WEDGE" The software wedge provides you with the option of using only phoneme based speech or using a direct English-Speech translation. The format for using the direct English-Speech command from Commodore 64 BASIC can be accessed by direct quote or string variable as demonstrated below:

```
10 A$ = "during these turbulent times"
20 ]SAY "now is the time for..."
30 ]SAY A$
40 Goto 100
```

It is also noteworthy that you have a remarkable amount of control over the pitch, stress and speed of speech. This control is obtained by using the BASIC commands of]SPEED,]PITCH and]KNOBS from within the actual BASIC program utilizing the speech synthesis capability. The format for using these commands is outlined below:

```
10 ]RECITER
20 ]SPEED 60
30 ]PITCH 55
40 ]KNOBS 125,176
50 ]SAY "This is a trial speech pattern"
```

EXPLANATION OF BASIC COMMANDS

Line 10 — Calls the Reciter machine language subroutine which allows the program to use direct English-Speech translation.

Line 20 — The Speed command sets the speed of speech production by the Reciter subroutine. The range of values for Speed is 0-225 where the speed is inversely proportional to the value (i.e. 225 is very slow and 20 is quite fast).

Line 30 — The Pitch selection determines the quality of the spoken voice. The range of values for Pitch are 0-255 where the highest value corresponds with the lowest voice and the lowest value produces the highest voice.

Line 40 — The Knobs command essentially provides a means of changing the quality of the voice without affecting the Speed or Pitch of the voice. The analogy used in the user guide compares the Knobs control to defining the size of the mouth and the throat. The range of values is 0-255 with a direct correspondence between the higher values and a larger throat and mouth (i.e. 20 is a small mouth whereas 225 is a large mouth).

The Reciter subroutine called from BASIC provides a convenient means of providing English-Speech translation of material which varies from one run to the next. However, the system also includes a mechanism for providing a more refined control over speech which may be repetitive in nature (i.e. in a program which utilizes a small speech vocabulary). This system is based upon the conversion of English words into spoken phonemes. These phonemes are then placed within the Speech string in place of the English words and spoken with a remarkable degree of clarity when compared with the direct English-Speech translation. In addition, this phoneme-based system allows direct control of stress points within the program, this is done by placing a numeric value of 1-8 directly after the vowel to be stressed. This is useful because of the importance of context in spoken English and the ability this feature gives the programmer in conveying context through stressing key words, phrases, etc.

This all sounds fine and good you say, but what about the price you will pay in loss of BASIC RAM? Well, the truth is that the S.A.M. package can be very frugal if used properly. The machine subroutine which drives S.A.M. is approximately 10.75 K in length but the noticeable decrease in BASIC RAM is only approximately 2.75 K. This is possible because most of the S.A.M. package is located in memory not accessed by the BASIC operating system. If you decide to use the Reciter subroutine which allows direct English-Speech synthesis you will use additional RAM; however, this will conflict with the DOS WEDGE. The alternative placement of Reciter into low memory will consume an additional 6.0 K RAM.

In conclusion, the advantages of the S.A.M. system can be summarized as: 1) Relatively inexpensive (sugg. retail $60.00); 2) No additional hardware and resulting loss of user/expansion port as required in other speech systems; 3) Drive software provided to provide direct English-Speech translation at no additional cost as is true with some other systems; 4) Inclusion of a phoneme dictionary with the User manual provided with the software package; 5) It is available now, not some undefined point in the future that never seems to arrive. The disadvantages might be summarized as: 1) Quality of speech dependent upon the quality of the monitor speaker; 2) Loss of BASIC RAM in programs which may require a large amount of RAM 3) "Machine sounding" voice, the system not able to provide the more human sounding voices as other systems but then again it doesn't have their price either.

Letters of inquiry for the product may be directed to:

TRONIX
8295 S. La Cienega Blvd.
Inglewood, CA 90301
U.S.A.

(1) S.A.M. registered Trademark of TRONIX, Inc.
(2) APPLE registered Trademark of APPLE Computer
ATARI registered Trademark of ATARI Computer
VIC-20 is registered Trademark of COMMODORE ELECTRONICS, LTD.

BYTES **by Patrick Corrigan**

Creating Sprites on the C64

By David Bradley, Toronto, Ont.

First, get some graph paper and set up a grid 24 columns wide by 21 rows deep and number it as shown in Figure 1. This is the space that you have in which to create your sprites.

Look at Figure 2. OK, so it isn't the best Commodore flag, but it serves its purpose. The next thing I should tell you is how to change that grid into data. Look at the top of Figure 2. Note the way that the columns are numbered. If a square is filled in, it is considered 'on' or a logical 1. So change the filled-in squares to 1's and the blank squares to 0's such as Figure 3. Now, look at byte 1, row 1. All of the bits are 0. So the first byte will be 0. The bits in byte 2, 3 are also made up of 0's. So the data for the first line will be:

```
DATA 0, 0, 0
```

Now we look at row 2. The first byte is all 0's so it, like all the bytes in row 1, is equal to 0. But look at the next byte, byte 2, row 2. It is "0 1 1 1 1 1 0 0". So look at it this way:

```
1 0 0 0 0 0 0 0
2 6 3 1 0 0 0 0
8 4 2 6 8 4 2 1

0 1 1 1 1 1 0 0

0 + 64 + 32 + 16 + 8 + 4 + 0 + 0 = 124
```

So what you are doing is adding up the bits by replacing all of the 1's with the value of their columns and leaving the 0's as 0's.

Now it is time to put this data into the computer and see what the sprite looks like. So type in the following program, add the data statements for my sprite and then run the program.

To save you some time, here is the data for Figure 2.

```
        Byte 1    Byte 2    Byte 3
Row   01-000,    000,      000
Row   02-000,    124,      000
Row   03-001,    254,      000
Row   04-003,    254,      000
Row   05-007,    254,      000
Row   06-015,    130,      000
Row   07-030,    001,      255
Row   08-060,    001,      254
Row   09-056,    001,      252
Row   10-056,    001,      248
Row   11-056,    000,      000
Row   12-056,    001,      248
Row   13-056,    001,      252
Row   14-060,    001,      254
Row   15-030,    001,      255
Row   16-015,    130,      000
Row   17-007,    254,      000
Row   18-003,    254,      000
Row   19-001,    254,      000
Row   20-000,    124,      000
Row   21-000,'   000,      000
```

```
10 B = 53248
20 FOR I = 0 TO 62
30 READ A
40 POKE 64 * 200 + I, A
50 NEXTI
60 POKE 2040, 200
70 POKE B + 21,1
80 POKE   B, 160
90 POKE   B + 1, 127
100 DATA...
```

If you did the program correctly and got the data right, you should now see a Commodore flag in approximately the middle of your monitor. You have just created your first sprite! You know where the data came from but I bet you would like to know what some of the rest of the program does. So here is a line by line explanation of the program.

LINES AND DESCRIPTIONS

10 Sets the value of B to the start of display chip
20 Start of read loop
30 Reads A
40 Pokes A. The 200 * 64 is the place in memory that you will have to point the sprite so it knows what data it is supposed to use.
50 End of read loop
60 Pokes (points) 2040 (sprite 1) to get its data starting at memory location 200 * 64.
70 Turns on sprite 1
80 Sets vertical position for sprite 1
90 Sets horizontal position for sprite 1
100 The data you entered.

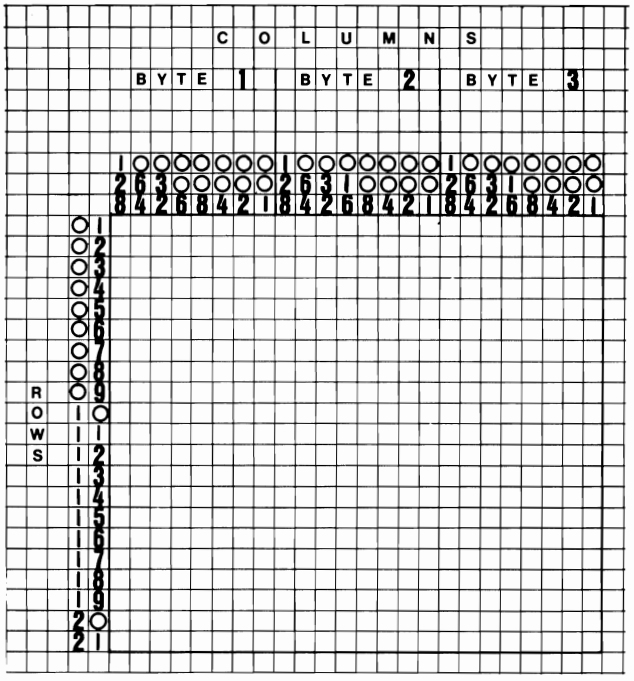


FIGURE 1

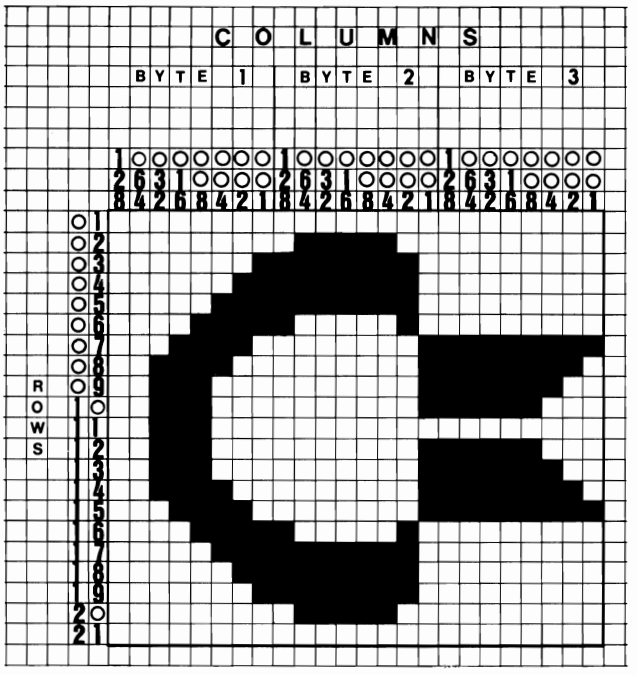


FIGURE 2

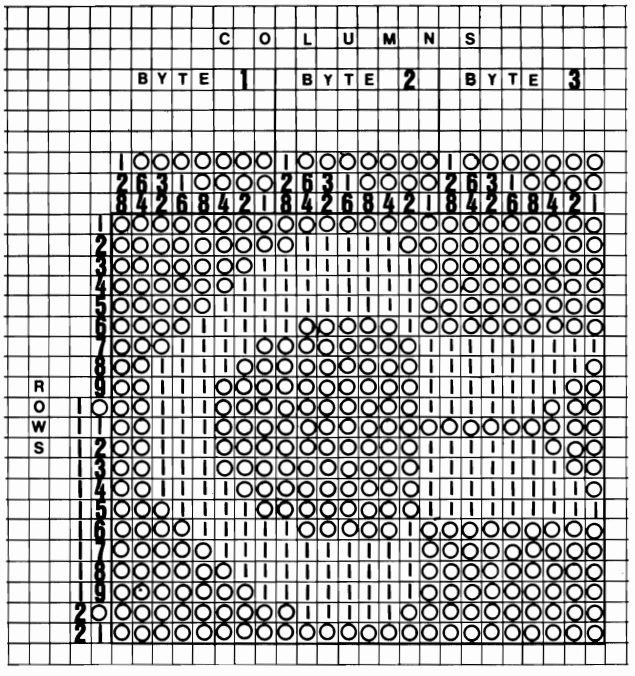
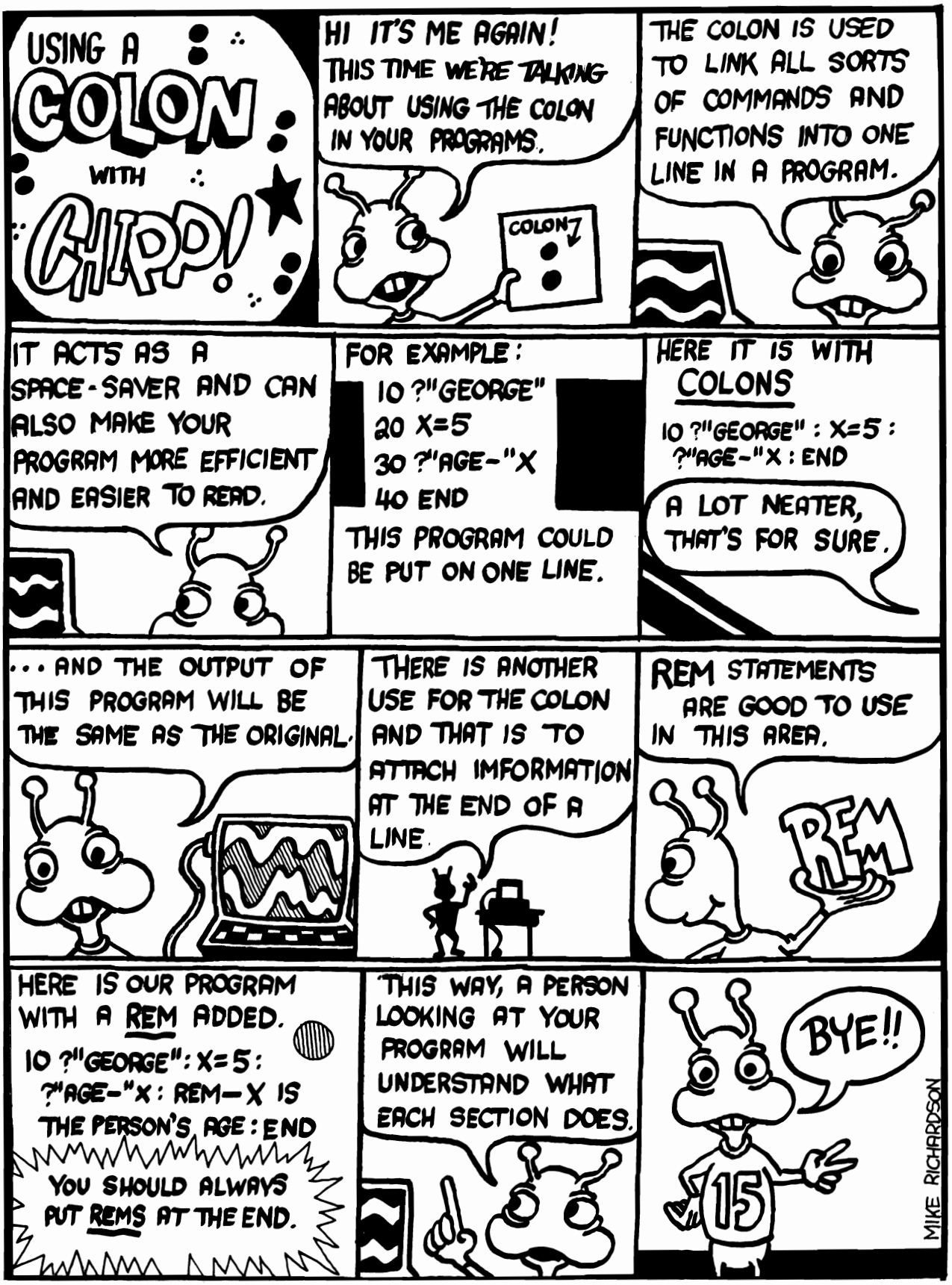


FIGURE 3


USING A COLON WITH CHIPP!
HI IT'S ME AGAIN! THIS TIME WE'RE TALKING ABOUT USING THE COLON IN YOUR PROGRAMS.
COLON
THE COLON IS USED TO LINK ALL SORTS OF COMMANDS AND FUNCTIONS INTO ONE LINE IN A PROGRAM.
IT ACTS AS A SPACE-SAVER AND CAN ALSO MAKE YOUR PROGRAM MORE EFFICIENT AND EASIER TO READ.
FOR EXAMPLE:
10 ?"GEORGE"
20 X=5
30 ?"AGE-"X
40 END
THIS PROGRAM COULD BE PUT ON ONE LINE.
HERE IT IS WITH COLONS
10 ?"GEORGE" : X=5 : ?"AGE-"X : END
A LOT NEATER, THAT'S FOR SURE.
... AND THE OUTPUT OF THIS PROGRAM WILL BE THE SAME AS THE ORIGINAL.
THERE IS ANOTHER USE FOR THE COLON AND THAT IS TO ATTACH IMFORMATION AT THE END OF A LINE.
REM STATEMENTS ARE GOOD TO USE IN THIS AREA.
REM
HERE IS OUR PROGRAM WITH A REM ADDED.
10 ?"GEORGE": X=5: ?"AGE-"X: REM-X IS THE PERSON'S AGE: END
YOU SHOULD ALWAYS PUT REMS AT THE END.
THIS WAY, A PERSON LOOKING AT YOUR PROGRAM WILL UNDERSTAND WHAT EACH SECTION DOES.
BYE!!
15
MIKE RICHARDSON

VIC

PAGE

Best of Both Worlds 46
Steve Garmon, Houston, Tex.
A VIC user compares his VIC with his Commodore 64.

Cursor Key Control 47
Terry Herckenrath, Toronto, Ont.
How to disenable and enable the cursor control keys in a program.

A Game input Routine 48
Terry Herckenrath, Toronto, Ont.
This is a small machine language routine that you can add to your game programs to increase their speed and versatility.

Talk is Cheap for VIC 51
Lee Urbanski, Madison, Wisc.
Here is a little commercial hardware device that will add speech to your VIC.

ROM RABBIT 52
Mayland Harriman, Pt. Arthur, Tex.
This is a description of a relatively inexpensive device that makes life a lot easier for the regular tape user. It has received many favourable reviews.

Take A Load Off Your VIC 53
Tony Davidson, Gananoque, Ont.
Some pointers for the hardware buff on how to keep the voltage requirements of add-ons from overloading the VIC.

VIC-20 Modifications 55
Dennis Sievers, Breeze, IL.
How to add a re-set switch and a numeric keypad to your VIC-20.

Redesigned 8K Card 57
Dennis Sievers, Breeze, IL.
How to add a write-protect switch and make your 8K RAM card re-locatable.

Best of Both Worlds

By Steve Garmon, Houston, Tex.

(reprint from CHUG)

There is no doubt in my mind that the VIC is still the cheapest and best way to get into computing on a true microcomputer. Even with its memory limitations and 22 column screen, it is a very useful machine, limited only by the imagination (and sometimes the budget) of the user. It has been sufficient enough to keep me happy for a long time and I am still finding new uses for it all the time.

I have used a VIC for everything from a dedicated printer interface for another computer to a 27K word-processor. It is extremely versatile with respect to its 4 large blocks (8K each) that are set aside in the unexpanded VIC for use as the owner sees fit. If he needs the extra RAM memory then he can purchase it in several different sized cartridges ranging from 3K to 27K and more! There are 2 separate areas set aside for input/output expansion that I have used for things like printer interfaces, voice synthesizers, and even a 2K EPROM that held some machine language routines which I used often.

With a little imagination, a lot can be done with a VIC and I intend to keep on using mine as long as I can. I just can't beat the price for the capabilities that it gives me. On the other hand, if the user starts needing large amounts of RAM, or if he needs a 40 column screen, then he has to decide between an expanded VIC or a 64.

In most cases, a person would spend more money trying to expand a VIC to fit his needs than he would if he bought a 64 to start with.

There are some trade-offs though. The 64 has 64K of RAM built in which eliminates any need for expanders. It also has a 40 column display but, in some cases, this could be considered a disadvantage. In school classrooms, the 22 column display offered by the VIC-20 is easier to read for small children.

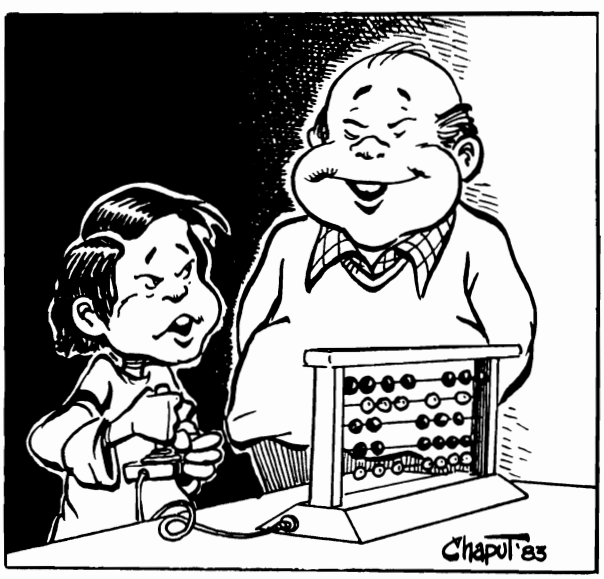
I think that I could sum up the differences between the VIC and the 64 by saying that the VIC was built with the hardware person in mind while the 64 was built with the programmer in mind.

Of course, I have no way of knowing exactly what Commodore International had in mind when they designed either one of these fine machines but the descriptions I just gave seem to fit very well. I finally had to get involved with the 64 because of a job-related application. The application that I am involved with needs a very accurate real time clock. The built-in clock on both the VIC and the 64 was determined to be not accurate enough because they are software clocks and they are dependent on system software to keep them updated. I had two options. I could spend about $130 on a plug-in card for the VIC or I could buy a 64 which has 2 built-in hardware clocks. The price of the VIC with plug-in card would be about $220. Needless to say, I bought the 64. In this situation, the 64 heavily out-weighed the VIC in terms of price/performance ratio.

So, I have entered the world of the programmer and I am having to learn the intricacies of the 64. I'm sure that with time I will become as familiar with it as I am with the VIC and when I do I will be telling you about it in this newsletter.

By the way, I couldn't leave the computer store without at least one of the excellent games for the 64, could I? I bought a copy of "JUMPMAN" and let me just say that I WAS IMPRESSED! You will have to have a disk drive to play this game because, according to the salesman, it is not available on cartridge or cassette. It was definitely the friendly way to start out a good friendship between myself and the 64.

After all, isn't it supposed to be the "USER FRIENDLY COMPUTER"?- - - Steve Garmon



COME ON, DAD! COULDN'T WE UPGRADE OUR SYSTEM?

Cursor Key Control

By Terry Herckenrath, Toronto, Ont.

This program is on The Best Programs Disk

QUESTION...

Dennis Smith from Marshall, Michigan wants to know how he can disable and enable the CURSOR CONTROL keys.

REPLY...

I assume that Dennis uses the INPUT statement to get some information from the 'user', but he doesn't want him/her to wander all over the screen with the cursor. I also assume that Dennis wants the cursor to show when input is required.

There is no convenient 'switch' that can be set if one wants to ignore certain keys. The only way you can ignore certain keys is by using the GET statement to get the input, one character at a time, examine the characters, ignore the ones you don't want and print the rest to the screen and collect them in another string variable. The GET statement however doesn't provide us with a visible cursor, so we will have to look after the cursor ourselves.

The VIC always has a cursor somewhere on the screen, but it makes it visible only when specifically told to. (The cursor is positioned on the screen by the PRINT statement). There are a number of memory locations in the VIC that tell the VIC whether the cursor should be visible and that keep track of what the cursor is doing and where it is at any given time. The ones we will be concerned with are:

204 - CURSOR ENABLE (0 = ENABLED)

207 CURSOR IN BLINK PHASE (0 = OFF,1 = ON)

To make the cursor appear on the screen we poke a zero in location 204. We will have to do this quite often, because the PRINT statement has a tendency to turn it off for us. We'll enable the cursor each time we GET a character. To keep the cursor visible while characters are printed on the screen, we force the VIC to flash the cursor each time we print a character by poking a zero in location 207. To make sure we don't leave a 'ghost' cursor behind when we print the CARRIAGE RETURN character, we will have to make sure that the cursor is OFF when we print the carriage return. We do this by WAITing until location 207 has a zero in its low order bit position (BIT 0). In order to save some processing time when we execute this input routine, we'll use an INFINITE LOOP to avoid GOTO statements. (If a routine is located near the end of a program, and you keep going back to a line near the beginning of the routine using the GOTO statement until some condition is met, BASIC will search for that line number right from the beginning of the program. Depending on the size of the program, that can introduce a noticeable delay.)

Whenever you want input from the 'user' and you want to disable the cursor control keys, GOSUB to this routine and it will return the inputted string in variable C$.

```
10000 C$="":FOR A=1 TO A:A=0:POKE 204,0
10010 REM CLEAR VARIABLE; START INFINITE LOOP;
    MAKE SURE CURSOR STAYS ENABLED
10020 GET A$:IF A$="" THEN NEXT A
10030 REM GET A CHARACTER
10040 IF A$="[DOWN]"OR A$="[UP]" OR A$="[LEFT]"OR A$="[RIGHT]" OR A$="
   [INST]"OR A$="[CLEAR]"OR A$="[HOME]" THEN NEXT A
10050 REM WEED OUT ALL CURSOR CONTROL KEYS PLUS THE INSERT KEY
10060 IF A$=CHR$(20) AND C$<>"" THEN C$=LEFT$(C$,LEN(C$)-1):PRINT A$;
10070 IF A$=CHR$(20) THEN NEXT A
10080 REM DELETE CHARACTER FROM STRING (IF STRING IS NOT NULL)
10090 IF A$=CHR$(13) THEN A=1:WAIT 207,1,1:PRINT:A$=""
10100 REM IF RETURN THEN END LOOP AND WAIT TILL CHARACTER NOT BLINKING
10110 C$=C$+A$:POKE 207,0:PRINT A$;:NEXT A:POKE 204,1:RETURN
10120 REM BUILD STRING VARIABLE; FORCE CHARACTER TO BLINK;
    PRINT CHARACTER
10130 REM WHEN LOOP IS FINISHED TURN OFF CURSOR, RETURN
```

A Game Input Routine

By Terry Herckenrath, Toronto, Ont.

This program is on The Best Programs Disk

Include this G.I.R. in your new or existing program when you want versatility, speed or both.

The G.I.R. will accept input from either the keyboard or the joystick. This is specified at run time, so you can give the user the choice.

The input can either be momentary (the input defaults to CENTRE if no direction is indicated from the keyboard/joystick), or latched (the input remains fixed until another direction is indicated from the keyboard/joystick — NO CENTRE position).

When input is to come from the joystick, the G.I.R. will either allow or disallow diagonal directions.

The G.I.R. is linked to the VIC's interrupt handler, so that the inputted values are always up-to-date when you use them in the program. This allows the user to indicate a change in direction even when the program isn't ready yet to check the inputted values. This is not possible when you handle the input from BASIC.

The routine as shown below is designed to be appended to an existing BASIC program (see note). After it is appended, RUN 10000 (see note) will POKE the actual G.I.R. in place at the end of the BASIC program. Statements 10000 and on can then be deleted from the program.

Before you can use the G.I.R. in your program, the following statement must be executed from within the BASIC program to link the G.I.R. to the VIC's interrupt handler.

```
SYS PEEK (46)*256 + PEEK (45)-30
```

To end the program, use SYS 65234 instead of 'END', to remove the G.I.R. from the interrupt handler.

The G.I.R. uses the value of memory location 155 to determine where the input is to come from. So, before the program starts to use the G.I.R., you must put the proper value in memory location 155:

```
  0 — Momentary input from keyboard (default)
  1 — Momentary input from joystick — no diagonals
 65 — Momentary input from joystick — with diagonals
128 — Latched input from keyboard
129 — Latched input from joystick — no diagonals
193 — Latched input from joystick — diagonals
```

When you select input from the keyboard, the G.I.R. will check the following keys:

```
P for UP
L for LEFT
; for RIGHT (semi-colon)
. for DOWN (period)
SHIFT for FIRE
```

The G.I.R. will set the value of memory locations 156, 158 and 159 to pass the user's input to the BASIC program:

```
156 - Fire button/key 0-no 1-yes
158 - Vertical direction 0-up 1-centre 2-down
159 - Hortizontal direction 0-left 1-centre 2-right
```

The BASIC program then simply 'peeks' these memory locations to find out what the user wants.

NOTE:

Type in and save both of the following programs. Load and run "G.I.R.—INSTR". This will give you instructions on adding G.I.R. to your programs.

NOTE:

The crazy spacing in this listing is supposed to be here because it is meant for the VIC screen.

```
1 poke 36879,8:print"[clear,text,lock,green,down,space5,
   rvs]VIC G.I.R.[rvoff]
2 print"[down]This is a machine      language,
    interrupt   driven Game Input     Routine.
3 print"[down]The GIR will accept    input from either the keyboard or th
   e joy-  stick.
```

```
4 print"[down]This is decided at RUNtime, so you can give the player the
    choice.
5 gosub 49
6 print"[clear]You can also tell the GIR whether the input is to be 'rem
  embered'.";
7 print"[down]This is called latchedinput.
8 print"[down]When the input is NOT latched,
    it returns toCENTER if no direction";
9 print"is indicated.
10 print"[down]The GIR as supplied onthis tape starts with line# 63000.
11 print"[down]As it will be appendedto your program,
    your program may NOT have
12 print"line#'s greater than  62999.
13 gosub 49
14 print"[clear]To add the GIR to yourprogram:
15 print"[down]First LOAD your pro-  gram into the VIC,
       then clear the screen
16 print"and enter:
17 print"[down]PRINT PEEK(43)PEEK(44)
18 print"Write down the two    numbers that the VIC  has printed on the
19 print"screen, then enter:
20 print"[down]I=PEEK(45)+PEEK(46)*256-2:I%=I/256:POKE43,I-I%*256
  :POKE44,I%
21 gosub 49
22 print"[clear]Now LOAD VIC G.I.R.
23 print"[down]Then POKE the two num-bers you wrote down   earlier into
  locations43 and 44.
24 print"[down]Next enter RUN63000 toLINK the GIR to your  program,
    then DELETE
25 print"lines 63000 and on.
26 print"[down]Now the actual machinelanguage routine is   stuck to the
  end of
27 print"your BASIC program.
28 gosub 49
29 print"[clear]To use the VIC G.I.R.:
30 print"[down]SYSPEEK(45)+PEEK(46)*256-30 to link the GIR to the interr
  upt hand-ler.
31 print"[down]Select the features   you want and ADD the  required valu
  es into
32 print"location 155.
33 gosub 49
34 print"[clear]Example:
35 print"[down]To select latched in- put from the joystick
36 print"without diagonals:    [down]POKE155,1+128.
37 print"[down3]Joystick=1
38 print"[down]Diagonals=64
39 print"[down]Latched input=128
40 print"[down2]Diagonals are not pos-sible with keyboard   input.
41 gosub 49
42 print"[clear]The input will be     stored in locations   156,
    158 and 159.
43 print"[down]156 = fire button/key 0=off 1=on
44 print"[down]158 = vertical         0=up 1=center 2=down
45 print"[down]159 = horizontal       0=left 1=center       2=right
46 print"[down]Keyboard input:
```

```
47 print"[down]P=up L=left ;=right    .=down SHIFT=fire
48 end
49 print"[home,down22]HIT A KEY TO CONTINUE";
50 get a$:get a$
51 get a$:if a$=""then 51
52 return
```

G.I.R. BY T. HERCKENRATH

```
63000 I%=PEEK(46)*256+PEEK(45)+4
63001 READJ%:IFJ%>=0THENPOKEI%,J%:I%=I%+1:GOTO63001
63002 J%=I%/256:I%=I%-J%*256:POKE45,I%:POKE46,J%:CLR:END
63003 DATA 165,155,106,176,69,162,0,110,141,2,144,1,232,134,156,162
63004 DATA 1,165,197,201,13,208,7,134,159,202,134,158,240,41,201,21
63005 DATA 208,7,134,158,202,134,159,240,30,201,22,208,7,134,158,232
63006 DATA 134,159,208,19,201,37,208,7,134,159,232,134,158,208,8,36
63007 DATA 155,48,4,134,158,134,159,108,18,3,162,1,160,127,140,34
63008 DATA 145,160,255,44,32,145,140,34,145,48,1,232,173,17,145,44
63009 DATA 124,255,208,1,202,160,1,44,155,254,208,1,200,44,175,255
63010 DATA 208,1,136,44,202,255,208,4,169,1,208,2,169,0,133,156
63011 DATA 228,177,208,4,196,176,240,44,134,177,132,176,36,155,112,20
63012 DATA 224,1,240,16,192,1,240,12,165,159,201,1,240,4,162,1
63013 DATA 208,2,160,1,36,155,16,8,224,1,208,4,192,1,240,4
63014 DATA 134,159,132,158,108,18,3,173,20,3,141,18,3,173,21,3
63015 DATA 141,19,3,120,165,45,56,233,213,141,20,3,165,46,233,0
63016 DATA 141,21,3,88,96,-1
READY.
```

HE SAYS AS FAR AS HE IS CONCERNED, WE'VE BOTH BEEN REPLACED BY A COMPUTER.

Talk is Cheap for VIC

By Lee Urbanski, Madison, Wisc.

One of the joys of owning a VIC is finding new things to plug into the user and expansion ports. The newest thing that I have found to plug in there is a speech synthesizer for only $79.00 (U.S.). At that price I couldn't pass it by.

The Speakeasy is manufactured by Personal Peripheral Products of Aurora, Illinois. It is distributed by Protecto Enterprises of Barrington, Illinois.

The Speakeasy is a single printed circuit board that plugs into the expansion port of the VIC. It will drive any 8 ohm speaker directly from the phone jack on the PCB. It will drive the TV speaker if you open up the RF modulator and make the connection there. No external power or amplification is necessary.

On the PCB is a Votrax SC 01 IC. It can pronounce 64 phonemes with four levels of pitch control. Volume and overall pitch can be adjusted by turning two ports on the PCB. Volume cannot be controlled by software.

Speakeasy speaks in a male voice with a marked robot accent. It's the sort of thing you have to listen to for a while before it is readily understood. Once you become accustomed to it, its accent is quite intelligible.

I think that the accent is too thick to use it to teach phonics. Unless you want your kid to speak like a computer. Programming makes all the difference here. Misplace a few commas and it will sound like your Pakistani uncle.

Votrax claims that any English word can be pronounced. This is generally true. I've tried some Spanish phrases and it did all right with them too. It didn't fare too well with Tzutuhil (a modern Mayan dialect) but it should be able to speak any language that doesn't have too many glottal or blatantly non-English sounds.

Getting it to talk is pretty easy. It uses the 1K block of address space decoded by the 102 signal. It will not interfere with any other device unless that device also uses the 102 signal.

It can be programmed in BASIC or machine language. Simply Poke 38912 (9800 hex) with the value of the phoneme code you wish to use. It will pronounce that phoneme until you tell it to say something else. Unlike some other speech synthesizers (the Speakeasy kit from Netronics for example) there are no preprogrammed words or phrases. You must give it a separate phoneme for each sound you wish to hear.

The most intelligible speech comes when you let each phoneme continue for its optimal length. That length varies from 47 to 185 milliseconds. Now that sounds like a hassle but it's not. Speakeasy knows the optimal length for each phoneme, and there is a signal to indicate when Votrax is ready. When bit 7 of 38912 is 1 then Votrax is ready for a new phoneme. This translates into some easy BASIC. 100 If Peek(38912) = 128 Then 100.

Improper phoneme spacing will make it sound like a Kurdish rebel with a mouth full of marbles.

Individual pokes to Votrax take lots of valuable memory space from the unexpanded VIC. The most economical way to use Speakeasy is files or data statements. Speakeasy will not interfere with memory expansion but you'll need an expansion board. Files are the easiest way to go.

SOFTWARE AVAILABLE

This brings us to software. Also available from Protecto is a tape with four programs on it. There are disk and tape versions of an editor program and a reader program.

If you're serious about making your VIC talk, this could be the best $9.95 you've ever spent.

The editor program allows you to create, see, hear, edit and save files of phonemes up to 256 bytes long. That's about a paragraph.

The filereader programs allow you to listen to the files you created with the editor. These programs are also the only source of decimal values for the pokes. That comes in handy if you're like me, still thinking in decimal.

There is a tape and disk version of each program. Both programs are easy to use and well documented. The documentation and installation instructions that come with the Speakeasy are also quite adequate.

Though it doesn't speak as clearly as I would like it to, Speakeasy is intelligible. And it's a lot of fun. If talking computers turn you on, then go for it.

ROM RABBIT

By Mayland Harriman, Pt. Arthur, Tex.

The ROM RABBIT has evidently been available for a couple of years but has now been reduced ten dollars to $39.95 for the PET CBM and $19.95 for the VIC.

Here is the greatest thing to come along for the cassette user. This ROM allows you to SAVE or LOAD a cassette program which normally takes 3 minutes and 45 seconds in just 45 beautiful seconds!!!

The ROM RABBIT is available on cassette for the 3.0 ROM 2001 PET only and in ROM for all other PETS, CBMS and VICS. Beyond the terrifically speeded up SAVE or LOAD the ROM RABBIT gives 12 commands which can be executed in BASIC's direct mode and further allows any key on the keyboard to repeat if held down for 0.5 seconds.

The ROM RABBIT installs in a spare socket and is initialized every time the computer is turned on by typing: SYS 9*4096 and taken out of action by typing *K (for Kill the Rabbit).

To use the ROM RABBIT with a tape program you load the computer in the usual manner and then save by typing: *S "name" and it will go to the cassette in RABBIT speed and of course load just as fast. One of the most interesting abilities of the RABBIT is the VERIFYING a program....unlike in Commodore Basic you don't have to have the program in memory; just type: *V "name" . If the recording is OK the name of the program will be displayed in reverse format. If the recording is bad then the message CASSETTE ERROR will be displayed.

Another splendid advantage with ROM RABBIT saving is its ability to take into consideration the tape leader when saving a program on the first part of the cassette. Use *SS for a short leader cassette or *SL for a long one and it works perfectly.

Here are the 12 commands. All must have the * ahead of the letter:

*SS or *SL to save
*H Convert Hex to Decimal
*Go to PET monitor
*L to load
*D Convert Decimal to Hex
*Z Toggle lower case/versus graphic
*V to verify
*T Test RAM IC's
*E load and run
*G Go to ML at HEX XXXX
*K Kill the RABBIT

Programs SAVED in RABBIT format can only be loaded with RABBIT commands and conversely BASIC saved programs must be loaded by BASIC Command.

The ROM RABBIT is a lovely animal and does everything claimed in exactly the manner described in the neat eight-page booklet furnished. Shipment was in a couple of days, well packed and sent first class mail.

I highly recommend the ROM RABBIT from EASTERN HOUSE SOFTWARE, 3239 LINDA DRIVE, WINSTON-SALEM, N.C. 27106.

LEMON COMPUTERS INC.

Take A Load Off Your VIC

By Tony Davidson, Gananoque, Ont.

The address and data bus buffers on the 6502 microprocessor are capable of driving at least 130 pico farad of capacitance and 1 standard TTL Load. What does this mean? It means you can only have 1 TTL device input connected to each of these bus lines, otherwise you will overload the outputs. Overloading causes the internal buffers to sink too much current; this causes output voltage level problems such as the low logic level voltage being too high. If the internal bus buffers are forced to drive too much capacitance, due to long lines, external off board connections, etc., the rise and fall times of the output signals become too long. So exceeding the drive of the internal bus buffers generally means things will not work too well.

The address and data bus lines on the VIC 20 computer's expansion port connector come directly from the 6502 microprocessor, and are therefore subject to the above drive limitations. To overcome these drive limitations external bus buffers, usually TTL devices, should be added to any expansion board that requires increased drive.

Low power Schottky (LS) TTL devices are preferable for 2 main reasons. First, LS devices require less drive than standard TTL devices, typically 50 per cent less, and, second, they consume less power. As the address and data lines on the 6502 microprocessor are already connected to I.C.'s inside the VIC, their drive capability is lower than 1 standard TTL load. Therefore, the use of a buffer which requires low drive on an expansion board prevents overloading. As the VIC 20 has a limited amount of current available (500 mA MAX) to use on the expansion port, the low power consumption of LS devices makes them even more desirable. CMOS buffers could also be used as they require even less drive, and consume far less power than LS devices. However, CMOS devices do have problems driving capacitive loads, so whenever long lines must be driven it often pays to use an LS buffer. The LS buffer will drive any TTL or CMOS I.C.'s on your expansion board.

The type of buffers to use depends upon the application. Buffers are available in many different forms; the most common are the simple inverting and non-inverting 1 Input Gates (Fig. 1.). The non-inverting buffer would normally be used for most applications, as the output signal's logic state is the same as the input's. The drive capability of a digital logic I.C., such as a buffer, is called its fan out. The fan out of a typical TTL buffer is 30. This means it can drive 30 standard TTL inputs. It should be noted that some TTL inputs require more drive than others. The amount of drive required is called the fan in of a device. A typical TTL input has a fan in of 1. Therefore, if your buffer gate has a fan out of 30, and the devices connected to the output of the buffer gate all have a fan in of 1, you can connect up to 30 devices without overloading the buffer gate's output.

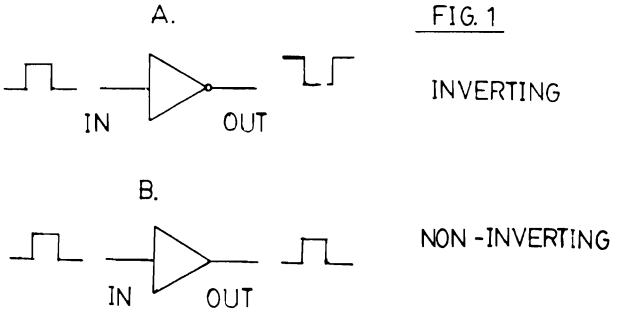


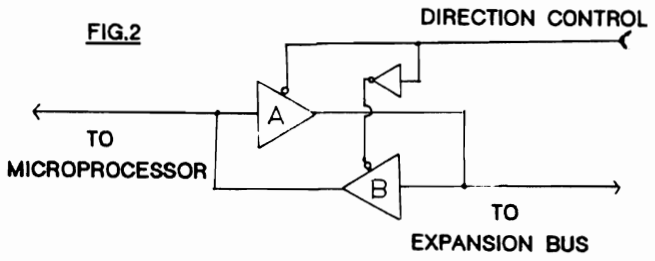
FIG. 1

These simple non-inverting buffers work well for the address bus, however, the data bus is slightly more complicated. The data bus is bi-directional, which means it is both an input and an output to the microprocessor and all other devices connected to the data bus. Therefore, to buffer a data line you require 2 buffer gates, one to allow data to be inputted from your expansion board to the microprocessor (Read), and one to allow data to be outputted from the microprocessor to your expansion board (Write). Special I.C.'s have been designed for this purpose; they are called bus transceivers. Most bus transceivers use Tri-State logic. The third state in Tri-State logic is a high impedance state. The other 2 states are the usually high and low logic conditions. All Tri-State devices have an enable control. When the enable control is active the Tri-State device behaves like an ordinary gate. When the enable control is inactive the output goes to a high impedance state which essentially disconnects the output from the bus. In this state there is virtually no loading on the bus from the Tri-State device.

On bus transceivers, the enable control is called the direction control. This is used to tell the transceiver which set of buffers will be active. Figure 2 shows one pair of buffers connected as they would be in a bus transceiver. If a low level signal is applied to the Direction Control, Buffer A is enabled allowing data to be sent from the microprocessor to the data bus. This low level signal is inverted to a high level signal before be-

ing applied to Buffer B, therefore, Buffer B is inactive and in a high impedance state. If a high level signal is applied to the Direction Control then Buffer B is active and Buffer A is in a high impedance state. In this mode data is transferred from the data bus to the microprocessor.

FIG.2



Internally in the VIC 20, Commodore uses a 74LS245 OCTAL TTL bus transceiver (Fig.3) to buffer the data lines going to the internal 2114 RAM memory devices. This particular transceiver has both a Direction Control (Pin 1) and an Enable Control (Pin 19). The Enable Control is active low, which means a low logic level signal must be applied to the device in order to turn it on. Once enabled, a low logic level signal applied to the Direction Control allows data to travel from bus B to bus A. A high level logic signal applied to the direction control allows data to travel from bus A to bus B. If the Enable Control is held high the device goes into a high impedance state.

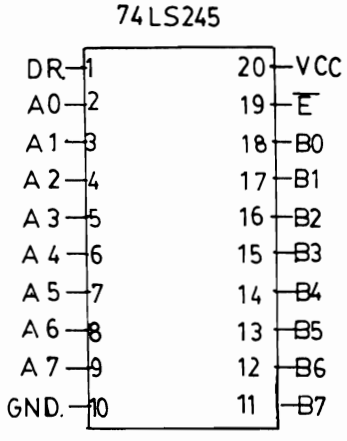


FIG. 3

To use this device on an A Expansion Board, connect bus B to the data lines on the expansion port connector of the VIC, D0 To B0, D1 To B1 etc. Join bus A to the Data bus on your expansion board, D0 To A0, D1 To A1 etc. Then connect the VR/W line (Read/Write) to the Direction Control. When the VR/W line is high the microprocessor is reading data, bus A To bus B, when low the microprocessor is writing data, bus B To bus A.

The Enable Line could be grounded if you wished. This would mean the transceiver would always be active. To overcome this you could use a select line to turn the transceiver on and off. The expansion port connector on the VIC 20 has 4 fully decoded select lines, one for each unused block of VIC memory. All of these select lines are active low (Table 1).

PIN *	SIGNAL	ADDRESS
10	$\overline{\text{BLK 1}}$	2000 - 3FFF
11	$\overline{\text{BLK 2}}$	4000 - 5FFF
12	$\overline{\text{BLK 3}}$	6000 - 7FFF
13	$\overline{\text{BLK 5}}$	A000 - BFFF

TABLE 1

Suppose that you are building a PROM programmer and you have decided to locate it at ADDRESS A000 hex. If you connect Select Line BLK 5 to the Enable Control, the transceiver will only be enabled during a Read or Write to an Address contained Block 5. Therefore, the transceiver will be in a high impedance state at any other time.

The 74LS245 can also be used to buffer the address lines by connecting bus B to the address lines on the VIC expansion connector, and bus A to the address lines on your expansion board. If you then ground the direction control the transceiver would operate as a buffer transmitter only. This may seem to be a waste of half of the device; however, the 74LS245 is relatively inexpensive and it also has the advantage of keeping all the buffer I.C.'s on your expansion board the same. This allows for easier construction and trouble-shooting of the board.

As there are 14 Address Lines on the expansion port connector, you will need 2 74LS245's to buffer them all. This will leave 2 buffer gates unused. I suggest they should be used for buffering any control lines you may require on your expansion board, such as the S02 Clock. The Enable Control can be connected in the same manner as the data bus transceivers enable control.

You may note that Commodore does not buffer the expansion bus on their 8K and 16K RAM expansion cartridges. This is because they use CMOS memory devices which require far less drive than TTL devices.

If you are planning on building some kind of expansion board for your VIC, check to see if it will overload the microprocessor. If so, install bus buffers and take a load off your VIC.

VIC-20 Modifications

By Dennis Sievers, Breeze, IL

The VIC 20 represents one of the most versatile small computers available for both the beginner and the advanced programmer. Though many desirable features are included in the machine, a few simple modifications can be easily made to enhance the use of this computer.

One of the easiest and most useful modifications to a VIC 20 is the addition of a reset switch. It is fairly easy to install such a switch in several locations. The easiest and most convenient is between pins 1 and 2 of the 555 timer IC. This is a small 8-pin DIP IC located at UB 6 on the main board (see photo 1). The upper two pins on the left are numbers 1 and 2. By connecting these to a normally open pushbutton switch, a reset button can be created. This reset button enables the user to gain control of the computer should a program accidentally crash and cause the computer to hang. The advantage of this reset switch is that the power supply is not constantly turned off and on during a session, and thus saves on wear and tear on this component. In addition, if the following line of code is entered in direct mode prior to running the program, it is possible to not only reset the computer but to recover any program in memory.

```
PRINT  PEEK(4097),PEEK(4098),PEEK(45),PEEK(46)
```

These locations are for the start and end of program pointers. Record these results before the program is run. Should a program crash, press the reset button and poke these values into their respective memory locations.

A second modification is the result of having used a CBM 8032. The presence of a numeric keypad makes machine language entry very easy, yet is lacking from the VIC. Several surplus stores have keypads ranging from ten-key numeric to 29-key calculator pads that offer even greater utility as mini keyboards or hex keypads.

The keyboard connector of the VIC 20 is a single-sided, 18-pin edge card. By use of a slightly oversized 24/44-pin edge card connector, a convenient connector can be fashioned. A piece of PC board is used to make a tight fit between the pins used by the keyboard and the unused portion of the connector.

The PC board is cemented in place using epoxy or similar suitable material. Wires are then attached from this connector to a suitable connector to be placed outside the computer. I had a spare DB 50 connector and used it, but a DB 18 would work just as well. A hole is cut in the VIC case and the connector mounted with either epoxy or small bolts.

The keypad should be composed of normally open switches as is the VIC keyboard. Using Table II, it is possible to wire the keypad to a male connector compatible with that installed in the VIC. According to Table II, a closed circuit must exist between the indicated pair of wires to form a character on the screen. Ribbon cable would be preferable to join the keypad to its connector; however, any type of stranded wire can be used.

Depending on the type of keypad selected and how it is wired, one can construct a ten-key numeric pad, a calculator type pad with return or a hex pad with return or any combination of the user's choice. The cost of this project can be as great as $15.00 if you must purchase all parts, or as little as nothing if your junkbox is well-stocked. Your own creativity and junk box will determine the final cost.

You should also be aware that the above alterations will most certainly void any warranty and make your computer easier to operate.

TABLE I

RESET A PROGRAM

1. Program entered
2. PRINT PEEK(4097),PEEK(4098),PEEK(45), PEEK(46)
3. Record the above numbers and addresses.
4. Run program (if computer hangs then 5.).
5. Press reset.
6. Poke values into addresses from 2. above.
7. List program and make corrections.

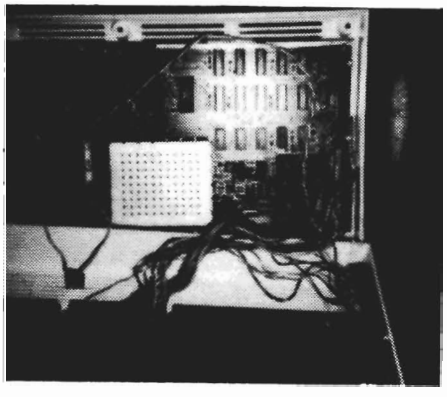
Then there was the old joke about the ethnic computer that used BASE 1 arithmetic.

— Ylimaki

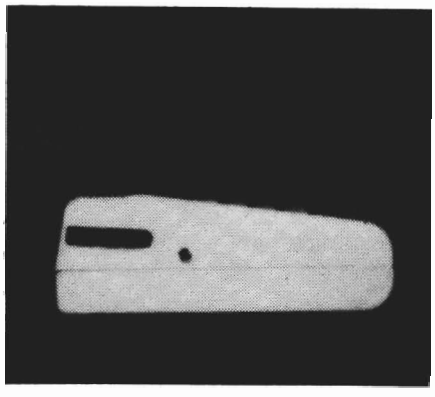
TABLE II
KEYBOARD MATRIX

WITH WHITE STRIPE

	Grey	Purple	Blue	Green	Yellow	Orange	Red	Brown
Brown	F7	home		0	8	6	4 4	2
Red	F5		@	0	u	T	E	Q
Orange	F3		:	K	H	F	S	commodore
Yellow	F1	rt.shift	·	M	B	C	Z	space
Green	csr	/	,	N	V	X	lt.shift	run/stop
Blue	csr	;	L	J	G	D	A	ctrl
Purple	return	*	P	I	Y	R	W	
Grey	del	E	+	9	7	5	3	1



The 555 timer is shown just to the left of the large central box. Pins 1 and 2 are indicated by arrows.



Shown here is an end view of a VIC 20, showing the installation of the reset button (to right of centre) and the keypad connector (on left).

TABLE III
KEYPAD PARTS LIST

1 18-pin or larger edge card connector
1 DB or larger male connector
1 DB 18 or larger female connector
1 keypad of choice (see Table IV)
4' 18-strand ribbon cable or comparable length of stranded wire

TABLE IV
SOURCES OF KEYPADS

All Electronics Corp.
905 S. Vermont Ave.
Box 20406
Los Angeles, CA 90006

Jameco Electronics
1355 Shoreway Rd.
Belmont, CA 94002

Best of The TORPET VIC

Redesigned 8K Card
By Dennis Sievers, Breeze, IL

One of the first pieces of equipment usually added to the VIC 20 is additional memory. Many times, this is in the form of the 8K RAM cartridge. This is a most valuable asset for the memory-poor VIC. While offering added power, even this card can be improved. Two very simple modifications are shown here to make the card even more useful and easier to use.

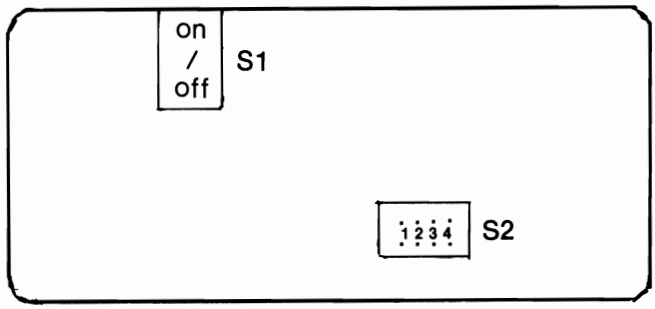
The RAM card can reside in any of the various expansion blocks of the VIC by changing the setting of the four-place DIP switch inside the case. Taking the case apart constantly to change the settings can be a source of many problems. As an alternative, I cut a small hole in the top of the case using a 6mm drill and small file at a point shown in Figure 1. This opening is then covered with clear tape to prevent the entry of dust, and at the same time allow the user to know the exact placement of the switches. When the card needs to be moved to another block, the tape is removed, the switches set and the hole re-covered.

A second useful modification is the installation of a write-protect switch. A common SPST switch is installed in the VR/W line (pin 17) of the 8K RAM cartridge. To do this, open the case and locate pin 17. Trace the lines to a small solder pad and install a #28 wire in the pad hole. The other end goes to one of the switch contacts. Turn the card over and locate the printed circuit line from this pad to one located approximately 14cm above. Check to ensure that the proper line has been isolated by using an ohm meter between the pad and pin 17. If the circuit conducts, the proper line has been found and the second solder pad located. The line between these two pads must now be cut using a thin knife. Use caution to cut only one line, and be certain that the line has been completely severed.

A second #28 wire is installed between the second solder pad and the other switch contact. When the switch is open, the information that has been poked into the card will remain active and available, even upon reset.

The above modifications will void any warranty on the card, and may also void any warranty on the computer as well. All such modifications should be carefully checked before actual trial in a computer.

FIGURE I
8K RAM CARD MODIFIED

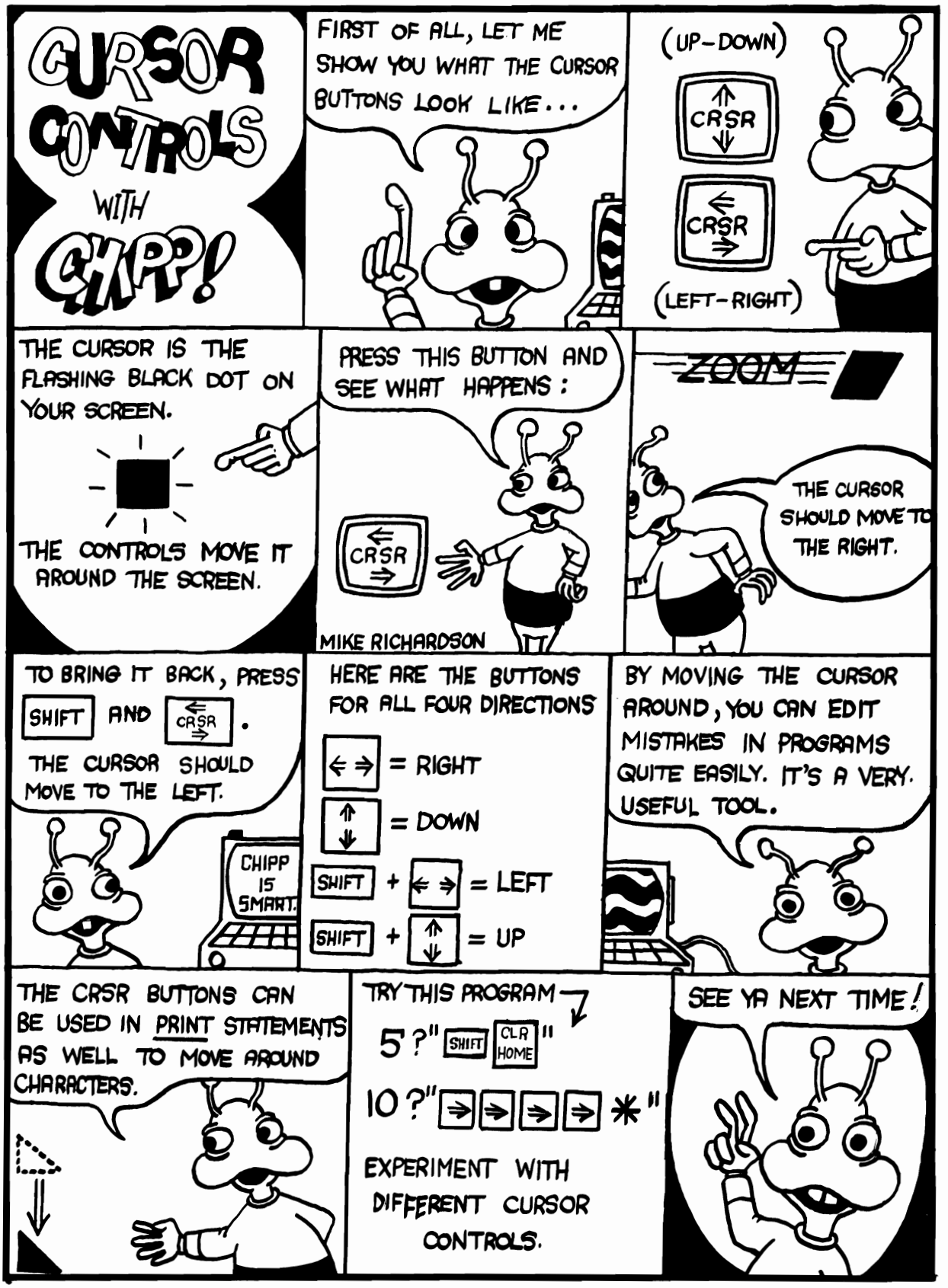


S1 = SPST switch for write protection
S2 = 4-place DIP switch for block selection

Approximate location of switches in card case.

WORD PROCESSING is a little like wallpapering. In both, you CUT and PASTE, and try to keep your margins straight. They are both easy once you get the HANG of it.

— Ylimaki


CURSOR CONTROLS WITH CHIPP!
FIRST OF ALL, LET ME SHOW YOU WHAT THE CURSOR BUTTONS LOOK LIKE...
(UP-DOWN)
CRSR
CRSR
(LEFT-RIGHT)
THE CURSOR IS THE FLASHING BLACK DOT ON YOUR SCREEN.
THE CONTROLS MOVE IT AROUND THE SCREEN.
PRESS THIS BUTTON AND SEE WHAT HAPPENS:
CRSR
MIKE RICHARDSON
ZOOM
THE CURSOR SHOULD MOVE TO THE RIGHT.
TO BRING IT BACK, PRESS SHIFT AND CRSR.
THE CURSOR SHOULD MOVE TO THE LEFT.
CHIPP IS SMART.
HERE ARE THE BUTTONS FOR ALL FOUR DIRECTIONS
= RIGHT
= DOWN
SHIFT + = LEFT
SHIFT + = UP
BY MOVING THE CURSOR AROUND, YOU CAN EDIT MISTAKES IN PROGRAMS QUITE EASILY. IT'S A VERY USEFUL TOOL.
THE CRSR BUTTONS CAN BE USED IN PRINT STATEMENTS AS WELL TO MOVE AROUND CHARACTERS.
TRY THIS PROGRAM
5 ?" SHIFT CLR HOME "
10 ?" * "
EXPERIMENT WITH DIFFERENT CURSOR CONTROLS.
SEE YA NEXT TIME!

General Programming

PAGE

Tower of Babel 60
Herbert Gross, Elgin, IL
A plea for standardization among computer languages.

Tips for Good Programming 61
Robert Dray, Peterborough, Ont.
Here are some very good tips for the beginning programmer. Start right and you will never go wrong.

Structured Listings for Commodore 64 62
Charles Kluepfel, New York, N.Y.
How to "Pretty Up" your listings. A program that does the job for you.

Structured Programming 67
Elizabeth Deal, Malvern, PA
Not everyone is enthused about GOTO-less programming. Here is a short introduction to what all the debate is about.

BASIC: Structured or Unstructured 69
Marg McRitchie, Winnipeg, Man.
How to write structured programs: the concepts of top-down design, style coding and modularity explained.

Tower of Babel

By Herbert Gross, Elgin, Ill.

Fortunately, man did not create computers in his own image. While we can design them to look like us and program them to do tasks in a similar manner to us, the differences are profound...except in one area. For both computers and man there are enough language differences to hinder intra-species communication.

Even among Commodores, which are largely based on the same circuits, by the same manufacturer, the same problem exists. Programs for different PETs will not RUN on each other unchanged. They will have even more difficulty in RUNning on VICs, 64s or the new B series. I recently read an article about a program made to translate Chinese into English. It is ironic that such a program, designed to translate such diverse languages, has to RUN on a specific model of a specific brand of machine.

What I would really like to know is whether the translation of ideagrams from Chinese into English is easier than the translation of variables, values and operations through electrical impulses from DEC, IBM or Apple into Commodore? Perhaps because I am a novice, I take an over-simplistic view of complexities. It still seems to me that, after setting aside operations of color and sound, anything that one computer can put on one screen, another computer can put on another screen. Any operation performed by any other computer can probably be performed by a PET or 64. Though the 'techniques', for want of a better word, and even the technology may be different, the basic operations would be the same.

A translation program is basically a comparison and substitution of one value or operation for another. This is something computers should be able to be programmed to do faster, and more accurately, than man. The major difficulties to be overcome would be programs that required more memory than the second machine had. Even this could be compensated for, to an appreciable extent, by using some kinds of interactive mass storage techniques.

Let's get back to sound and color. Sound capabilities can be added to virtually any computer without undue difficulty or expense. The add-on might not have the richness of true sound synthesis, but would probably be sufficient for 99% of games, household, business and research applications. Color would simply have to be translated to monochromatic shades.

Photographic artists have been working in black and white long after color was easily available. Newspapers usually use approximate shades of grey in photos by the spacing of black and white dots. This is virtually identical to the turning off and on pixels in high resolution graphics.

WE HAVE TO CHANGE

There has to be a change in thinking on the part of some programmers. Usually, and often with good reason, most programmers' efforts are directed toward protecting and restricting access to their programs. One of these days, though, someone will come along, do the opposite and become wealthy. Designing a program to translate yours and others' programs from one machine to another can greatly increase your income. First, it could double income from previously successful machine specific programs. It could double income from future programs. In addition, the translation program itself could be sold or licensed. Aside from monetary factors, it would improve the whole industry. After all, someone who composes a song or writes a book doesn't have to worry about the manufacturer or brand name and model number of the printing press, record or phonograph.

Whoever is the first to do it, please hurry!! Our local school district has just purchased some Apples, we have a 64 and one of my child's friends has a TRS80. It would be so great if they could all learn together and see their own programs running on each other's machines.

Tips for Good Programming

By Robert Dray, Peterborough, Ont.

Often in our programs we will ask for input such as the name of the user or some other data. The program will then jump to the next section as soon as the user presses the RETURN key.

From the user's point of view, this often seems rather abrupt. The program can be much more "user friendly" by making the machine say "Thank you". This message need only appear on the screen for 1 to 2 seconds and then have the program continue, but this is enough to make the user feel that the machine "appreciates" his action, and therefore the user is put at ease.

Humans do not react as quickly as the computers and the judicious use of time delays can greatly enhance an interactive program. One must avoid the tendancy to place time delays where other methods would work better.

It is often necessary to print instructions on the screen, and as often happens, they won't all fit on a single screen. It is tempting to try something cute here such as using a time delay or printing out each word separately and having the text eventually scroll off the top of the screen. These methods will work fine as long as you are accurately able to judge the reading ability of the user. If the user is a slower reader, or has to sneeze, the material may pass by too quickly and the user is left with a bad feeling about the program.

A better way to handle this situation is to have a little routine that lets the user control when the next material is to be presented. This routine and the time delay routine can be set up as subroutines and can be activated with the GOSUB command. Here is a sample of each routine:

```
2000 REM TIME DELAY
2010 FOR T = 1 TO TD*1000:NEXT
2020 RETURN

3000 REM USER CONTROLLED DELAY
3010 PRINT"PRESS ";CHR$(34);"C" ;CHR$(34);"To CONTINUE
3020 GET G$: IF G$ "C" THEN 3020
3030 RETURN
```

The time delay is called with a line such as:

```
50 TD = 2: GOSUB 2000
```

The value of TD determines the number of seconds (approximately) for the delay, and this may be varied each time the routine is called.

The CHR$(34)'s in the other routine are necessary to place the letter C in quotation marks. The line following the GOSUB 3000 command will often start with a clear the screen command and then the new material is presented.

The REM statement is used to add documentation to a program, which means that it makes the program LISTing easier to read. These REM statements are completely ignored by the computer as it executes the program, and thus they are only used by the human who is looking at the LISTing. Some people go overboard and will put one or more REM statements for each line of program, but this can make the program harder to read than having no REMarks at all. If the program is separated into blocks which perform specific tasks, as outlined in earlier articles in this series, then one or two brief REM statements is usually sufficient to tell what that block of code is doing. A calculation that may not be obvious is also a prime candidate for a REM statement, such as the following:

```
250 FIN V*(1 + 1/100)*Y X) :REM finds value of investment.
```

Another useful application is to outline loop structures as shown:

```
200 REM LOOP
210 FOR I = 1 TO 25
220 PRINT "WHAT IS THE NUMBER";
230 INPUT NUM(I)
240 NEXT I
250 REM ENDLOOP
```

Whether an automatic FOR-NEXT loop is used, or a conditional loop using an IF-THEN statement, the use of the REM statement makes the loop more visible in the listing and thus the program will be easier to understand.

```
200 REM LOOP
210 PRINT "PLEASE GIVE A POSITIVE NUMBER"
220 INPUT NUM
230 IF NUM ► 0 THEN 210
240 REM ENDLOOP
```

This is a simple example and obviously there could be many lines of code between lines 210

and 230, but the entire loop structure would stand out on the page as a unit. This will make debugging the program easier as well as trying to figure out, 6 months from now, what the program is doing.

You could also include in your programs: a list of variables and what they stand for, your name and address, what the program does..... Anything that helps to make the program a little easier to understand is fair game for REM statements.

Best of The TORPET Programming — General

Structured Listings for Commodore 64

By Charles Kluepfel, New York, N.Y.

This program is on The Best Programs Disk

Listing 1 has two parts, both listings of the same program. Which is easier to read and understand? ...the second one, of course. The spaces that make reading easier, but take up precious program space, left out of the program listed at the top, are inserted into the listing at the bottom. That listing also shows only one statement per line, indents FOR...NEXT loops for easier reading, indents conditional statements dependent on an IF...THEN, and shows the ASCII values of unknown characters (this on a non-Commodore printer). Line 90 was stored with some wild spacing, even putting a space between the two characters of a two-character variable name. The formatted listing corrects all this. The decimal code for PI that the C64 sends out to the printer is 255 or hex FF. My Epson printer interprets this as an instruction to delete the preceding character: thus, the last two characters in quotes in line 70 do not show up at all, as the last tells the printer not to print the next to last. The formatted listing shows the last two characters correctly. The same thing happened on line 80 X = PI, with the PI being spelled out as [PI] on the formatted one. Note that within quotes PI is listed as [255], but outside of quotes it is a token that translates to [PI]. See Appendices E and F of the Commodore 64 User's Guide for an explanation of ASCII codes. The Cardco printer interface on the first listing did rescue some otherwise unprintable codes, but who would know that the italic comma at the end was a CHR$(172); the formatted listing tells you just what is there. Note that NEXT J,I is expanded, showing in parentheses the imagined second NEXT, as the statement is equivalent to NEXT J: NEXT I, but NEXT J,NEXT I would be invalid syntax; the parentheses say "do not type in that word NEXT".

The concept of making programs easier to read has been called "structured programming", and one of the techniques used is indentation for readability.

In May of 1980, a program called LIST FORMATTER appeared in Call-APPLE magazine, written by Mark Capella. I have been using what I consider an improved version of that program to list my Apple programs. When I got a Commodore 64, I felt the need even more strongly to have such a program. An Apple in a normal LIST at least inserts spaces around key words so that all the spaces that it automatically strips away from within input program lines, conserving program space, are provided where necessary to separate words (no two non-key words ever appear consecutively, so all words are separated by at least one space). Accustomed as I am not to type spaces, as they are ignored by the Apple anyway (and they take up precious space in large C64 programs), I felt a strong need to have a C64 version of the LIST FORMATTER. So, I re-wrote it for the Commodore and gained all its benefits.

Four blank lines are printed for each 62 lines of actual printing, thus skipping over perforations. Note that, to get the proper standard format for an IF ... THEN with just a line number, either use IF ... THEN line number, or IF ... THEN GOTO line number; do not use IF ... GOTO line number in your program to be listed; such a line would be listed on one line. When listing to the video screen rather than a printer, the [and] (braces) appear as graphic symbols of a plus sign or cross and a vertical bar, which then surround special symbol designations.

HOW DO YOU USE THE PROGRAM?

First load the program you wish listed into memory. Then type the following:

```
POKE 251,PEEK(43):POKE 252,PEEK(44)
```

terminating the line with a carriage return. Then type

```
POKE 43,PEEK(45):POKE 44,PEEK(46):NEW
```

again, ending with RETURN. Be sure you do not make any mistakes, as you are changing the pointers to the beginning of BASIC program memory space. Then LOAD the LIST FORMATTER program previously saved to tape or disk, and RUN it.

You will be prompted TESTING,VIDEO:? The reply is to be two numbers, usually 0,0. If, however, you want to have the listing appear on the TV screen, make the second number non-zero, say type 0,1. Making the first number (testing) non-zero produces a listing that contains some inside information on the locations of the program in memory, and is rarely used. Try it if you would like to learn more about how BASIC is stored internally, and if you can follow the program listing.

If you selected video by making the second number non-zero, you will soon see the listing form on the TV screen. If, however, you specified the second number as zero so that you would get printed output, you will be prompted for some further information first. In response to TITLE:? enter the title of the program followed by RETURN. Then, in response to NAME:? type in your name. In response to DATE:? type in today's date, but remember, do not use a comma or colon in these entries, as they are each input via an INPUT statement, and these characters would cut off the data being input. To the prompt SECONDARY ADDRESSES:? you should respond with two numbers separated by a comma. Usually, a secondary address of 0 indicates the normal upper-case-only mode, while 7 indicates upper-and-lower-case mode. The first of the two secondary addresses is for the heading (title, author's name, listing. I usually use the 7 for the former, while the latter depends on whether the program being listed was written for upper/lower case. If using 7 for the titles, before responding to the TITLE:? prompt, press Comodore-Shift so that the characters go to lower case on the screen; then use the shift key to enter the headings as you want them printed; be sure not to have the shift down or locked on when pressing RETURN at the end of any input.

Once you have typed in the two secondary addresses, separated by a comma, the listing will start. If you allow it to complete, it will print COMMENTS: at the bottom, thus labelling the final blank portion of a page. You will get a READY back and the program in memory will be the one that had been being listed, and not the FORMATTER program.

If you wish to interrupt the listing while it is still in progress, you have two choices. If you press the STOP key, the LIST FORMATTER program will still be in memory, and you may even modify it and then RUN it again; the listing will start over. Before restarting, press STOP/RESTORE and type CLOSE 4 to assure that the print file is closed. If, however, you decide that you want the listing to stop and you want the program in memory to revert to that one being listed, press the F1 key instead of the STOP key. That will cause a premature end of the LIST FORMATTER as soon as the line being printed at tht moment has finished printing, and the subject program will be the only one in memory.

The following is an example of a program listed both normally and using the LIST FORMATTER program, printed on the following page.

LISTING 1 — UNFORMATTED

```
10 FORI=5TO25STEP5:FORJ=1TO5
15 IFI=JTHENNEXTJ,I:GOTO50
20 PRINTI,J;
30 PRINTI+J
40 NEXTJ,I
50 FORY=1TO30:PRINTY;:IFY<15THENPRINTSQR(Y);
60 PRINT:NEXT
70 PRINT"{BK}{WH}{RD}{CY}{PU}{GR}{BL}{YL}{RV}{RO}{OR}{BR}{LR}{G1}{G2}
   {LG}{LB}{G3}{??}{??},"
80 X
90 J = 30+ 5*X Y

   READY.
```

LISTING 1 — FORMATTED

```
10    FOR I = 5 TO 25 STEP 5 :
          FOR J = 1 TO 5
15            IF I = J THEN
                  NEXT J,
                  (NEXT) I :
                  GOTO 50
20            PRINT I,J;
30            PRINT I + J
40        NEXT J,
      (NEXT) I
50    FOR Y = 1 TO 30 :
          PRINT Y; :
          IF Y < 15 THEN
              PRINT  SQR (Y);
60        PRINT  :
      NEXT
70    PRINT "{BLK}{WHT}{RED}{CYN}{PUR}{GRN}{BLU}{YEL}{RVS}{RVSO}{ORNG}
      {BRWN} {LTRD}{GR1}{GR2}{LTGRN}{LTBLU}{GR3}{1}{2}{172}{165}{255}"
80    X =  {PI}
90    J = 30 + 5 * XY
```

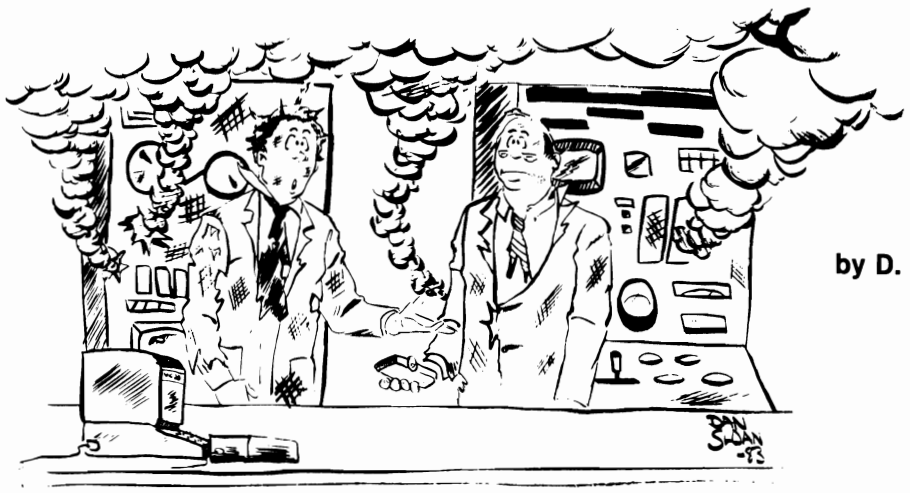
LISTING 2

Listing 2 is the actual LIST FORMATTER program listed normally, except for the [F1] in lines 7003 and 7103. They should be replaced with the actual F1 key character.

```
140 input"testing,video:";te,vi
150 gosub1000:gosub2000:gosub3000
160 ifvi=0thenprint:print:print:print"comments: "
170 ifvi=0thenprint#4:close4
180 poke45,peek(43):poke46,peek(44):poke43,peek(251):poke44,peek(252):clr:end
1000 rem set up tokens
1030 iftestingthenprint"setting up tokens"
1040 dimtkn$(128):fori=1to75:readtkn$(i):next
1045 tkn$(128)=chr$(123)+"pi"+chr$(125)
1050 dimrp$(255)
1060 fori=5to31:readrp$(i):next:fori=129to159:readrp$(i):next
1070 return
1130 data  end,for,next,data,input#,input,dim,read,let,goto
1140 data  run,if,restore,gosub,return,rem,stop,on,wait,load
1150 data  save,verify,def,poke,print#,print,cont,list,clr,cmd
1160 data  sys,open,close,get,new,tab(,to,fn,spc(,then
1170 data  not,step,+,-,*,/,^,and,or,>
1180 data  =,<,sgn,int,abs,usr,fre,pos,sqr,rnd
1190 data  log,exp,cos,sin,tan,atn,peek,len,str$,val
1200 data  asc,chr$,left$,right$,mid$
1300 data wht,,,disc,ensc,,,,rtn,lc,,,csrd,rvs,home,del,,,,,,,,red,csrr,grn,blu
1310 data orng,,,,f1,f3,f5,f7,f2,f4,f6,f8,srtn,uc,,blk,csru,rvso,clr,inst,brwn
1320 data ltrd,gr1,gr2,ltgrn,ltblu,gr3,pur,csrl,yel,cyn
2000 rem get headings
2030 ifvi=0theninput"title: ";tt$:input"name  : ";fi$:input"date  :";da$
2035 ifvi=0theninput"secondary addresses: ";sa(1),sa(2)
2040 return
```

```
3000 rem print the listing
3050 ifvideothenprint"■"
3060 ifvi=0thenopen4,4,sa(1):cmd4:print:printtt$:printfi$spc(60-len(tt$))da$:lc=3
3065 ifvi=0thenprint#4:close4:open4,4,sa(2):cmd4
3070 iftestingthenprint"main line."
3080 in=0:tn=0:li$="":srf=0:pb=peek(252)*256+peek(251)
3081 nb=peek(43)+peek(44)*256-pb-1
3090 gosub4000:ifeop=0then3090
3100 return
4000 rem print one numbered line
4040 lp=0:tn=0:qf=0:rf=0:nx=0:df=0
4045 ifpsthengosub7000
4050 gosub6000:x=by:gosub6000:x=by*256+x:nb=nb-2:iftestingthenprint"memptr= "x
4060 iftestingthenprint"nb= ";nb
4070 ifnb<1theneop=1:return
4080 gosub6000:x=by:gosub6000:lne=x+by*256:nb=nb-2:iftestingthenprint"line #= "lne
4090 iftestingthenprint"nb= ";nb
4100 ifnb<1theneop=1:return
4110 gosub6000:nb=nb-1:ifby=0thenreturn
4114 ifby<128then4135
4115 iftk$(by-127)="rem"andsrfthengosub7100
4120 iftk$(by-127)="rem"andsrf=0thengosub7100:ct=0:srf=1:goto4140
4130 iftk$(by-127)<>"rem"orsrf=0thengosub7100:ct=0:srf=0:goto4140
4135 gosub7100:ct=0:srf=0
4140 printright$("     "+str$(lne),5);:ct=5:goto4160
4150 gosub6000:nb=nb-1:ifby=0thenct=0:return
4160 lp=lp+1
4165 ifct>79thengosub7000
4170 ifby>127andqf=0thengosub5000:goto4150
4180 ifby=32andqf=0andrf=0anddf=0then4150
4200 ifct<8+in+tnthenprintspc(8+in+tn-ct);:ct=8+in+tn
4220 iflp=1andrf=0andtf=0thenprintspc(8+in+tn-ct):ct=8+in+tn
4224 iflp<>1orrf>0ortf=0then4230
4225 ifby>=48andby<=57thenprintspc(8+tn+in-ct)"goto ";:lp=lp+5:ct=8+in+tn+5
4230 ifby=asc(":")andqf=0thenprint" ";:nx=0:ct=ct+1
4235 ifct>79thengosub7000
4240 iflp=1andtfthenprintspc(8+in+tn-ct);:ct=8+in+tn
4250 iflp=1andby=32then4150
4253 ifby>31andby<96or(by>31andby<128orby>191andby<224)and(sa(2)>=6)then4262
4255 ifrp$(by)>""then4259
4256 s$=mid$(str$(by),2):ifct>79-len(s$)-1thengosub7000
4257 printchr$(123)s$chr$(125);:ct=ct+len(s$)+2:goto4270
4259 ifct>79-len(rp$(by))-1thengosub7000
4260 printchr$(123)rp$(by)chr$(125);:ct=ct+len(rp$(by))+2
4261 goto4270
4262 printchr$(by);:ct=ct+1:ifct>79thengosub7000
4270 ifby=34thenqf=1-qf
4275 ifct>79thengosub7100
4276 ifby<>asc(",")ornx=0then4280
4277 iftn=0orli$="f"thenin=in-4
4278 lp=1:gosub7100:printspc(8+in+tn)"(next) ";:ct=8+in+tn+7
4280 ifby<>asc(":")orqfthen4150
4285 df=0
```

```
4290 gosub6000:nb=nb-1:ifby=0thengosub7100:ct=0:return
4300 ifby=asc(":")then4160
4310 gosub7100:lp=1
4320 goto4170
5000 rem print a token's meaning
5030 tf=0
5040 iflp>1thenprint" ";:ct=ct+1
5045 ifct>79thengosub7000
5050 iftk$(by-127)="next"thennx=1:iftn=0orli$="f"thenin=in-4
5060 iflp=1thenprintspc(8+in+tn-ct);:ct=8+in+tn
5065 ifct>79-len(tkn$(by-127))thengosub7000
5066 ifct<8+in+tnthenprintspc(8+in+tn-ct);:ct=8+in+tn
5067 iftk$(by-127)=""thenprintchr$(123)mid$(str$(by),2)chr$(125)" ";:ct=ct+5:
     goto5074
5070 printtkn$(by-127)" ";:ct=ct+len(tkn$(by-127))+1
5074 ps=0:iftn>0then5080
5075 iftk$(by-127)="goto"ortk$(by-127)="return"ortk$(by-127)="end"thenps=1
5076 iftk$(by-127)="run"ortk$(by-127)="stop"thenps=1
5080 iftk$(by-127)="for"thenin=in+4:li$="f":return
5090 iftk$(by-127)="then"thengosub7000:tn=tn+4:lp=0:tf=1:li$="t":return
5100 iftk$(by-127)="rem"thenrf=1
5105 iftk$(by-127)="data"thendf=1
5110 return
6000 rem get next byte of program
6030 iftestingthenprint"*";
6040 by=peek(pb):pb=pb+1:return
7000 print:lc=lc+1:iflc>61andvi=0andte=0thenprint:print:print:print:lc=0
7003 getx$:ifx$="[f1]"then170:rem f1
7004 ifvi=0thencmd4
7005 printspc(8+in+tn);:ct=8+in+tn
7010 return
7100 print:lc=lc+1:iflc>61andvi=0andte=0thenprint:print:print:print:lc=0
7103 getx$:ifx$="[f1]"then170:rem f1
7104 ifvi=0thencmd4
7105 ct=0
7110 return
```



by D. Sloan

What do you mean "Back to the old drawing board?" We blew that up too!

Structured Programming

By Elizabeth Deal, Malvern, PA

A man went to his doctor to find out why he had been suffering terrible pain. After some tests, he was told that the X-ray revealed a serious condition that needed surgery. The man asked how much it would cost. "About ten thousand dollars," the doctor replied. The man said, "Doctor, I can't afford that kind of treatment. I am poor and have no health insurance. What can you do for me for fifty dollars?"The doctor thought about this for a few minutes and said, "Well, I could touch up these X-rays."

STRUCTURED PROGRAMMING

A commonly-given description of "structured programming" is one that says: "you can indent lines of code and it has no GOTO statements". It's technically correct, of course, but is only partly true. I think it steers beginners onto a totally wrong course.

Several months ago, I saw just such a description of structured programming in a major computer magazine, and I wondered then how much damage it could do. Now I know.

Recently, I saw a paper for educators which encouraged them to teach structured programming. For pages and pages, it extolled the virtues of structured programming: it described our normal programming efforts as being a "rat's nest of branches, full of illogical thinking" and other such unpardonable sins. Among other things, this paper showed an example of unstructured coding. The program had many GOTOs, with branches crossing all over the place, like this:

```
100 ...
110 IF  PLUS  GOTO 200
120 IF  MINUS  GOTO  300
150 GOTO 400
200 A = A + B:GOTO400
300 A =  A-B:GOTO 400
400 GOTO 100
410 ...
```

To cure the problem and to demonstrate the power of structuring, the paper proceeded to draw a structured flowchart, whatever that is. Branches that crossed over were neatly replaced by little circles, or "termination points", like this (fig. 1):

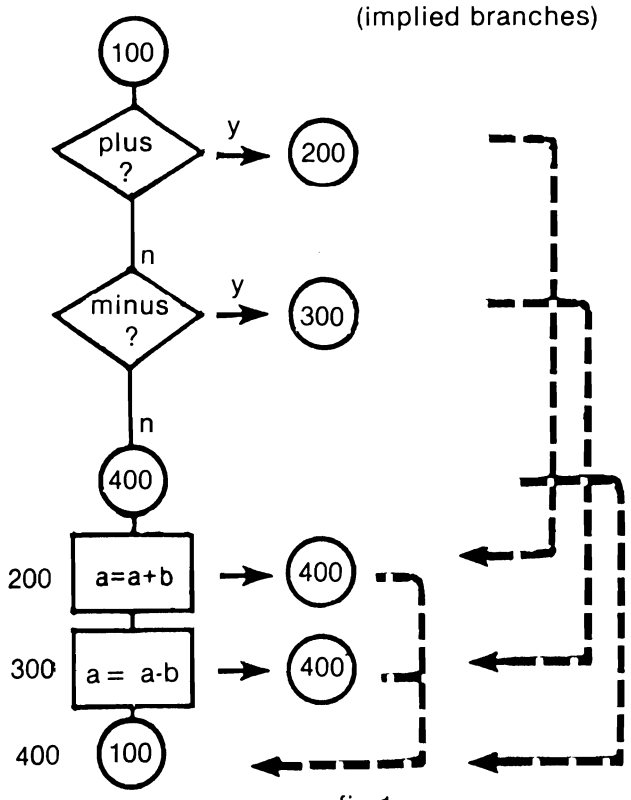


fig.1

To complete the job and to compound the confusion, most GOTO statements were carefully replaced by THEN, so that the program now read:

```
100 ...
110 IF  PLUS  THEN 200
120 IF  MINUS  THEN 300
130 and so on
```

The parting conclusion of this effort was "we now have a GOTO-less structured program".

In my humble opinion, we now have NOTHING. Replacing GOTO by THEN, a statement that does the same thing only slower, and replacing crossing branches by terminator points is as useful as touching up the X-rays: the mess is still there.

Clearly, this paper missed the point of structuring entirely: about seeing a large task as a collection of small, perhaps already solved, steps, about desirability of coding tiny units which can be easily debugged, about using flags, the condition of the machine, as a basis for decisions, about using arrays or sub-routines in general, about nesting of loops and branches if such must exist, and so on. Instead, the paper played a semantic, substitute-

the-keywords game, never offering simple code such as:

```
100 ...
110 IF PLUS THEN A = A + B
120 IF  MINUS  THEN A = A-B
130 GOTO 100
```

which can be cut neater by some Boolean functions anyway. Its uncluttered flowchart could have been (fig.2):

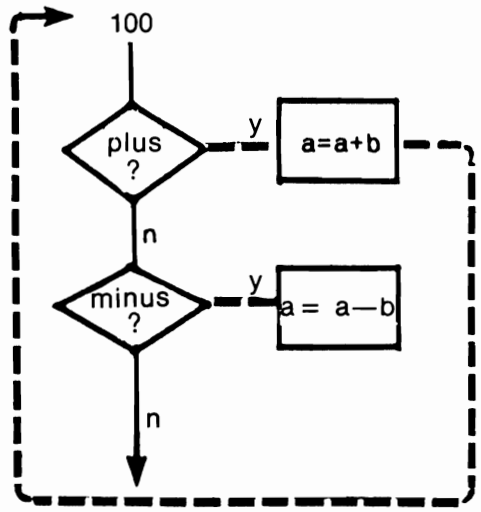


fig.2

I do not really intend to take part in or describe the raging debate over structuring, but the concepts of real structured programming are worth knowing indeed. The following are excellent sources, if you're interested in what all the fuss is about:

1. Various papers by Bohm and Jacopini, Dijkstra and others in the field are available in many libraries.

2. Paul Nagin and Henry Ledgard, BASIC WITH STYLE, PROGRAMMING PROVERBS, Hayden Book Company Inc., 1978.

3. Brian W. Kernighan, P.G. Plauger (Bell Labs, Inc.), THE ELEMENTS OF PROGRAMMING STYLE, McGraw Hill Book Co., 1974

4. Seymour Paper, MINDSTORMS: Children, Computers and Powerful Ideas, Basic Books, Inc., 1980

BASIC PERSPECTIVE

BASIC, or machine code for that matter, is difficult to do without branches (GOTOs), but many can be eliminated if we follow this pragmatic principle: code GOSUB when you can, code GOTO if you must.

But keep in mind that this is only a part of the structuring story. While an apparent goal of structuring is reducing unnecessary GOTO statements, the thought behind the exercise is a better program organization for purposes of readability and easier maintenance in the future. As the code is being re-organized, many GOTOs vanish from sight without any conscious effort. So, in a sense, it sometimes becomes a circular process of one thing helping another, an interesting process indeed.

I often wonder if the structuring fanatics haven't dug their own grave by having overshadowed their sensible campaign for orderly, logical program design with their loud anti-GOTO pronouncements. It leaves the audience with a misleading impression that the entire debate turns on the presence or absence of GOTO and GOTO alone...and we end up with keyword substitution as a cure.

DEBUGGING TROUBLES

It has crept up slowly, but I began taking notice recently of many programmers abandoning GOTO. I see a lot of THEN123 type coding, and begin to wonder why. It's a nasty, inconvenient, little switch in style. Basic-Aid type of utilities which help search through a program become difficult to use.

When a program contains GOTOs, and you are interested in "transfer of control" to know if somebody's line 100 is a GOTO/GOSUB target, this type of utility can easily plop all the simple GOTOs on the screen.

But when GOTO100 is coded as THEN100, we're in a terrible predicament of having to weed out genuine THEN-action (as in THENV = 15:K = 1) from THEN-line number. Often a program contains so many THEN-action statements that zeroing in on THEN-line number becomes an unnecessarily tedious task. Frankly, I fail to see a reason for ever substituting THEN for GOTO.

PLEASE, DO NOT TOUCH UP THE X-RAYS!

I think it's most convenient to code THEN for a process to happen and to code GOTO for transfer to another place in a program. The computer doesn't care, but it can make work much simpler for us.

BASIC: Structured or Unstructured

By Margaret McRitchie, Winnipeg, Man.

Although structured BASIC has been around for over 10 years, it is just recently that it has come into prominence. At present, programmers are split into two distinct groups: those who prefer structured programs and those who do not. On the fringes are many who are either unfamiliar with structured programming, or may be unsure which method to use. The purpose of this paper is merely to show the differences between the two styles of programming, and to point out the strengths and weaknesses of each.

Structured programming is simply a style of coding. It incorporates three simple concepts:

1. A top-down design.
2. A style of coding that is conducive to the reading of programs.
3. Modular if the program is long and routines contain detailed instructions.

TOP-DOWN DESIGN

Top-down means that statements are executed sequentially from the top of the program to the bottom. Control should not be transferred to a previous statement unless the branch develops into a loop. In such a loop, the controlling statement (usually an IF/THEN statement) should be the first statement in the loop or as near the top of the loop as possible.

STYLE OF CODING

To simplify the reading of programs, REM statements, blank lines and indentions are used extensively. GOTO statements are discouraged, although it is difficult to exclude them entirely.

MODULAR

Modules usually consist of a main routine and one or several sub-routines. The program retains the top-down design since control is transferred to the statement immediately following the GOSUB that called the sub-routine.

On most computer systems, structured BASIC must be simulated. This is because the statements that lend themselves to structured programming, such as REPEAT, WHILE...DO, and IF/THEN...ELSE (with the word ELSE on a separate line) are not supported by many computers. Thus, the change from unstructured to structured does not require a major overhaul of your resource material, but merely a change in the style of programming. The statements used in structured programs are the ones that are used in unstructured programs. No new statements are introduced. It should be mentioned that even the structured style of coding differs among programmers.

Shown below is a problem for which two programs are written: one program is unstructured; the other program is structured.

A retailer of fine furniture pays employees a basic salary of $200 a week and a commission based on sales. The rate of commission is graduated depending on the amount of the sales. The employees receive 20% commission on sales of $1000 and 23% commission on any excess sales. Find the incomes of three salespersons whose sales volumes are $900, $1000 and $1500.

UNSTRUCTURED

```
10 LET S1 = 200
20 LET B = 1000
30 READ S
40 DATA 900,1000,1500,-1
50 IF S = -1 THEN 999
60 IF S►1000 THEN 90
70 LET C = S*.2
80 GOTO 100
90 LET C = B*.2 + (S-B)*.23
100 LET I = S1 + C
110 PRINT
120 PRINT "SALES:";S
130 PRINT "COMMISSION:";C
140 PRINT "INCOME:";I
150 GOTO 30
999 END
RUN

SALES: 900
COMMISSION: 180
INCOME: 380

SALES: 1000
COMMISSION: 200
INCOME: 400

SALES: 1500
COMMISSION: 315
INCOME: 515
```

STRUCTURED

```
01 REM--FIND INCOME OF 3 EMPLOYEES
02 REM S1--SALARY
03 REM B--BASIC SALES
04 REM S--SALES
05 REM C--COMMISSION
06 REM I--INCOME
07 :
10 LET S1 = 200
```

```
20 LET B = 1000
30 READ S
40 DATA 900,1000,1500,-1
50 REM START LOOP
60 ::IF S = -1 THEN 110
70 ::::GOSUB 200 : REM--SUB COMM,INCOME
80 ::::GOSUB 300 : REM--SUB DISPLAY
90 ::::READ S
100 :GOTO 50
110 REM END LOOP
120 END
190 :
200 REM--SUB COMM,INCOME
210 ::IF S►B THEN 240
220 ::::LET C = S*.2                    : REM F
230 ::::GOTO 250
240 ::::LET C = B*.2 + (S-B)*.23 : REM T
250 ::REM END IF
260 ::LET I = S1 + C
270 RETURN
290 :
300 REM--SUB DISPLAY
310 ::PRINT
320 ::PRINT "SALES:";S
330 ::PRINT "COMMISSION:";C
340 ::PRINT "INCOME:";I
350 RETURN
RUN

SALES: 900
COMMISSION: 180
INCOME: 380

SALES: 1000
COMMISSION: 200
INCOME: 400

SALES: 1500
COMMISSION: 315
INCOME: 515
```

Although it is possible to control a loop more conveniently using FOR/NEXT statements, an IF/THEN statement is used in each program to demonstrate more clearly the differences between the two methods of coding.

The three features of structuring mentioned at the beginning of this article are incorporated in the structured program in a number of ways.

1. The program is well-documented by REM statements that identify the program and the variables.

2. Line numbers are segregated into groups. Each range is designated for a specific purpose.

a) REM statements are in the 1 to 9 range
b) The main routine has statements numbered in the 10 to 180 range
c) The first sub-routine has statements in the 200 range
d) The second sub-routine has statements in the 300 range.

3. A blank line separates routines. A blank line is coded as a line number and a colon. In this program, a blank line is left between the block of REM statements and the main routine, and a blank line is left before each sub-routine.

4. Two READ statements enter the data into the computer. One READ statement is placed before the loop; the other READ statement is placed inside the loop. The first READ statement initializes S before the loop begins. The second READ statement changes the value of S each time the computer passes through the loop.

5. REM statements mark the beginning and the ending of the loop.

6. The IF/THEN statement in line 60 and the GOTO statement in line 100 are indented several spaces. The statements executed inside the loop are indented even further.

7. The test that terminates the input of data is usually placed at the beginning of the loop.

8. A routine that contains detailed instructions and which is used several times during the course of the program is placed in a sub-routine. The trailing REM statement coded after the calling GOSUB identifies the sub-routine that is being called.

9. The first statement in a sub-routine must be a REM statement which describes the sub-routine's task. The statements inside the sub-routine are indented.

10. When a program contains an IF/THEN statement that determines which of two routines must be executed, trailing REM statements are coded as follows:

a) A trailing REM T is coded opposite the first statement of the routine that is executed when the condition in the IF/THEN statement is true.

b) A trailing REM F is coded opposite the first statement of the routine that is executed when the condition in the IF/THEN statement is false.

11. In sub-routine 200, the statements within the IF/THEN block are indented several spaces. A REM statement marks the end of the IF/THEN block. It was mentioned at the beginning of this paper that even structured programs can vary in style. These differences occur more often in an IF/THEN block in which one of two routines must be executed. Here are two other ways by which sub-routine 200 can be written.

a)
```
200 REM--SUB COMM,INCOME
210 ::IF S►B THEN 250
220 ::::REM F
230 ::::::LET C = S*.2
240 ::::::GOTO 270
250 ::::REM T
260 ::::::LET C = B*.2—(S-B)*.23
270 ::REM END IF
280 ::LET I = S1 + C
290 RETURN
295 :
```

b)
```
200 REM--SUB COMM,INCOME
210 ::IF S►B THEN 230
220 ::::GOTO 250
230 ::::::LET C = B*.2 + (S-B)*.23
240 ::::::GOTO 270
250 ::::REM ELSE
260 ::::::LET C = S*.2
270 ::REM END IF
280 ::LET I = S1 + C
290 RETURN
295 :
```

In sub-routine (a), the REM statements that identify the TRUE and FALSE routines are placed on separate lines, just before the routines begin. In sub-routine (b), a REM ELSE is coded just before a FALSE routine. A GOTO statement must transfer control to this REM statement.

The strengths and weaknesses of each style of program are listed below.

UNSTRUCTURED PROGRAMS

STRENGTHS

The unstructured program is much shorter. This is due mainly to the non-existent REM statements. For students who do not know how to type, this is probably an advantage. For students who know how to type, the extra time spent entering these statements may be negligible. Less planning (and thus less time) is probably involved in developing an unstructured program. Coding is simplified by omitting the indentations. Most statements will fit on a 40-character line, since character positions are not required for the indentations.

WEAKNESSES

An unstructured program is often difficult to read, particularly when the program is long and complex. The many GOTO statements make it difficult to keep track of variables and their values. Altering the program, such as changing variables, adding routines and deleting routines, is more involved than altering a structured program.

STRUCTURED PROGRAMS

STRENGTHS

A program is easier to trace when its variables have been identified. Stating a sub-routine's purpose opposite the calling statement identifies the sub-routine being called. When the same identification is coded in the first line of the sub-routine, the sub-routine is easier to locate. Isolating routines into separate modules is helpful for several reasons.

1. The main routine is not filled with detailed instructions which may detract attention from the main purpose of the program.

2. It is much easier to add and delete routines, since they are isolated into separate modules.

3. Altering a program is simplified. A change is usually restricted to a module.

A loop is easier to find when it starts and ends with REM statements. Indenting statements inside an IF/THEN block and within a sub-routine shows that the statements belong to that particular routine. Structured programs will probably be better understood because of the advance planning and thought that must go into the program before it is written.

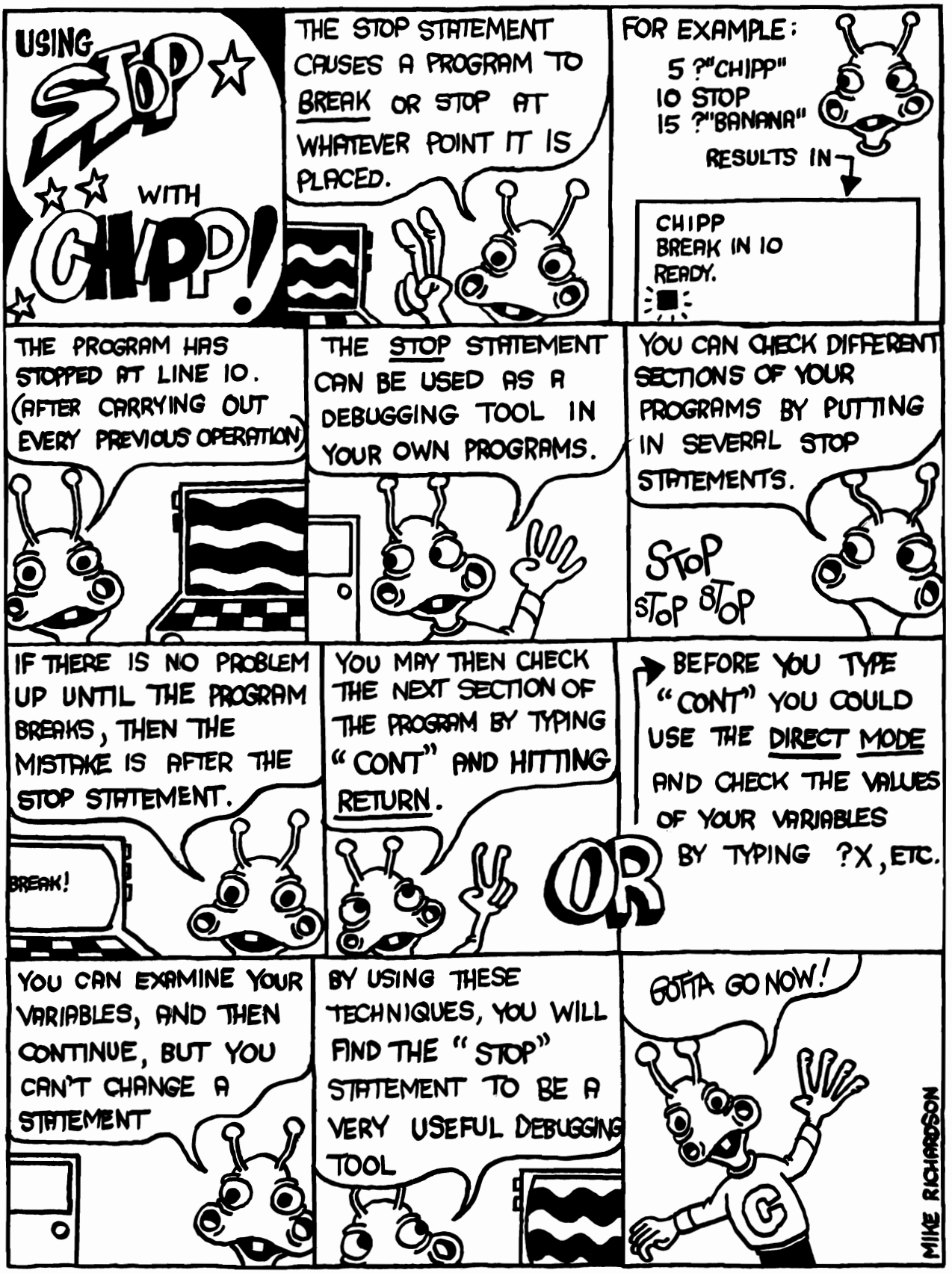
WEAKNESSES

Structured programs are generally longer. For students who do not know how to type, keying in a program may be quite a task. Because of the many REM statements, and possibly a few ELSE statements, programs tend to have an untidy appearance. When a REM ELSE statement is coded in an IF/THEN block, an extra GOTO statement is required. The use of too many GOTOs is discouraged in structured programming. Students may become confused as to where and when to indent. Furthermore, the indentations, on many computers, are not retained when the program is listed. To rectify this problem, colons can be placed where spaces are to appear. The many character positions that are left blank for indentations may cause some statements to become too long for a 40-column screen. However, this problem does not arise when the computer has an 80-column screen.

SUMMARY

Should you decide that you would like to try structured programming in your classes, there is no need to panic. All of your resource material, as well as the text you are currently using, can be retained. Structured programming is merely a different style of coding. This means that your programs and those written by your students must simply be changed in coding style. However, sub-routines should be introduced early in the course. Fortunately, this concept is easy to learn and students generally have little difficulty understanding how control is transferred to and from a sub-routine. Keep the concept simple. Calling one sub-routine from the main routine is probably all you will need.

For those who were unfamiliar with structured programming, and for those who were undecided as to which method to use, hopefully this paper will help you choose the group to join — structured or unstructured.


USING STOP WITH CHIPP!
THE STOP STATEMENT CAUSES A PROGRAM TO BREAK OR STOP AT WHATEVER POINT IT IS PLACED.
FOR EXAMPLE:
5 ?"CHIPP"
10 STOP
15 ?"BANANA"
RESULTS IN
CHIPP
BREAK IN 10
READY.
THE PROGRAM HAS STOPPED AT LINE 10. (AFTER CARRYING OUT EVERY PREVIOUS OPERATION)
THE STOP STATEMENT CAN BE USED AS A DEBUGGING TOOL IN YOUR OWN PROGRAMS.
YOU CAN CHECK DIFFERENT SECTIONS OF YOUR PROGRAMS BY PUTTING IN SEVERAL STOP STATEMENTS.
STOP
STOP STOP
IF THERE IS NO PROBLEM UP UNTIL THE PROGRAM BREAKS, THEN THE MISTAKE IS AFTER THE STOP STATEMENT.
BREAK!
YOU MAY THEN CHECK THE NEXT SECTION OF THE PROGRAM BY TYPING "CONT" AND HITTING RETURN.
OR
BEFORE YOU TYPE "CONT" YOU COULD USE THE DIRECT MODE AND CHECK THE VALUES OF YOUR VARIABLES BY TYPING ?X, ETC.
YOU CAN EXAMINE YOUR VARIABLES, AND THEN CONTINUE, BUT YOU CAN'T CHANGE A STATEMENT
BY USING THESE TECHNIQUES, YOU WILL FIND THE "STOP" STATEMENT TO BE A VERY USEFUL DEBUGGING TOOL
GOTTA GO NOW!
MIKE RICHARDSON

BASIC Programming

PAGE

Friendly Menu 74
M. Petersmeyer, Sao Paulo, Brazil
Let's make those programs user friendly. Here is a simple routine that is a step in the right direction.

Underlining Cursor 75
Peter Tattersall, Rexdale, Ont.
Some routines to improve the underlining cursor.

Blinking Prompt 77
T. H. Holmer, Ashland, Mass.
This little routine will make a message blink on the screen.

The Line-Number Speed Fallacy 78
David Williams, Toronto, Ont.
There are some myths in the computer world. This article puts one of them to rest.

The Lazy Programmer 79
Jim Butterfield, Toronto, Ont.
Some things to do in BASIC programming that will make your life simpler.

Reading Between The Lines 80
David Williams, Toronto, Ont.
Here is a short tutorial for the beginning programmer on some of the benefits in using data statements.

Input Idiosyncrasies 82
Jim Butterfield, Toronto, Ont.
Having trouble getting the input to work right sometimes? Big Jim explains how.

A Tiny PET/C64 Print Hint 83
Elizabeth Deal, Malvern, PA
Here is a little routine to help you in printing those frames around your screen.

Friendly Menu

By M. Petersmeyer, Sao Paulo, Brazil

This program is on The Best Programs Disk

Frequently, a program being executed will present a list of options (or menu) on the screen and request that the user select one of them. A typical menu display might look like the following example:

1. Add new names
2. Delete names
3. Update records
4. Print labels
5. — Quit —

Enter desired function (1-5)?

The program can then use the numerical response for conditional branches such as ON X GOTO YYY or ON X GOSUB YYY. The conventional menu routine has two disadvantages:

(a) the option number selected by the user must be verified to ensure that it falls within an allowable range. In the example above, entering a negative number or a number greater than 5 could result in an indignant or unwelcome response from the program.

(b) it is easy for the user to accidentally enter a wrong number. (Oops! I pressed 5, not 4.)

Friendly Menu is a subroutine which highlights one menu selection at a time in reverse video, using keys of your choice to move this 'cursor' line up or down. When the desired option is highlighted, pressing RETURN exits the subroutine with the corresponding option number in the variable Z1, making it unnecessary to check if the selection is acceptable. But it is the prominent display of one item at a time plus the ability to choose which keys will move the 'cursor' that make it more user-friendly; thus the name 'Friendly Menu'.

Although it was written for the Commodore PET, I imagine Friendly Menu could be adapted to other microcomputers supporting reverse video.

NOTES:

Lines 10-2020 are a sample driver program to illustrate the operation of the subroutines.

Line 110 — SP$ is a shifted space (PETASCII CHR$(160)), since PET's INPUT statement strips normal spaces (CHR$(32)) from the front of a string. This character is not required if the menu text starts at the left-hand edge of the screen.

Line 130 — PO$ is used to position the cursor on the desired line of the screen (1 to 25), like VTAB in other BASICs.

Line 150 — FIRST is the screen line number where the first menu item is to be printed. SP is spacing (single, double, etc) which defaults to single spacing in line 50000.

Line 50010 — opens PET's screen as an input device, so the statement INPUT #1 (line 50040) will take its input from the screen beginning at the prevailing cursor position.

Line 50080 — defines the keys used to move the highlighted 'cursor' line up and down. I selected the PET's cursor-up/cursor-down keys to perform their usual functions. The unshifted cursor-across key was added as an alternative to cursor-up, thus eliminating the need to use two fingers (SHIFT plus cursor-up) to move the highlighted 'cursor' up.

Lines 51000-51030 are optional, and can be used while printing the menu on the screen. The purpose of this subroutine is to set the values of ROW and LAST, which are subsequently needed for the input subroutine at line 50000.

```
10 GOTO 100
15 :
20    ****************************
25    * FRIENDLY MENU            *
30    *                          *
35    * BY:                      *
40    * MARK PETERSMEYER         *
45    * RUA ALBERTO FARIA, 1118  *
50    * SAO PAULO, S.P.          *
55    * BRAZIL  05459            *
60    *                          *
65    * AS OF AUGUST 20, 1982.   *
70    *                          *
75    ****************************
80
100 PRINT" □":SPC(13);" ■ FRIENDLY MENU"
110 SP$ = CHR$(160):REM SHIFTED SPACE
120 :
130 PO$ ="■■■■■■■■■■■■■■■■■■■■■■■■■":REM  HOME  +
    25  CURSOR  DOWN
140 :
150  FIRST = 5:SP = 2:ROW = FIRST-SP:REM  LOCATE  FIRST
LINE OF MENU AND SET SPACING
160 :
170 GOSUB 51000:PRINT SP$;SPC(12);"OPTION ONE"
180 GOSUB 51000:PRINT SP$;SPC(12);"OPTION TWO"
190 GOSUB 51000:PRINT SP$;SPC(14);"-END-"
200 :
210 GOSUB 50000
```

```
220 :
230 ON Z1 GOSUB 1000,1000,2000
240 :
250 GOTO 100
260 :
1000 PRINT "    YOU SELECTED OPTION NUMBER";Z1
1010 FOR I = 1 TO 1000:NEXT I
1020 RETURN
1030 :
2000 END
2010 :
49900 REM *** INPUT SUBROUTINE ***
49910 :
49920 REM VARIABLES REQUIRED: FIRST, LAST, (SP), PO$
49930 REM VARIABLES USED: IZ, ROW, ZZ$, Z$, Z1
49990 :
50000 IF SP = 0 THEN SP = 1
50010 OPEN 1,3
50020 FOR IZ = FIRST TO LAST STEP SP
50030 :ROW = IZ:PRINT LEFT$(PO$,ROW);
50040 :INPUT#1,ZZ$
50050 :ZZ$ = "  " + ZZ$:PRINT "  ";ZZ$
50060 :GET Z$:IF Z$ = "" THEN 50060
50070 :IF Z$ = CHR$(13) THEN 50170
50080 :IF (Z$ <  > "  " AND Z$ <  > "  "AND Z$ <>  "  ") THEN
50090 :PRINT "  ";MID$(ZZ$,2)                                  50060
50100 : IF (Z$ = "  "OR Z$ = "  " THEN 50140
50110 :IF IZ + SP > LAST THEN IZ = IZ-SP
50120 :GOTO 50150
50130 :
50140 :IZ = IZ-2*SP:IF IZ < FIRST-SP THEN IZ = FIRST-SP
50150 NEXT IZ
50160 :
50170 CLOSE1
50180 Z1 = (ROW-FIRST + SP)/SP
50200 RETURN
50210 :
50900 REM *** PRINT MENU; SET ROW & LAST ***
50910 :
50920 REM REQUIRES FIRST, PO$, (SP)
50930 REM USES FIRST, LAST, ROW
50990 :
51000 IF SP = 0 THEN SP = 1
51010 ROW = ROW + SP:LAST = ROW
51020 PRINT LEFT$(PO$,ROW);
51030 RETURN
51040 :
READY.
```

Best of The TORPET BASIC Programming

Underlining Cursor

By Peter Tattersall, Rexdale, Ont.

This program is on The Best Programs Disk

Bill Crimando of Carbondale, Illinois, wrote (HELP, TORPET #22) about changing the cursor character from a flashing block to a solid underline. While this is not a difficult undertaking, there are a couple of points that should be made.

FIRST, there is no 'cursor character' as such. The cursor is placed on the screen by toggling the high bit for the corresponding screen position. This, in turn, causes the VIC II chip to select a different character, normally a reversed character, rather than the character edited with the cursor on. Changing the cursor to a line rather than a block involves changing the character set by eliminating reverse field characters.

SECOND, the cursor flashes as a result of actions taken by the interrupt routine at $EA31. Eliminating the flashing requires modification of that routine, and must be done in machine language.

CHAR-SETUP carries out these tasks. It reserves memory for a new character set, copies the old character set, and modifies the reversed characters so as to produce an underline cursor. It then pokes in a new interrupt routine entry and enables it so that the cursor is always on. FLASHKILLER is the assembly language form of the interrupt entry. As shown, it sits at $8000, but it can be relocated to reside anywhere in free RAM. A further routine, ENABLER, simply enables FLASHKILLER.

NOTE that CHAR-SETUP reduces the amount of BASIC memory available. This is due to a fundamental restriction of the VIC II chip, which requires that the screen and character set be located in the same 16k block, unless some special handling is done. For the Commodore 64, the special handling is such that, for blocks 0 and 2, the second 4k sub-block of the block is 'strapped' to point to the ROM character set. If we want to place a full user-defined character set in block 0 or 2, we must use either the lowest 4k sub-block or the upper two. For most purposes, this works out as in figure 1.

From this, it can be seen that the only space available outside of BASIC memory (block 3, at $C000), will require the screen to sit in space already occupied by ROM and I/0, or in space occupied by the KERNAL. This would require that we re-write the kernal routines.

Placing the full character set at $8000 is no better since this requires the screen to sit in an area already occupied by BASIC ROM — no great problem if we aren't using BASIC, but not possible otherwise. The ideal (?) solution places the user-defined character set at $2000, and moves the bottom of BASIC up to $3000. While this is not dif-

ficult, it is a little awkward to do from inside a BASIC program and is better done from the keyboard or by a machine language program. As a result, this program places the character set at $7000, and puts the screen at $6800. The memory areas $0800-$Offf and $8000-$9fff, formerly screen and BASIC RAM respectively, are now available for other purposes, and are lost to BASIC.

I have assumed that the full double set of characters is to be implemented, so that a full 4k of RAM (2k for each complete character set) will be required. If we only require one character set, our problem is solved before we go any further, since we can use $C000 to hold the screen and one set of characters. This is left as an exercise for the reader....

FLASHKILLER is a machine language routine (see listing) which duplicates part of the interrupt processing, does some processing of its own, and jumps into the normal interrupt routine so that all the interrupt tasks are completed. CHAR-SETUP pokes the routine into memory at $8000, since space is available, and calls ENABLER, which adjusts the pointers to the interrupt routine. In principle, the pointers could have been reset from BASIC by disabling the interrupt routines in the same way as in line 150 of CHAR-SETUP. The machine language instructions SEI and CLI give less chance of trouble: the only reason they were not used in CHAR-SETUP was for purposes of demonstration.

One last point to consider is that this style of cursor is found on IBM 327x terminals, which may switch between flashing and permanent cursors and between block and underline cursors. An interesting exercise is to further modify the keyboard scan routine to use the user-definable keys to permit switching among these four possible cursor styles.

FLASHKILLER

Machine Language Routine to kill flash of the cursor

```
8000 20 ea ff   jsr UDTIM       ; update timer
8003 a5 cc      lda BLNSW       ; check if cursor is enabled
8005 d0 13      bne $801a       ; nope — do rest of routine
8007 a4 d3      ldy PNTR        ; get line pointer
8009 bl dl      lda (PNT),y     ; get the current character
800b 29 7f      and #$7f        ; disallow the high bit
800d 85 ce      sta GDBLN       ; save it
800f 09 80      ora #$80        ; turn on the high bit
8011 91 d1      sta (PNT),y     ; and put character on screen
8013 a9 01      lda #$01        ; fake blink on
8015 85 cf      sta BLNON       ; to fool other parts of system
8017 4c 4f ea   jmp $ea4f       ; that's all, folks
801a 4c 61 ea   jmp $ea61       ; skip cursor part of interrupt
```

ENABLER

A Routine to turn on FLASHKILLER

```
80ld 78         sei             ; disable interrupts
80le a9 00      lda #$00        ; low-byte of FLASHKILLER
8020 8d 14 03   sta CINV        ; interrupt pointer low
8022 a9 80      lda #$80        ; high-byte of FLASHKILLER
8024 8d 15 03   sta CINV + 1    ; interrupt pointer high
8025 58         cli             ; re-enable interrupts
8026 60         rts             ; and that's all there is.
```

FIGURE 1

Available Locations for character sets

BLOCK	SUB-BLOCK	ADDRESS	RESTRICTION
0	0	$0000	System use
	1	$1000	Not available
	2	$2000	Available (BASIC RAM)
	3	$3000	Available (BASIC RAM)
1	0	$4000	Available (BASIC RAM)
	1	$9000	Not available
	2	$A000	BASIC ROM
	3	$B000	BASIC ROM
2	0	$8000	Available (BASIC RAM)
	1	$9000	Not available
	2	$A000	BASIC ROM
	3	$B000	BASIC ROM
3	0	$C000	Available
	1	$D000	ROM & I/O
	2	$E000	KERNAL ROM
	3	$F000	KERNAL ROM

Note that the full double set of characters (switched using the shifted-COMMODORE key) is assumed. If a user-defined character set is placed in Blocks 1, 2 or 3, the screen must also be moved to the same block.

```
10 REM **********************
20 REM *
30 REM * CHAR-SETUP
40 REM *
50 REM * BY
60 REM *
70 REM * PETER TATTERSALL
80 REM *
90 REM **********************
100 :
101       :     POKE52,104:POKE56,104:CLR:REM
RESERVE MEMORY
102 :
103 REM  TURN ON CHARACTERS
104 :
105 : POKE53272,(PEEK(53272)AND240) + 14
106 :
110 REM  WE'LL DO ALL THIS IN BASIC
120 REM  SO YOU CAN SEE WHAT WE'RE DOING
130 :
140 REM  FIRST TURN OFF THE INTERRUPTS
145 REM  WHICH IS OK FROM BASIC AS LONG
146 REM  AS YOU DON'T HAVE BUS CARDS,
147 REM  OR (SOME) BASIC EXTENSIONS.
148 :
```

```
150 : POKE  56334,PEEK(56334)AND254
151 :
152 REM  RESET  THE  BLOCK  TO  1
153 :
154 : POKE56578,PEEK(56578)OR3:POKE56576,(PEEK(56576)
      AND2- 52)OR2
155 :
156 REM  RESET  THE  PAGE  POINTER
157 :
158 : POKE648,104:POKE53272,(PEEK(53272)AND15)OR160
159 :
160 :
170 REM  SWITCH  OUT  I/O  AND  SWITCH  IN
180 REM  CHARACTER  ROM
190 : POKE  1,PEEK(1) AND 251
200 :
210 REM  MOVE  CHARACTERS  TO  $3000
220 :
230 :  FORI = 0TO4095
240 : POKEI + 28672,PEEK(I + 53248)
250 : NEXT
260 :
270 REM RESTORE  I/0
280 :
290 : POKE  1,PEEK(1)OR4
300 :
310 REM  RESTORE  INTERRUPTS
320 :
330 : POKE 56334,PEEK(56334) OR 1
340 :
350 REM  MODIFY  "BLOCKS"
360 :
370 : FOR  I  = 1024 TO 2047
380 : POKE  28672 + I,PEEK(27648 + I)
390 : NEXT
400 :
410 : FOR  I  = 1031 TO 2047 STEP 8
420 : POKE 28672 + I,255
430 : NEXT
440 :
450 : FOR  I  = 3072 TO 4095
460 : POKE 28672  +  I,PEEK(27648  +  I)
470 : NEXT
480 :
490 : FOR  I  = 3079 TO 4095 STEP 8
500 : B = 255AND(NOT(PEEK(28672 + I)))
510 : POKE 28672 + I,B
520 : NEXT
530 :
540 REM  ALL  DONE  -  BUT  NOW  WE  HAVE  NO
550 REM  REVERSE  CHARACTERS,  AND  SOME
560 REM  MEMORY  SPACE  IS  WASTED.
570 :
580  REM  OF  COURSE,  MACHINE  CODE  WOULD
DO  THIS  FASTER...
590 :
600 REM  SET  UP  A  NEW  INTERRUPT  ENTRY
610 REM  AND  ENABLE  IT
620 : FOR  I  = 32768 TO 32809
630 : READ  A
640 : POKE  I,A
650 :  NEXT
660 :
670 : SYS 32797
680 :
690 : END
695 :
696  REM FIRST—TWO DATA LINES ARE FLASHKILLER
697 :
700  DATA  32,234,255,165,204,208,19,165,211,177,209,41,
     127,133,206,9
710 DATA128,145,209,169, 1,133,207, 76, 79,234, 76, 97,234
715 :
716 REM  NEXT  LINE  IS  ENABLER
720 :
730 DATA120,169, 0,141, 20, 3,169,128,141, 21, 3, 88, 96
```

Best of The TORPET BASIC Programming

Blinking Prompt

By T.H. Holmer, Ashland, Mass.

This program is on The Best Programs Disk

This very simple program shows you how to make a message blink on the screen.

NOTE: When you type in the program, be sure to type in the key itself rather than the words in the listing that describe the key.

```
10 PRINT "(clear/home)"
20 PRINTTAB(3)"This is a test"
30 GET A$:IF A$ = "" THEN:PRINT"(home)(sp4)(dn3)PRESS
RETURN"
40 IF A$ = CHR$(13) THEN 70
50  FOR  T = 1TO400:NEXT:PRINT"(sp4)(up1)(rvs)PRESS
RETURN(rvs off)"
60 FOR T = 1TO200:NEXT:GOTO 30
70 PRINT"(dn1)It works"
```

PUN-ishment

Don't be PET-rified, but be sure to be careful in your computer repairs. Monkeying with the parallel user port could render you PARALLELIZ-ED. It HERTZ.

Ylimaki

The Line-Number Speed Fallacy

By David Williams, Toronto, Ont.

Almost anyone who has become reasonably fluent in programming in Commodore BASIC has also learned a few tricks which are supposed to increase the speed with which programs will run. We all know, for example, that integer variables are (surprisingly) slower to process than real-value ones; that extensive string manipulatins are liable to lead to a process called "garbage collection", which is (except in BASIC 4.0) very time-consuming; and that subroutines are usually accessed faster if they are placed near the beginning of a program than if they are at its end.

We also know — or think we know — that programs will run faster if they are renumbered so that their line numbers are as low as possible. This is because instructions such as GOTO and GOSUB are stored in program memory with the destination line number in the form of a string of ASCII digits. This has to be converted into the computer's internal binary notation before the machine can prodeed with the instruction. A five-digit line number, for example, takes longer to convert than a two-digit one, so programs with low line numbers should run faster than those with high numbers.

BUT IT'S NOT TRUE!

This piece of programming lore has one major defect. It isn't true! If you have access to a line-renumbering utility, try the following little experiment. Take any BASIC program which takes a noticeable amount of time to process information, as opposed to waiting for user INPUT, etc.. Add a couple of lines to it so that it will print out the time it takes to do this processing.

The timing variable, TI, can, of course, be used for this. Renumber the program, starting at line zero and incrementing by one. This gives it the lowest possible set of line numbers. Run the program, making a note of any inputs you may have to give it, and observe the time it takes. Now renumber it again, starting at line zero and incrementing by 300 (yes, three hundred). If, by any chance, this leads to the end of the program having illegally high line numbers (64000 or more), use the highest increment which will keep the numbers within bounds. Run the program again, giving it exactly the same inputs as you did before. Almost certainly you will find, as I have done with this experiment, that the speed of the program is increased by a few percent by giving it the higher line numbers!

It is true that high line numbers take longer to translate into binary notation than do low ones, and it is possible to write programs which will demonstrate this by running faster with low line numbers than with high ones. But there is another effect, which I will describe below, which acts in the opposite direction. In most "normal" programs, which have not been deliberately set up to demonstrate one effect or the other, speed is optimized by using a wide "spread" of line numbers rather than by minimizing them.

Translating the number of the target line into binary notation is only the first step by which a GOTO (or similar) instruction is executed. The computer then has to search through the program to find the line. Because of the way programs are stored in memory, it is not practical (it would be possible but very slow) to search "backwards" to find a line which is earlier in the program than the one which is currently being executed. All searches are done "forwards", starting either at the current line, or from the start of the program.

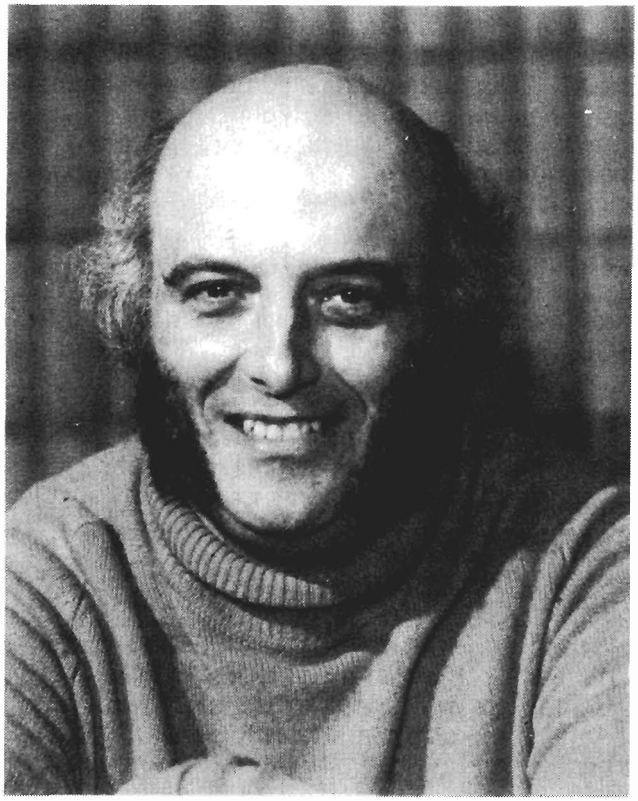
WHY?

There is a curious flaw in Commodore BASIC which affects the decision as to whether to search from the current line or from the first line of the program. Logically, one might expect that the numbers of the target and current lines would be compared. If the target line is later in the program than the current one, the search would proceed from the current line. If the target line is earlier than the current one, the search would begin at the start of the program. In essence, this is indeed what is done. However, the two line numbers are not compared exactly! When they are expressed in binary notation, each of them occupies two bytes of memory. One,known as the high-byte, contains the integer quotient which would be obtained by dividing the line number by 256 (2 to the power 8). The other, low-byte, contains the remainder from this division. To compare the two line numbers exactly, both the bytes would have to be compared, which would be easy enough to do. However in fact only the high-bytes are compared. If the high-byte of the line number of the target line is greater than the high-byte of the number of the current line, the search for the target line starts from the current one. In ALL other cases, BASIC goes back to the beginning of the program to search for the new line.

This little approximation in the BASIC interpreter

cannot be called a real "bug", since it never causes programs to misbehave; however, it can have significant effects on their execution times. Consider, for example, what happens if a program which has 256 lines or fewer is renumbered to start at line zero and increment by one. In binary notation, every line number has a high-byte of zero. Every time a GOTO or similar instruction is executed, the comparison of the high-bytes fails to find the target-line high-byte greater than that of the current line. Thus, every search for a new line has to start at the beginning of the program, even in cases when the target line is later than the current line in the program. This often leads to many lines being searched through unnecessarily, wasting time.

By way of contrast, consider a program which is numbered in increments of 256 or more (this is why I suggested 300 earlier). In this case, each line has a unique high-byte, different from all others. Comparing the high-bytes of two line numbers in this program can determine with certainty which line is later than the other. Thus, "forward" GOTO's, GOSUB's, IF ... THEN (line number) instructions, etc., always work in the most economical way, without searching through the earlier part of the program. This can save a significant amount of time in the execution of a program.



DAVID WILLIAMS

The Lazy Programmer

By Jim Butterfield, Toronto, Ont.

Laziness may be a fault sometimes, but, in programming, laziness can be about the most constructive force you have going for you.

If you ask a beginner to write a program to print the letter X five times, he will likely code PRINT "X":PRINT "X":PRINT "X":PRINT "X":PRINT "X". This will certainly do the job, of course, but around the time that I'm coding the third PRINT statement, I will tend to think: "There must be a better way".

It's not a great quantum leap to decide that the statement PRINT "X" is repeated five times, so we may put it into a loop, and write: FOR J = 1 TO 5:PRINT "X":NEXT J. Very nice and a step towards sophisticated programming.

Here's my theory: The FOR/NEXT program is better, but not because it's shorter (it is, slightly) and not because it's faster (it's not). It's better written because the programmer has made the jump from dealing with the problem to dealing with the nature of the problem.

What does that mean? Well, a couple of examples will illustrate the point. If I had written the above program either way, what would happen if I wanted to change it so that the program printed Y six times? With the first program, we'd have to rewrite almost completely; with the second, a couple of quick changes would do the job. In other words, the first program, written the hard way, does only one thing, but the lazy program is more flexible and solves the general problem.

Let's carry on with our lazy programming activities. If we're asked to print the numbers from 100 to 150, we could code: PRINT 100:PRINT 101:PRINT 102....and so on. Once again, the lazy

instinct says, "There must be a better way." This time, the PRINT lines are not identical. No problem; we just code FOR J = 100 TO 150:PRINT J:NEXT J. Our laziness has led us to a new technique. When we get tired of typing in similar lines, we may be able to use a variable to insert the changeable part. It's not just less work; the program result is really better.

Continuing along the scale of escalating laziness, our next job is to print a series of names: Bob, Carol, Ted and so on. As we grind out PRINT "BOB":PRINT "CAROL":PRINT "TED"....and so on, the old instincts come into play again. A programmer might get writer's cramp putting in all those names; there's gotta be an easier way. This time, we can't compute values like John, Mary and so forth, since they don't arrive in any special pattern, but we can put them in Data statements and Read them as we need them.

So we code DATA BOB,CAROL,TED ... MIKE, followed by FOR J = 1 TO 20:READ N$:PRINT N$:NEXT J. Once again, laziness wins the day! The program will adapt much more easily to a change in the size of the bowling club, or to Carol dropping out and being replaced by Phoebe.

Of course, if we wanted to use the names twice within our program, it would be anti-lazy to have to type them into the data statements twice, and if we wanted to enter the bowling scores we'd have to look for a lazier method. After all, we wouldn't think of writing an INPUT statement for each player — that would be work. So we graduate to arrays. Each name is first copied from a data statement into an array where we can use it over and over. We code DIM N$(20), S(20) to make room for twenty names and twenty scores, and then use the previous data statement with FOR J = 1 TO 20:READ N$(J):NEXT J. Now we can prompt each name and set each bowling score with FOR J = 1 TO 20: PRINT N$(J); :INPUT S(J):NEXT J. Following this, our program can work out the average and print each player's result. It's a lot easier than doing things the hard way. It's also better.

If you write a few lines of code that do something handy — calculate the interest on an amount, say — you could repeat the coding later when you needed to do it again. But we lazy people put the things into a sub-routine and save ourselves the work.

What happened to the good old work ethic? Well, if you want to program the hard way, slugging through each thing to be done one at a time, be my guest. If your programs fit into the machine at all, they will run faster. But they won't be better. It's laziness that causes us to search out the system behind what we're doing, and thus build sounder programs that are easier to change.

For my part, I program the lazy way.

Reading Between The Lines

By David Williams, Toronto, Ont.

This program is on The Best Programs Disk

The BASIC words DATA and READ provide a powerful tool for programmers of Commodore computers. Information which the program will need as it runs can be conveniently written into the program itself, instead of having to be stored in separate files. For example, suppose the program is going to need the names of the months of the year in the course of its operation. The simplest way to store these names is in the form of a string array. If there were no DATA or READ statements in BASIC, two methods would be available to the programmer to set up this array. One would be to specify each array element separately, something like this:

```
10  DIM M$(12)
20  M$(2) = "JANUARY"
30  M$(2) = "FEBRUARY"
40  M$(3) = "MARCH"
    and so on.
```

The other method would be to make a separate file on disk or tape and to access it during the program run with coding such as:

```
10  DIM M$(12)
20  OPEN 1,8,5,"MONTHS"
30  FOR N = 1 TO 12
40  INPUT#1,M$(N)
50  NEXT
60  CLOSE 1
```

This method uses six lines of code, which is obviously more economical than the thirteen lines which the first method needs. However, the need to have a separate disk file is a definite drawback.

The DATA and READ words in BASIC allow the task to be carried out like this:

```
10 DIM M$(12)
20 FOR N = 1 TO 12
30 READ M$(N)
40 NEXT
50 DATA JANUARY, FEBRUARY, MARCH, APRIL
60 DATA MAY,JUNE,JULY,AUGUST
70 DATA SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER
```

That's all there is to it. No separate disk file is needed, yet the similarity to the coding for the separate-file method is clear.

The READ command causes the computer to take the next item of DATA and to place it into a variable in much the same way as an INPUT or IN PUT# command. The first READ statement in a program reads the first DATA statement in the program, and this is true even if they are widely separated. For example, some programmers like to put all their DATA at the end of the program, but the READ commands are usually near the beginning. It is also perfectly acceptable for the DATA to be earlier in the program than the READ commands.

After the first READ command, the next one takes the next item of DATA, and so on. Thus the DATA is read strictly in the order in which it appears in the program.

For many purposes this arrangement is perfectly satisfactory. However, there are other situations in which a programmer may want the DATA to be read in some different order. This is likely to be true if the program contains several subroutines which make use of READ and DATA statements. The order in which the subroutines are executed may be different from the order in which they appear in the program. Indeed the execution order may not be fixed. It may depend on what the program's user decides to do with it. In situations like this, a method has to be devised to allow the DATA to be in a different order than that in which it is to be read.

There is one more command which is associated with DATA. This is RESTORE. When this command is executed in Commodore BASIC (the versions which are used by other computer manufacturers sometimes differ from this) the effect is to set the READ command to start from the beginning of the program again. No matter how many READs have already been carried out, if the program says RESTORE, the next READ will take the first item of DATA in the program.

The RESTORE command can be used with a little bit of cunning to make the computer start READing DATA from anywhere in the program. For example, look at the following piece of coding:

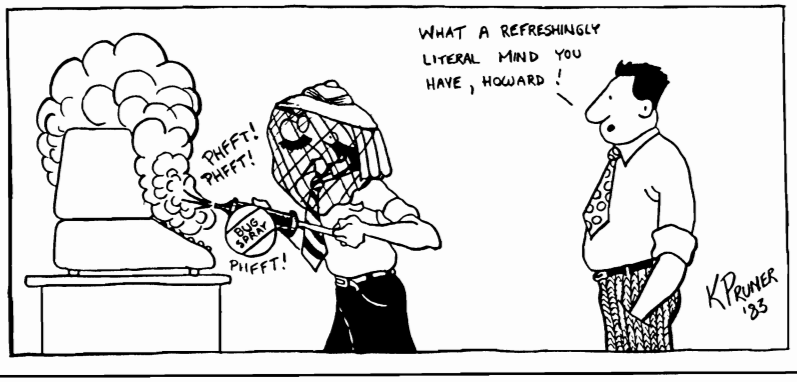
```
1000 DIM N$(3)
1010 RESTORE
1020 READ X$: IF X$◄►"NAMES" THEN 1020
1030 FOR N=1 TO 3
1040 READ N$(N)
1050 NEXT
1060 DATA NAMES,JOHN,SUE,MARY
```

Lines 1010 and 1020 have the effect of finding an item of DATA consisting of the word "NAMES", and they will find this item no matter where in the program it occurs and however many READ statements have already been executed. Line 1010 re-starts the READing process from the beginning of the program, then line 1020 goes through the items of DATA until it finds one consisting of the word "NAMES". It doesn't matter if there are a lot of items of DATA in the program before line 1000, or if a previous READ statement was looking at DATA in line 5000. Providing there is only one "NAMES" in the program, the above piece of coding will put "JOHN", "SUE" and "MARY" into the string array.

If a programmer uses this technique, he can have his DATA in any order which is convenient for himself, and he can arrange for it to be read in some totally different order. Modular programs in which the different routines each contain their own DATA and READ statements are thus perfectly possible. All that is necessary is for each module to have a recognizable keyword at the start of its DATA, and for it to execute a couple of lines of code such as lines 1010 and 1020 in the routine above.

HOWARD

by K. Pruner



Input Idiosyncrasies

By Jim Butterfield, Toronto, Ont.

STRING THING is on The Best Programs Disk

There are some kinds of information we can't seem to get with the INPUT statement. INPUT is a very clever command ... sometimes too clever for its own good. We seem to be forced to use GET to overcome all the things that INPUT does for us ... that we don't want.

Let's take an example. You have a program which asks,

```
INPUT "YOUR NAME";N$
```

and the user types in a reply such as STEVE PUNTER, PhD. the comma "breaks" the input and the user is told, ?EXTRA IGNORED.

We have a somewhat more severe problem if we use the colon character in our input. Not only is the EXTRA once again IGNORED, but we can't even get the second part of the input if we try for it. Coding:

```
INPUT  "DATA";D$,E$
```

and responding with an input of ATTENTION: JIM, JACK will put ATTENTION into variable D$; but JIM and JACK will be lost (we'll get another prompt for string E$). Annoying. This is information that we might want to input and process.

Another problem in addition to the forbidden comma and colon: we are not allowed to input nothing. That sounds like bad grammar; let me restate it. We can't input "nothing" by simply striking a carriage return. PET/CBM machines will stop. VIC and 64 computers will leave the input string with its previous value. And yet "nothing" might be the correct response to various INPUT prompts (middle initial? apartment number? name of spouse? ... you might have no middle name, live in a house, and be unmarried).

There is an answer to all these clumsy things. It's simple, but it's a bit clumsy itself. Tell the user to put his or her reply in quotation marks. In other words, don't type STEVE PUNTER, PHD; instead type "STEVE PUNTER,PHD", including the quotation marks. Commas and colons will be allowed, and you may even type in "nothing" without stopping the computer.

The quotation marks will be removed by the INPUT statement, which leads to the lesser problem: you can't easily input quotation marks. But most of everything else will straighten out.

It seems a little stuffy to require the user to always put in the quotation marks. Mistakes and oversights may occur. The best answer to this problem is buffer-stuffing. Just before giving the INPUT command, place a quotation mark into the keyboard input buffer, and a count of 1 into the input buffer counter. On a recent PET/CBM, you'd do this with POKE 623,34: POKE 158,1; on VIC or C-64, you'd type POKE 631,34: POKE 198,1. This will cause the leading quotes to appear on the screen and be part of the input. The user doesn't really need to type in the closing quotation mark; the system will accept correct input without it.

This takes care of much of the problems of INPUT. A series of GET statements could accomplish the same thing and would be more bullet proof; but there would be more coding needed, and we might risk the danger of invoking a dreaded garbage collection.

However we do it, we are probably setting ourselves up for the next problem. Once we get the input data safely from the keyboard, it's likely that we will put it on a file. Later, when we read the file, we'll want to use the INPUT# statement. And the problem starts all over again.

One way to fix this input problem is to PRINT a quotation mark at the beginning of each record placed on disk or tape. So instead of saying

```
PRINT#6,N$
```

we would code

```
PRINT#6,CHR$(34);N$
```

and each line would start with the quotation mark.

I prefer to use STRING THING to get this kind of input. That's a small machine language routine that does the job without the need for the extra quotation mark. It's been published in The Transactor, and is in The TPUG library.

The important thing is to know to watch for these INPUT problems. Once you know how to spot them, you'll be able to think up a solution.

One more thing to watch when you are doing an INPUT# from a file — you can't get more than 80 characters or so at a time, and so when you write the information, be sure it is broken up into sufficiently small chunks.

INPUT and INPUT# are nice commands. They are fast and convenient. But watch for these problems of curious characters (comma and colon) and "null" inputs.

A Tiny PET/C64 Print Hint

By Elizabeth Deal, Malvern, PA

This program is on The Best Programs Disk

Sometimes a screen design requires using all forty positions of a line. You can define a 40-character string and print it with a semicolon at the end, falling through to the next line. Don't do it!

There is a pitfall in this method, causing difficulties in debugging. You cannot reliably overprint such a screen. The computer keeps track of the double lines and in so doing messes up your best intentions. You issue one cursor down, for instance, and try to print your name ... and it invariably (or one out of two times) ends up precisely where you didn't want it.

The reason is that the computer is doing its best to keep its house in order. It follows the pattern of single or double lines established by the first PRINT, and the subsequent overprinting follows the line length already known. The table of line lengths is kept in page zero. See memory map under "screen line link table" or a similar name, if you're interested in details.

There is a way to print forty character strings by doing a little trick with the insert-character. To understand how this works, try this: in direct mode, print 39 stars. Now move one cursor left and push the INST-key (that's shifted-delete). Type a star in the gap remaining, and you have a clean, 40-character line.

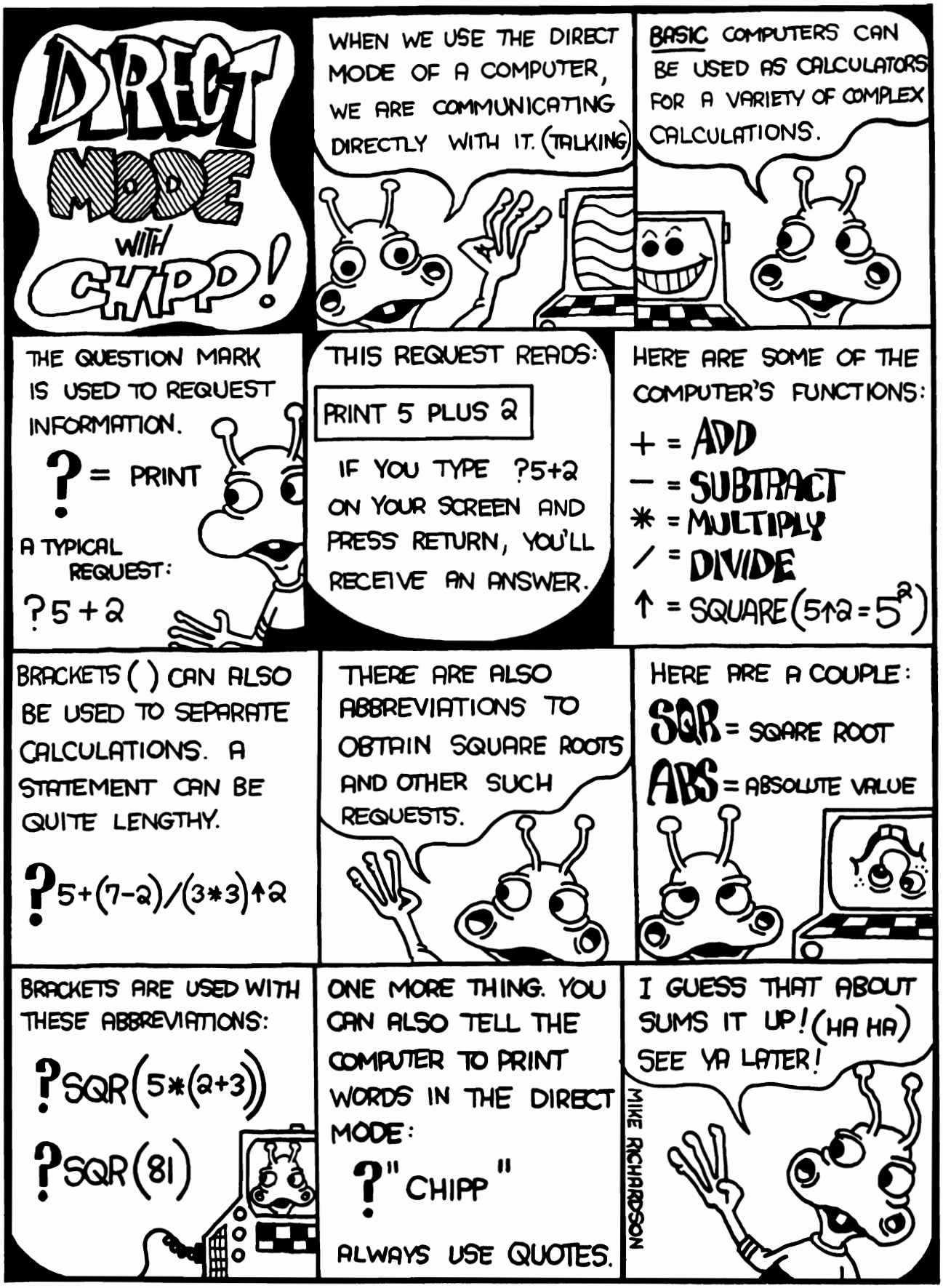
As an illustration of using this sort of thing for more than one line on the screen, use the framemaker routine. The same process is used as the previous, direct mode, procedure. Note that each line ultimately ends in carriage return (no semicolon). This keeps us out of trouble. Following the last line of the program you can insert some code of your own to print the screen in any position you like, or you can use what's there already. If you plan to print lines of 40 characters, again, use the insert procedure. For this demo I have not done so.

This method is particularly handy on the C64. On the PET, we often print 39 characters, and poke the fortieth one onto the edge. Often however, there is no need to bother.

On the C64 this is more critical especially when the edge and background colors are different. The screen may look sloppy if the fortieth position is empty. Furthermore, the C64 opens up lines on the screen when the cursor is in the last position, useful in program editing, a headache in neat screen design! Since poking both the screen and the color memory isn't fun, the framemaker method can be a viable alternative if you need it.

```
10 REM FRAMEMAKER....ELIZABETH DEAL
20 F$=CHR$(157)+CHR$(148)
30 TB$="****************************************":REM 39* IN ""
35 TB$=TB$+CHR$(145)+CHR$(13)+CHR$(148)+"*"
40 MD$="*                                      *"+F$:REM *,37 SPACES,* IN ""
50 PRINTTB$:REM TOP LINE
60 FOR J=1 TO 23:PRINTMD$:NEXT J:REM MIDDLE
70 PRINTTB$;"■";:REM BOTTOM
80 REM HOME IS USEFUL FROM BOTTOM LINE
90 PRINT"■■■■■LINES":REM 5 DOWN IN ""
```

I am beginning to think some kind of military sabotage is going on inside my 64. If I don't supply it with exactly the right code, programs have been bombed in mid-run. I first blamed it on the C.I.A. chips; however, I have since learned the operations are controlled by a tough military man, the KERNAL. — Ylimaki


DIRECT MODE WITH CHIPP!
WHEN WE USE THE DIRECT MODE OF A COMPUTER, WE ARE COMMUNICATING DIRECTLY WITH IT. (TALKING)
BASIC COMPUTERS CAN BE USED AS CALCULATORS FOR A VARIETY OF COMPLEX CALCULATIONS.
THE QUESTION MARK IS USED TO REQUEST INFORMATION.
? = PRINT
A TYPICAL REQUEST:
?5+2
THIS REQUEST READS:
PRINT 5 PLUS 2
IF YOU TYPE ?5+2 ON YOUR SCREEN AND PRESS RETURN, YOU'LL RECEIVE AN ANSWER.
HERE ARE SOME OF THE COMPUTER'S FUNCTIONS:
+ = ADD
— = SUBTRACT
* = MULTIPLY
/ = DIVIDE
↑ = SQUARE (5↑2 = 5²)
BRACKETS ( ) CAN ALSO BE USED TO SEPARATE CALCULATIONS. A STATEMENT CAN BE QUITE LENGTHY.
?5+(7-2)/(3*3)↑2
THERE ARE ALSO ABBREVIATIONS TO OBTAIN SQUARE ROOTS AND OTHER SUCH REQUESTS.
HERE ARE A COUPLE:
SQR = SQARE ROOT
ABS = ABSOLUTE VALUE
BRACKETS ARE USED WITH THESE ABBREVIATIONS:
?SQR(5*(2+3))
?SQR(81)
ONE MORE THING. YOU CAN ALSO TELL THE COMPUTER TO PRINT WORDS IN THE DIRECT MODE:
?"CHIPP"
ALWAYS USE QUOTES.
I GUESS THAT ABOUT SUMS IT UP! (HA HA) SEE YA LATER!
MIKE RICHARDSON

Machine Language Programming

PAGE

Machine Language From Square One 86
Larry Goldstein, Bolton, Ont.
Here is your first lesson if you are wondering about machine language programming.

If It Worked Once It'll Work Again 89
Larry Goldstein, Bolton, Ont.
Here is some help for the beginning machine language programmer.

If Then Branching — Machine Language 92
Vince Sorenson, Regina, Sask.
If you are just beginning to learn machine language these will be some of the first techniques you need to learn.

How to Include ML Routines in your VIC BASIC Programs 93
Terry Herckenrath, Toronto, Ont.
Sometimes all you need is a little bit of machine language to make a program work better.

Generating Random Numbers in Machine Language 95
Vince Sorenson, Regina, Sask.
When you get a little further along in machine language you sometimes need random numbers for games and things. This is sometimes a little tricky to do in machine language but this article tells you how to do it.

Differential Relocation of Machine Code 97
Harold Anderson, Oakville, Ont.
Here is how to solve a difficult problem for the more advanced programmer.

Putting It All Together: The Assembler 98
Larry Goldstein, Bolton, Ont.
Before you go very far in machine language programming you realize you need an assembler. This article explains what an assembler is and how to use one.

6502/6510 ML Instructions 100
Vince Sorenson, Regina, Sask.
The 64 and VIC have quite a number of extra instructions that many people have not recognized. Here is a list and explanation of those hidden instructions.

CBM Conditional Assembly 101
Mark Niggemann, Ames, Iowa
How to write two different programs by using the same source files.

Machine Language From Square One

By Larry Goldstein, Bolton, Ont.

SUPERMON/64 and VIC MICROMON are on The Best Programs Disk

So you've finally gotten over the shock of actually owning a COMPUTER, and you're starting to get pretty good at BASIC programming. The next challenge has to be machine language, but where to start? In fact machine language programming is not terribly difficult but the first steps are big ones. After that you'll be able to design your own routines, understand the listings in magazines, and maybe even be able to comprehend the beginner's machine language columns in COMPUTE! I hope to present a series of columns to help the reader take those first steps. I'll assume that you understand BASIC programming but that you have no computer expertise beyond that.

WHAT IS MACHINE LANGUAGE

Machine language is the method by which all commands and data are stored and transferred within your computer. This consists of patterns of electrical voltage which are stored in microscopically small switching circuits. If one of these switches is turned on, it can deliver a voltage when necessary; and off switch delivers no voltage. These switches comprise the memory of your computer. Some of these switches can be turned on and off continually and they are the RAM which stores your programmes and your data. Other switches have been set permanently in manufacture; they comprise the ROM which holds the routines that run the computer. There are tens of thousands of such switches in any of our favourite computers. The following little programme will show the patterns which exist in your computer. It will work with any Commodore, but the colour machines should be set for good contrast between the background and the printed characters.

```
10 A$ = CHR$(207) + CHR$(146): B$ = CHR$(18) : Z = 128
20 FOR I = 0 TO 65535: X = PEEK(I)
30 FOR J = 1 TO 8: IF(X AND Z) THEN PRINT B$;
40 PRINT A$;: X X*2
50 NEXTJ,I
```

The lighted (or foreground colour) squares represent switches that are turned on, while the dark (background) squares are off switches. The programme scans the entire potential memory of your machine (largely empty in an unexpanded VIC) and runs a couple of hours so you may want to continue reading while it carries on.

It is the pattern of switches, rather than individual switches, which is important to us. In fact it is the pattern presented by a group of eight switches that runs the computer. Eight switches, acting as a unit, make up one byte of memory; each individual switch is a bit. So the switching patterns you see on your screen, taken in groups of eight (or bytes), comprise machine language.

You've never seen machine language listings made up of patterns like this or even of patterns of "on" and "off" or "5 volts" and "0 volts". Such a system is simply too unwieldy. But if we use the digit "1' to mean an on switch and a "0" to mean off, a miserable machine language byte like "on off on off on off off on" (that's no byte--it's a mouthful) becomes 10101001. This is a definite improvement, but still pretty awkward. What we have now is something that looks like a binary number, a number made up of only the digits "0" and "1". In our usual (decimal) system, we have ten digits to work with, and when we want to count beyond nine we start a second column for multiples of ten, and then a third column for multiples of one hundred, and so on, where the value of each column is ten times the value of the one to its right. In the binary system we can count only to one before we need a second column (for twos), and then we need another for fours, and so on with each column having a value of twice the value of the one to its right. If we want to take our pattern of eight and apply it to the binary system, we must figure out the value for the eight columns involved. This will be:

```
128   64   32   16   8   4   2   1
and 10101001 becomes
1     0    1    0    1   0   0   1
or
1x128 + 0x64 + 1x32 + 0x16 + 1x8 + 0x4 + 0x2 + 1x1 = 169
```

Now that's a nice neat comprehensible number. To us. The computer would much prefer"on off on off on off off on", but as a special favour to an inferior, the computer allows us to feed it information as decimal numbers which it then converts to switching patterns and stores and uses. If we want to understand the workings of the computer, it is sometimes useful to work in binary notaton which, as you can see, is midway between our familiar decimal notation and the patterns that are important to the machine.

WHY BOTHER?

This is beginning to look like work. Why would

anyone want to get involved in machine language anyway? There is a variety of reasons:

1. So you can come away from meetings without feeling inferior to the kids.

2. Things happen a lot faster in machine language than in BASIC, especially repetitive routines.

3. Machine language programmes generally make more efficient use of memory space than BASIC routines.

It's up to you to decide whether the first reason is worthwhile; let's look at the second and third.

If your machine is still spewing out coloured squares, you may as well STOP it and enter this little routine: (In line 10, substitute for the stars the appropriate values for your machine from the following table.)

PET/CBM	A	B
40 col.	32768	33767
80 col.	32768	34767
VIC 20	7680	8186 (unexpanded)
C-64	1024	2023

Note that line 5 is used only for VIC.

```
  5 FOR I = 38400 TO 38905: POKEI,0: NExT: REM VIC ONLY
10 A = *****: B = *****:
20 FOR I = 0 TO 255
30 FOR J = A TO   B: POKE J,I
40 NEXTJ,I
```

This routine is just as mind-numbing as the last one, but it doesn't run nearly as long. You might like to time it. When you're done , get rid of this with a NEW.

Now for comparison, try this one. Again, where you see a letter in the data statements, substitute the appropriate value from the table:

PET/CBM	A	B
40 col.	127	131
80 col.	127	135
VIC 20	29	31
C-64	3	7

and add line 5 as in previous program.

```
10 FOR I = 830 TO 862: READ X: POKE I,X: NEXT
20 DATA 160, 0, 169, 0, 133, 3, 170, 169
30 DATA A, 133, 5, 169, B, 133, 4, 138
40 DATA 145, 3, 136, 208, 251, 198, 4, 165
50 DATA 4, 197, 5, 208, 242, 232, 208, 235, 96
```

Carefully check your typing and make sure that you have the right values in line 30 for A and B. Now run it. How's that? The best yet, right? What has happened is that you have changed the patterns in memory locations 830 to 862. These patterns, if acted upon, will cause something to show up. To tell the machine to go to memory location 830 and follow the machine language instructions from there on, you type SYS 830 and press RETURN. That's better. Still a mindless routine, but fast.

To understand the difference in speed, we must realize that a BASIC programme is read by the computer one command at a time, this command is translated into machine language and is then executed. Then the next command, and the next, and so on. This all happens very quickly by most standards, but the command "POKE J,I" in line 30 of the BASIC programme is read, translated, and executed over a quarter of a million times (for a 40 column machine), and you saw how the time added up. The machine language routine does essentially the same job as the BASIC, but the reading and translating are not necessary, and the time saving is obvious. As far as memory is concerned the machine language takes up only thirty-three bytes, whereas the BASIC needs over 45. That's why machine language programming is worth the bother. By the way, the BASIC programme we used to get the patterns into memory is called a BASIC loader. Once the machine language is safely in place, the loader is no longer needed; you can get rid of the loader with a NEW command, and still use the SYS 830 until your brain rots. The BASIC loader is one of a few ways of dealing with machine language. Each method has certain advantages and disadvatages over the others.

WHERE DID THOSE NUMBERS COME FROM?

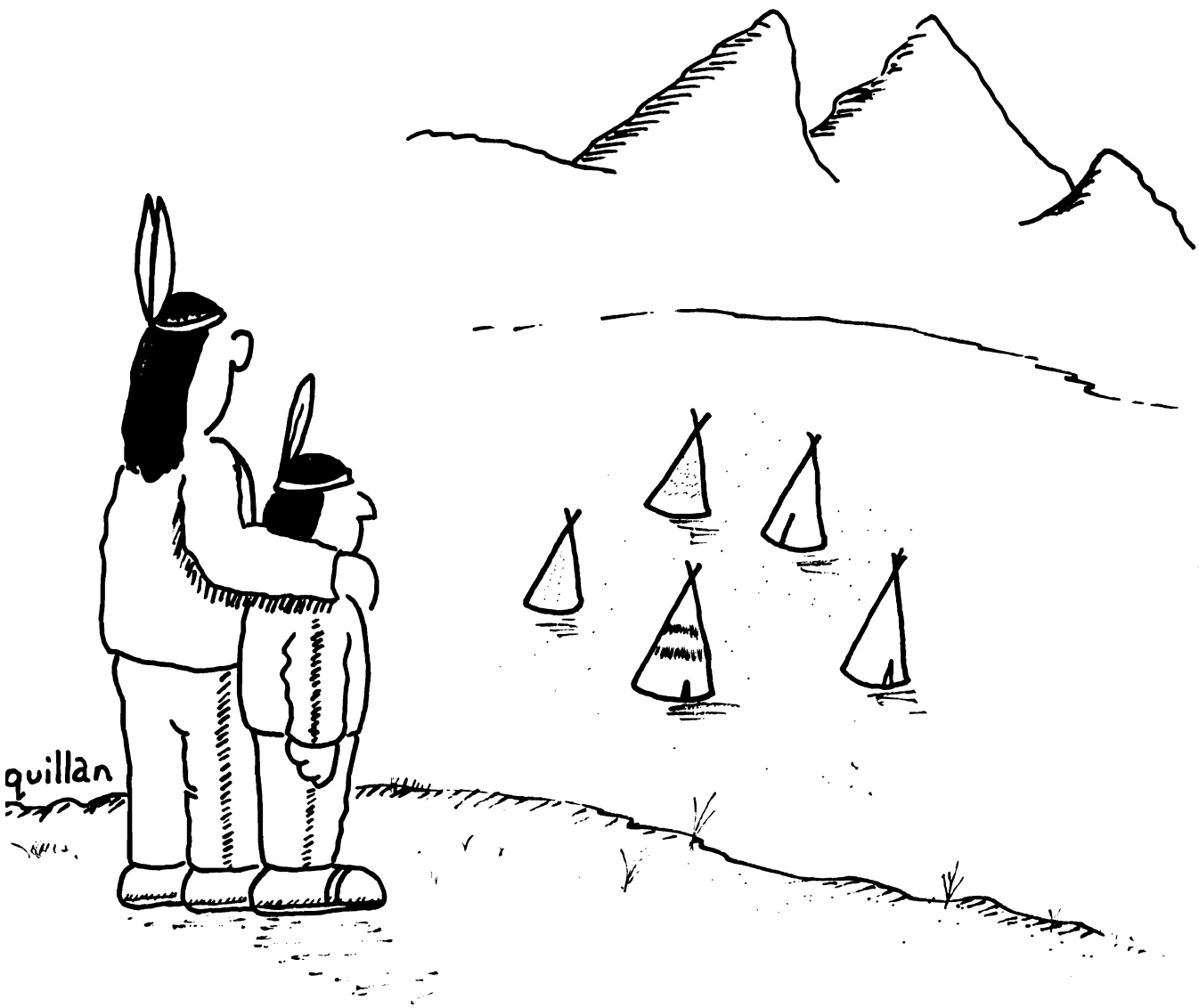
I was afraid I'd ask that question. The machine language programmer has a few dozen instructions to use on the machine. These are different from BASIC commands, although some of them do similar jobs. After writing a programme using these instructions, you have to look up the numbers (patterns) for each one and fill in your data statements. This wouldn't be too bad for our little letter flasher, but you can imagine that a programme of any size would be almost impossibly tedious to write by this method. Since computers

are supposed to make life easier, it seems logical to let the machine do some of the job itself and look up the numbers for us. (This doesn't happen every time the programme is run, as for BASIC, but just once when the programme is first written.) Such programmes, called ASSEMBLERS, are readily available for various prices, each one working a bit differently and offering different aids and shortcuts. By far the best buys are the Supermons and Tinymons and Vicmons, etc. They don't do a lot of the fancy things that the expensive ones can, but they are great for starting out, and you may never need anything fancier. If you don't have one of these in your library of club programmes already, please buy one. We'll be continuing on it in future installments.

As a final frivolous fling, let's redo the first memory scan programme in machine language. Here's the BASIC loader (the same for all machines):

```
10 FOR I = 830 TO 873: READ X:POKE I,X: NEXT
100 DATA 169, 0, 133, 3, 133, 4, 168, 177
110 DATA 3, 72, 162, 8, 104, 10, 72, 144
120 DATA 5, 169, 18, 32, 210, 255, 169, 207
130 DATA 32, 210, 255, 169, 146, 32, 210, 255
140 DATA 202, 208, 233, 104, 200, 208, 224, 230
150 DATA 4, 208, 220, 96
```

After checking your typing, RUN this loader programme, and enter SYS 830. It will run in the time it takes to drink a leisurely cup of coffee. The repeating vertical lines indicate empty memory space, so C-64 owners especially will get plenty of that.



ONE DAY, SON, ALL THIS WILL BE SILICONE VALLEY.

If It Worked Once It'll Work Again

SUPERMON/64 and VIC MICROMON are on The Best Programs Disk

By Larry Goldstein, Bolton, Ont.

A program which needs a separate command for every operation is tedious to write and slow to run. Filling a screen, for example, would require between 500 (for a VIC) and 2000 (for an 8032) LDA and STA commands to put the information where you want it. In BASIC we would use a FOR...NEXT loop to accomplish this; in Machine or Assembly Language we use branching loops.

Consider the following BASIC program (substitute the appropriate values for your machine for the stars in line 10):

```
5 FOR V =    38400 TO 38905:POKE V,0:NEXT:REM FOR VIC
ONLY
10 SM =    ****: REM USE 32767 FOR PET/CBM, 7679 FOR
VIC, 1023 FOR C-64
20 FOR I =    1 to 19
30 READ Q
40 POKE SM + I, Q
50 NEXT
60 DATA 19,21,2,19,3,18,9,2,5,32
70 DATA 20,15,32,20,15,18,16,5,20
```

Obviously, this routine will read the data one at a time and place them in screen memory starting at the upper left hand corner of the screen and ending 18 spaces to the right. In Assembler we would LOAD the data into the Accumulator and then STORE them in screen memory, using the X- or Y-Register as a counter or index.

To duplicate this programme in Assembly Language, load Supermon into your machine, and RUN it, then enter the following:

```
.A 034B LDX #$13
.A 034D LDA $0356,X
.A 0350 STA $7FFF,X
.A 0353 DEX
.A 0354 BNE $034D
.A 0356 RTS
```

*** For VIC substitute $1DFF in the third line. For C-64 use $03FF.
("A"is the instruction to perform an assembly.)

The first line is pretty well self-explanatory. We simply load the number $13 (hex) or 19 (decimal) into the X-Register to act as our counter.

Next we load the accumulator with the contents of memory location $0356+X ($0356 + $13 = $0369).

Then store this value in memory location $7FFF + X or $8012 (for PET/CBM). This location is part of screen memory, so the character appears on the screen.

Now decrease (or decrement) the contents of the X-Register by 1, resulting in a value of $12. Whenever a numerical transaction like this occurs, there may be a change in the Status Register. If the result of the transaction is zero, this is noted, as is a negative result. The result, $12, would be noted as being non-zero and non-negative. The next operation uses this information.

BNE means "Branch if the result is *N*ot *E*qual to zero". In Assembler, this command is followed by the memory location that you want the programme to go to. This instruction resets the Programme Counter to this address, and the branch is accomplished. However, the branch can only be a maximum of 128 steps forward or backward.

The programme now branches back to the LDA command with X set at $12, so another datum is loaded and stored in another screen location. This procedure operates in reverse compared to the BASIC version, starting at the end of the message and working its way back. Eventually the X-Register contents will become zero and the branch will not work (the last datum used will have been loaded from $0356 + 1) so the programme will proceed to... RTS which returns the computer to BASIC (in this case).

So far we have no data to work with. Let's get out of the assembly mode by pressing RETURN. Since the programme has taken us to location $0356, we can store data starting at $0357 (which is why the address $0356 was chosen in the second line of the routine). Enter the instruction, M 0357 0367, and you will see a display of a number of memory locations starting at $0357. Simply change the two digit numbers to duplicate the ones below by typing the changes right over the display and pressing RETURN at the end of each line. (The VIC display will look a little different because of the shorter line lengths, but the principle is the same.)

```
0357 13 15 02 13 03 12 09 02
035F 05 20 14 0F 20 14 0F 12
0367 10 05 14 00 00 00 00 00
```

These are the same numbers as in the BASIC DATA statements but in hexadecimal notation.

The zeroes at the end mean nothing. Exit Supermon by entering X RETURN, clear the screen, and RUN the routine with SYS 843. (VIC NOTE: When you clear the VIC screen you also clear colour memory and your display is invisible. Use line 5 from the BASIC programme to fill colour memory, then SYS 843.)

If you now reactivate Supermon (SYS 4 for PET/CBM, SYS 8 for VIC and C-64), you can take a fuller look at the programme. Once Supermon is running again, enter D 034B RETURN and you will get a screen full of stuff. The columns of numbers on the left side of the screen are the memory locations of the commands followed by the machine language number (pattern) for the command and the two or three number address (if any). Note that the two byte address $0356 is stored in reverse or LOBYTE/HIBYTE order. Finally, there is the Assembly Language version, which is not actually stored in memory, but has just now been translated (or disassembled) by Supermon for our convenience.

LET'S TAKE ANOTHER LOOK

We have begun to accumulate quite a repertoire of commands, so let's take another look at them.

LDA, LDX and LDY are commands to put a value into the Accumulator or the X- or Y-Register. We can specify the actual value to be loaded as we did with LDX #$13. This is called Immediate Mode addressing and works for all three registers. This method of addressing is easy to use and to understand, but it is not very flexible, since the actual value must be provided each time the command is used. We could also specify "load the accumulator with the value found at memory location $1234" using the command, LDA $1234, and similarly LDX $1234 or LDY $1234. In giving the specific address from which the value is to be fetched, we are using Absolute addressing. Finally, we specified "load the accumulator with the value to be found in memory X steps away from location $0356". This method is very handy when wanting to use a series of values, as we did here, or one of a series of values depending on the value of the index. This is Absolute, Indexed addressing, and we could use either the X- or Y-Register value as the index.

STA, STX and STY operations are also available with Absolute and Absolute, Indexed addressing modes.

The decrement commands, DEX and DEY decrease the value in the respective registers by 1. If the register holds a zero, then DEX or DEY yields a value of 255 ($FF). It's just like rolling back a hexadecimal odometer. The opposite commands, INX and INY, will increment the registers or increase them by 1.

Since the X- and Y-Registers can hold values only up to 255 ($FF), the indexed addressing modes can only reach 255 locations ahead of the starting address. In other words LDA $0356,X can only LOAD from $0356 + FF at the maximum and STA $7FFF,X can only store in location $7FFF + 255 maximum.

The branch command BNE (and its opposite BEQ or "Branch if the result is EQual to zero") responds to the Status Register and whether or not it holds information of a zero-result. BNE executes the branch if there was not a zero result, while BEQ branches if the result was zero.

In running the loop "backwards" we simplified things by specifying the extent of the loop by setting the index to $13 at the beginning and using the ability of the machine to distinguish between a zero and a non-zero. We could have done it the other way around by setting X to 1 and then using INX each time, checking to see if it had reached $14 yet, and then doing the branch if X was less than $14. This requires an extra step, checking for $14, which uses up memory and wastes time when running.

If we want to affect the whole screen using routines similar to this one, we run into the limitation mentioned previously, that indexed addressing can only affect up to 256 addresses (for X = 0 to 255). The simplest way around this is to put in a number of commands which will affect the whole screen. For a 40-column PET, screen memory starts at $8000 and includes 1000 locations. So we can fill the screen with the letter A using a routine like:

```
A 033E LDA #$01
A 0340 LDX #$00
A 0342 STA $8000,X
A 0345 STA $8100,X
A 0348 STA $8200,X
A 034B STA $8300,X
A 034E DEX
A 034F BNE $0342
A 0351 RTS
```

The third line will put character #1 (i.e. A) in locations $8000 to $80FF, the next line carries on for the next 256 locations and so on. Notice that the first execution of the routine fills $8000, $8100, $8200, and $8300. The index then becomes $FF, the branch executes and repeats until X becomes 0 again. To run the same on the 8032, we would need three more lines continuing to $8600,X. The VIC needs only two lines to fill its 506 locations

($1E00,X and $1F00,X) while the C64 would need four lines (STA $0400,X STA $0500,X STA $0600,X and STA $0700,X--don't do it yet; read on).

Another piece of memory of interest is colour memory in the VIC and 64. Every screen location is represented by a colour memory location whose contents determine the colour of the character displayed there. Using the routines shown above, but changing the addresses to colour memory instead of screen memory, allows you to change the entire display instantly. For example:

```
A 033E LDA #$02
A 0340 LDX #$00
A 0342 STA $9600,X
A 0345 STA $9700,X
A 0348 DEX
A 0349 BNE $0342
```

This routine will fill VIC colour memory with the value of two, resulting in a red display. If entered as shown it will fit in just ahead of our previous advertising message, and a SYS 830 will give a visible display without the "line 5" routine. A similar routine can be written for the 64 to change the display colour of all or part of its screen. If used by itself, the routine would end with RTS to return to BASIC when it's finished.

In experimenting with the VIC remember that screen and colour memory shift depending on whether or not you are using a memory expander. The locations are as follows:

	Unexpanded	**Expanded**
Screen Memory	7680-8191 $1E00-$1FFF	4096-4607 $1000-$11FF
Colour Memory	38400-38911 $9600-$97FF	37888-38399 $9400-$95FF

The screen memory of the C64 can be shifted all over the place with starting addresses ranging from 0 to 15360 ($0000 to $3C00), but the basic location is 1024 to 2047 ($0400 to $07FF). When changing screen memory, remember that the last 8 bytes at the end of screen memory (wherever it's located) are used as sprite pointers. This means that a routine including $0700,X , for example, could clobber your sprites. To be safe, use $06E0,X instead. Colour memory does not move, and can always be found at 55296 - 56295 ($D800 - $DBE7).

The operations included in this article are sufficient to design some very useful and attractive routines. Let your imagination run loose with them.

YOU FAILED WHAT?!?

If Then Branching — Machine Language

By Vince Sorensen, Regina, Sask.

After the ML beginner has understood how to say "LET" and "STORE" (LDA and STA), the next thing he'll probably want to learn is how to say "IF...THEN". With these commands, most applications can be accomplished. However, saying "IF...THEN" in ML involves many more commands than just an "IF" statement and a "THEN" statement, and this is where many beginners can be led astray. It has happened to everyone I know just starting out, including myself.

The thing to remember is that there are eight conditions which can be used as part of the ML "IF ...THEN" or branch statement. If there is or isn't a carry left over, if the last number referenced to was or wasn't a zero, if it was or wasn't negative, or if there was or wasn't an overflow, you can check for it. When you load a register or accumulator (your three ML variables are A for accumulator, X for the X register, and Y for the Y register), the result is examined for negatives, or zeros. When you compare, increase, or decrease, the result is again checked, for negatives, zeros, carries. This is what I mean by the last number referenced. Your eight commands for these possibilities are:

BCC - Branch if the carry is clear
BCS - Branch if the carry is set
BEQ - Branch if equal (last result was zero)
BNE - Branch if not equal (not zero)
BMI - Branch if minus (negative)
BPL - Branch if plus (not negtive)
BVC - Branch if overflow clear
BVS - Branch if overflow set

Along with these branch commands, you will usually use comparison commands (when in doubt, check or compare again). To compare, you will use CMP, CPX, and CPY. In my examples, I will use immediate mode, where the register is compared with what immediately follows.

Due to the fact that I believe that you learn more from demonstration, here is an example of a typical branch:

LDA $A2 Load the accumulator with the low byte of VIC's clock.

CMP #$10 Compare it with 10. If it is 10, then the ZERO or equal bit will be set, and the negative bit cleared. If the accumulator is less than 10, the negative bit is set, and the carry register is cleared, as well as the zero. If it is more, then the negative bit is cleared, the zero bit cleared, and the carry bit set.

BEQ EQUAL If the zero bit is set then go the EQUAL routine.

BCC LESS If the carry is clear, then go to the LESS routine.

BCS MORE If the carry is set, then go to the MORE routine.

In place of BCC, BMI could have been used. In place of BCS, BPL could have been used. However, BEQ should be the first operation, since the fact that zero is considered positive could have you going to the MORE routine if you're not careful.

Already, you have the BASIC branch statement under control. After your programs get longer, however, you'll have to watch how far away you are branching to. Since branches use relative addressing (that is to say, they go to a certain spot a certain number of bytes away from themselves), they can only go so far. If you wish to branch further than 128 bytes in either direction, you are unable to. The solution to this is to use absolute addressing, where saying goto $4000 will take you to location $4000, instead of $4000 bytes up. An example of this coding:

```
LDA $A2
CMP #$10
BEQ EQ1   }
BCC LE1   } Branches to correct jumping point
BCS M01   }
EQ1 JMP EQUAL  }
LE1 JMP LESS   } Jumps to correct routine
MO1 JMP MORE   }
```

The command JMP says to go to a location, no matter what. Thus you can use branches as an "IF...THEN" statement, and the JMP command as a "GOTO" statement. At this point, we run into the problem that beginners keep straying into. They try this coding:

```
LDA $A2
CMP #$10
JMP ITS10
```

Sorry, it's less work, but it doesn't work at all. When the JMP statement is executed, it doesn't care if you're comparing or not. The proper way to code this is:

```
LDA $A2
CMP #$10
BNE CONT
JMP ITS10
CONT .....
```

With this kind of coding, you'll notice that the only time the JMP statement is run into is when the accumulator has $10 in it. Otherwise, your program carries on at CONT. What I am trying to emphasize here is that if you give your computer a chance to make a mistake, it will. Always make

sure that you have compared what you wanted to compare, and then use that comparison. Then you are well on your way to becoming a good ML programmer.

FURTHER READING ON ASSEMBLY LANGUAGE PROGRAMMING

6502 Assembly Language Progrmming - by Lance Levanthal (Osborne/McGraw Hill)

VIC & C-64 Programmer's Reference Guide from Commodore (Howard W. Sams & Co., Inc.)

Compute! Magazine (Small Systems Publications)

These should be available at your local Commodore dealer's.

*In ML, negative numbers are those that are from 128 to 255. This is due to the way that numbers are stored, as bits.

How to Include ML Routines in your VIC BASIC Programs

By Terry Herckenrath, Toronto, Ont.

Since I've started playing around with my VIC, I have encountered a number of ways of including M.L. routines in a BASIC program.

The most common approach seems to be to include the M.L. routine in the form of DATA statements. At the beginning of the program, a little FOR-NEXT loop POKES the M.L. routine in either high memory or the cassette buffer.

This approach has some drawbacks:

1. The M.L. routine takes up at least THREE TIMES more memory than is necessary.

For each byte of M.L. code you need up to three bytes to represent its value in BASIC, one byte for the actual M.L. code, plus the additional space needed for the DATA tokens, data delimiters (comma's), line numbers, line links and the FOR-NEXT loop to put it all in place.

2. You cannot re-use other information kept in DATA statements.

In some programs it is desirable to read the DATA statements over and over again.

3. The DATA statements can accidentally be clobbered.

While making changes to the program DATA statements can be inserted in the wrong spot, or could even be deleted altogether.

For some time now, I have added M.L. routines to the end of the Basic text through the use of my M.L. monitor.

You will have to include a statement in your BASIC program that calculates the START ADDRESS or ENTRY POINT ADDRESS of the M.L. routine. You will notice that two addresses will be printed on the screen while the M.L. routine is being appended to the BASIC program. These are the START and ENDING + 1 addresses of the M.L. routine AT THIS MOMENT. Subtract the first from the second to arrive at the LENGTH of the M.L. routine. Then, in your program you calculate the ACTUAL ENTRY POINT ADDRESS as follows:

```
EP% = PEEK(46) * 256 + PEEK(45) - MLlength
```

where MLlength is the length of the M.L. routine.

This is assuming that the START ADDRESS and the ENTRY POINT ADDRESS are the same. If you start executing the M.L. routine somewhere beyond the START address, you must adjust the value of M.L. length accordingly.

From then on you use SYS EP% when you want to execute the M.L. routine.

If you append more than one M.L. routine to the same program, be sure to re-adjust the entry point address of the "older" routine.

One final word of caution:

Do NOT use "VIC TINY AID" (by Jim Butterfield/Dave Hook) to manipulate a program that has an M.L. routine appended to it. This utility WILL clobber the M.L. routine (and so might other "toolbox" type utilities).

Storing the M.L. routine at the end of the BASIC program eliminates the drawbacks cited above. It takes up the minimum amount of space since we

are storing the actual M.L. routine; no DATA statements are needed; and YES you can STILL make changes to the BASIC program once the M.L. routine has been appended. BASIC just keeps moving it up and down in memory with the rest of the BASIC program.

There is ONE limitation that I can think of: the M.L. routine must be relocatable; i.e. there can be no JUMP (JMP and JSR) instructions that refer to some point within the M.L. routine. In most cases, this is not a serious limitation.

For those of you who like to understand how the VIC operates: BASIC keeps track of the end of a BASIC program in TWO ways:

1. It keeps track of the LOGICAL end of the BASIC program through a "line link" of zero.

This tells BASIC when to stop LISTING or RUNNING a program.

2. It keeps track of the PHYSICAL end of the BASIC program through the pointer in locations 45 & 46, which points to the byte FOLLOWING the BASIC program. This pointer is used to LOAD and SAVE the program, and for RELOCATING parts of the BASIC program as changes are made to it.

We use this pointer to append the M.L. routine to the BASIC program; then we change its value to tell BASIC that the PHYSICAL length of the program has increased.

The following three line program will append an M.L. routine to any BASIC program, even if it already has some other M.L. routine appended to it.

```
10000 I% = PEEK(46) * 256 + PEEK(45) + 4 : PRINT I%
10001 READ J% : IF J% ►=0 THEN POKE I%,J% : I% = I%
+ 1 : GOTO 10001
10002 PRINT I% : J% = I%/256 : I% = I% - J% * 256
10003 POKE 45,I% : POKE 46,J% : CLR : END
```

NOTES:

1. The DATA used for the M.L. routine must be followed by a negative value to end the loop.

2. The DATA statements used for the M.L. routine must be the FIRST or ONLY DATA statements in the program.

3. The "+4" in line 10000 represents the space needed for the variables I% and J%. These four bytes will be imbedded in the final product and are "wasted".

4. The CLR command at the end of line 10002 will adjust the two pointers used by BASIC to keep track of numeric variables. If you run your program after appending the M.L. routine without adjusting these two pointers, the M.L. routine will get clobbered as soon as a numeric variable is used.

Once the M.L. routine is in place, you can delete the DATA statements and the above program.

"AT LEAST THE COMPUTER KEEPS HIM OFF OF THE STREETS."

Generating Random Numbers in Machine Language

By Vince Sorensen, Regina, Sask.

One of the most difficult problems to tackle is to find how random, or illogical, numbers can be generated in a totally logical machine. Commodore BASIC solves this by making lists of "random" numbers using logarithms and subtraction. Since this is not a completely random process, the same list is generated each time the computer is turned on. Thus if you PRINT RND(1) after turning on your computer today, it will show the same number it did last time you did this. Try doing this. Write down the number, turn your computer off and on, wait a while, and PRINT RND(1).

Since I have a VIC, I was able to try plugging in extra memory to see if that made any difference. It didn't, so I was able to conclude that the process of generating random numbers is independent from the amount of memory, and the timer 9 (because of the previous test). Random numbers are therefore found using logic. However, each machine has its own list, different from any other's, but the same as its own. Compare your first RND(1) with mine: .185564016. A RND(1) is a serial number for your VIC, PET, or C-64.

At this point, you may say that if there are no true random numbers, then how can there be events that will take a user by surprise in SPACE KONG? There are ways of getting a more "random" random number, and easily. The key is the argument for RND, the argument being the number in brackets after RND. If the number is positive, BASIC will reference another list if asked. If the number is negative, BASIC will rescramble the lists. The most random number is a RND(RND(-TI)), because TI is always changing, giving a different base for the lists to be scrambled on. With it, the random numbers are based upon logic, and the timer, which is now randomly referenced to.

Now, the reason we'd like to use random numbers is so that things are not too predictable. Where would arcade games be if the fifth invader from the left always fired when it was above the second bunker? They'd be boring! Random numbers give the Commodore computer that element of unexpectedness that makes us humans so interesting.

The question is: How do you combine machine language speed with random numbers? The easy way is to use BASIC's own routine. BASIC is only slow in translation, so it will still be fast.

The RND routine is located at $E094. At the start of this routine, there are checks for the sign of the argument, and which list is referenced. I'll separate these subroutines for our use right now. JSR $E09B will generate a random number between 128 and 255, and place it in location $62 (see Program 1). JSR $EOBB will do the same thing, except it will assume you want the lists scrambled first, thus giving you a more "random" random. It simulates what happens when a negative argument is found.

Now that you have a random number, you'll probably want to generate odds, such as a one in four chance of a bomb dropping from your COSMIC EAGLE. After you get your random number, AND it with a bit pattern, and compare to get the results. In program 2, there are four possible outcomes, after the AND. They are the numbers zero to three. Comparing the outcome with one of the possibilities will give you the odds one in four or three in four.

The odds will always be calculated this way: You have X chances in 2^n where X is the number of comparisons, and N is the number of bits on in your bit pattern mentioned above. Some examples for finding N:

AND #$02 = %0010 ... there are 2^1 or 2 possibilities.
AND #$06 = %0110 ... there are 2^2 or 4 possibilities.
AND #$0E = %1110 ... there are 2^3 or 8 possibilities.

Your next question is probably: What if you want odds out of a number that is not a power of 2? Program 3 finds four possibilities, rejects one, leaving you with odds out of three.

If you want odds higher than one in 128, just store the first random number, and generate another. If the first is say 128, and the second meets further conditions like the ones outlined in previous paragraphs, then you will have your "one in 129-256" possibility. Usually, odds between 1 and 128 will be enough, but using this technique, you can get odds so high that your longshot horse will come in once in a billion years.

I hope I've helped you with a problem that has puzzled me for quite a while. Here are the example programs: (in ML format, for the VIC).

PROGRAM ONE

```
L01 JSR $E09B  Get a random number, have it placed in $62
L02 LDA #$00   Print out as an integer between 128 and
L03 LDX $62    255. See TORPET No. 17, "Non-Kernal Routines
L04 JSR $DDCD  in the VIC 20" by Thomas Henry.
L05 LDA #$20   Load accumulator with ASCII value of a space
L06 JSR $FFD2  Print it.
L07 RTS        Finished.
```

PROGRAM TWO

```
L01 JSR $E09B  Get a random number, have it placed in $62
L02 LDA $62    Load the accumulator with the random
               number
L03 AND #$03   AND it with three (%0011)
L04 CMP #$03   Is it the possiblity #3?
L05 BEQ L07    Yes...
L06 RTS        No: We're done.
L07 TAX        Print out the three,
L08 LDA #$00   using the PRLINE routine explained
L09 JSR $DDCD  in TORPET #17.
L10 RTS        We're done.
```

PROGRAM THREE

```
L01 JSR $E09B  Get a random number, placed in $62 again.
L02 LDA $62    Put it in the accumulator.
L03 AND #$03   Four possibilities.
L04 CMP #$03   Is it the fourth possibility?
L05 BEQ L1     Yes: Get another random number
L06 TAX        It's one of three possibilities, so
L07 LDA #$00   print out which one it is using
L08 JSR $DDCD  PRLINE at $DDCD again.
L09 RTS        We're done.
```

PROGRAM FOUR

```
L01 JSR $E09B  Commodore VIC Random Generation routine
L02 LDA $62    Get number produced by above routine
L03 AND #$1F   Next highest exponent minus one (than
               below number)
L04 CMP #$15   Number of random integers you want (A)
L05 BCS L01    Reject extra numbers
L06 ADC #$3A   Add lowest number wanted (B)
L07 RTS        Routine finished
```

In brackets after certain lines above, a letter appears. It represents the same number that it would represent in the BASIC formula: R = INT(A*RND(1) + B).

The explanation for the program is as follows:

In Program 4, a random number between 58 and 79 is generated. In BASIC, you would ask for INT(21*RND(1) + 58). These numbers are used in the example routine to load the accumulator with the desired random number. The program is self-explanatory, but I'll enlarge on line 3. The number of random possibilities you want is 21. 2^4 is 16, so it isn't large enough, but 2^5 is 32, and is large enough to contain at least 21 possibilities. We therefore generate 32 numbers, and reject those above and equal to 21 (giving us the numbers between 0 and 20 inclusive). Then we add the 58, and VOILA! we have our random number between 58 and 79.

Differential Relocation of Machine Code

By Harold Anderson, Oakville, Ont.

This program is on The Best Programs Disk

Any person who has tried to relocate a sizable block of machine code without the benefit of a source listing knows that this can be nearly impossible. There are some obvious fixes required, such as changing the destination address of jump statements so that they go to the same place in the relocated code as they did in the original code. You can, in fact, easily write a program to do this for you.

In practice, most machine code contains far more subtle problem points than this. For example, there may be a table of destination addresses which are used in indirect jumps. The table will not even disassemble! In the face of this or similar problems, I suggest that you had better find something more sophisticated than brute force editing of the code.

One of the solutions which works in some cases is what I call "differential relocation". Given two versions of a block of machine code assembled to run at different locations, it is possible to generate a third version to run at any desired location. The only limiting factor is that all three blocks of machine code must be separated by an integral number of pages. For example, if one block of code starts at an address equal to 47 — 51 x 256, then the other blocks must start at 47 — N x 256 where N is an integer. This limitation is not a significant impediment.

SUPERMON

One good example of where this would be useful is for generating a ROM version of Supermon. (Supermon is a public domain, extended machine language monitor for the PET.) This program comes with a relocator which will allow you to generate a version which will run anywhere in RAM. This is not much help if you want a ROM version to run at $9000, a location where there is no RAM. Use of the program listed in this article allows you to generate a version to run at $9000, starting from two versions assembled to run at $7000 and $6000. ($9000 is a ROM location whose decimal address is 9 x 4096. $7000 and $6000 are RAM locations whose decimal addresses are at 7 x 4096 and 6 x 4096.) Even better, the version to run at $9000 can be parked wherever you want it (in RAM), so that you can save it, and then take it to your friendly neighbourhood EPROM burner.

The listing is pretty well documented with its own remark statements. A brief discussion of the philosophy may be of some help. The program looks at corresponding bytes in the two initial blocks of machine code. If the bytes are the same (test made in line 205), it assumes that the value of the byte is not dependent on the address at which the code is assembled to run. It then puts this byte value in the corresponding location in the code being generated. When the program discovers a pair of corresponding locations in the initial blocks of code that contain different byte values, it assumes that the value of the byte is dependent on the address at which the code is assembled to run. In this case, it calculates the value for the code being generated by using a linear extrapolation. (Extrapolation done in line 210) Before storing the byte, it checks that it is a legal byte value, i.e., between 0 and 255. This is done in line 220. If the value is not an acceptable byte, it prints unresolvable byte at ....." on the printer and the screen. This usually indicates that the byte is past the end of the assembled code or is a meaningless inclusion in the code and can be ignored.

The listing of the program in this article is set to work with two initial blocks of code, 1400 bytes long, starting at $7000 (7 x 4096) and $7800 (7.5 x 4096). The code produced is parked at $5000 (5 x 4096) and also runs at that location. Edit lines 120 to 160 to handle different configurations. The program as shown here was used to generate a version of code to run at $5000, which happened to be impossible to do with the assembler I was using, since it landed in the middle of the source code.

I have used this program about five times to relocate quite sizable blocks of code. So far, it has worked 100% of the time. One caution: the two initial blocks of code must be IDENTICAL in all respects except running location; otherwise, you will get garbage.

```
100 REM PROGRAM NAME = DIFFRELOCATE
105 REM WRITTEN BY HAROLD ANDERSON MARCH 18, 1983
110 REM THIS PROGRRAM IS DESIGNED TO PRODUCE A
THIRD RELOCATED VERSION OF A
111 REM PIECE OF MACHINE CODE FROM TWO BLOCKS
PROPERLY ASSEMBLED TO RUN AT
112 REM A1 AND A2
118 POKE53,64:REM LOWER TOP OF MEMORY
119 OPEN4,4
120 A1 = 7.0*4096 + 00:REM ADDRESS OF FIRST  BLOCK
130 A2 = 7.5*4096 + 00:REM ADDRESS OF SECOND BLOCK
140 AR = 5*4096 + 00:REM ADDRESS AT WHICH MODIFIED
CODE WILL RUN
150 AP = 5*4096 + 00:REM ADDRESS AT WHICH MODIFIED
CODE WILL BE PUT
```

```
160 LN = 1400 :REM LENGTH OF BLOCK OF CODE
200 FOR X = 0TO LN-1
205 BY = PEEK(A1 + X):IF PEEK(A2 + X) = BY THEN 225
210 BY = BY + (PEEK(A2 + X)-PEEK(A1 + X))*(AR-A1)/(A2-A1)
220 IF BY► = 0 AND BY◄ = 255 THEN 225
221PRINT#4,"UNRESOLVABLE BYTE AT X = ";X
222 PRINT"UNRESOLVEABLE BYTE AT X = ";X
223 BY = 0
225 POKE(AP + X),BY
230 PRINTX: NEXT X
240 END
```


Putting It All Together: The Assembler

By Larry Goldstein, Bolton, Ont.

Up till now, we have talked of machine language in terms of switching patterns which can be represented as binary numbers, which in turn can be converted to decimal numbers for somewhat greater convenience. Even this is a pain, however, since it means memorizing or looking up masses of numerical code when writing a program. Since memorizing and looking up are what computers do best, it is only sensible to write a look-up program to do this conversion to machine code. Such a program is called an Assembler as is represented by Jim Butterfield's Supermon and the other members of the same family.

With an assembler, if you want to put a number into the accumulator, instead of looking up the machine code 169(D), you enter the instruction LDA (for LoaD Accumulator). Then, if you want to store this number somewhere in memory, you simply enter STA (STore Accumulator), and let the assembler look up the appropriate code. (These two machine language instructions taken together are equivalent to the BASIC POKE command.) You'll notice one catch: although you don't have to memorize numerical code, you do have to learn a new vocabulary of letter codes. These are three letter groups, and they are abbreviations of their functions, so they are called opcode mnemonics (memory helpers). These instructions comprise Assembly Language.

ADDRESSING

The instruction is only part of a machine (assembly) language command, and it is usually completed by an "address". The machine code 169 tells the microprocessor to load a number into the accumulator, but it doesn't tell what number. So, the complete instruction might be 169 83, or load the number 83 into the accumulator. These two numbers will be stored in two successive memory locations (say 830 and 831). When the program counter comes to 830, the pattern corresponding to 169(D) will be sent to the instruction register and decoded, telling the microprocessor to bring in the number **immediately** following in memory (in location 831) and put it into the accumulator. Since the storage address of this number is immediately after the address of the instruction, this is called immediate addressing. Similar instructions, LDX and LDY, allow us to put numbers in the X-register and the Y-register.

Now, say we want to transfer a value to memory from the accumulator -- perhaps we want to put a heart character (83) on the screen. On the PET/CBM, screen memory starts at 32768 with a memory location for every screen location. In BASIC, POKE 32768,83 will put a heart in the first screen location and POKE 33107,83 will put the heart somewhere else on the screen. In Assembler, we do this by putting the number 83 into the Accumulator (or the X- or Y- register) and then storing it in the appropriate screen memory location. But now the catch. Recall that memory locations are 8 bits (1 byte) each, and can hold numbers only up to 11111111(B) or 255(D). Storing numbers, including addresses, above 255 requires the use of more than a single byte of memory. Addresses above 255 are stored in two parts, allowing the use of 16-bit addresses, so the largest address usable by the 6502 and 6510 is 1111111111111111(B) or 65535(D) (i.e., 64K). The address 33107 translates to 1000000101010011(B), which is stored as the two 8-bit fragments, 10000001 and 01010011 (129(D) and 83(D)). To make matters worse, these are stored in reverse order, 83, 129. So, in order to specify an address, you must (1) convert it to binary notation, (2) break the binary number into two 8-bit fragments, (3) convert each fragment into decimal, (4) store these fragments in reverse order (called LOBYTE/HIBYTE order). Again, the assembler can help us out, but it usually calls for another compromise from us, the use of the dreaded...

HEXADECIMAL NOTATION

Just as the decimal system is based on powers of 10 and the binary system on powers of 2, so the hexadecimal system is based on powers of 16. The right-most units digit is used to count from 0 to 15, the second digit represents multiples of 16

(or 4096). Since we have digits only from 0 to 9 readily available, the values from 10 to 15 are represented by the letters from A to F. It conveniently turns out that a 16-bit binary number can be represented by a 4-hex-digit (hit?) hexadecimal number and, furthermore, two hex-digits correspond to exactly 1 byte. Going back to 33107, it converts to 8153(H) which will be stored (in LOBYTE/HIBYTE order) as 53(H) (or 5x16 + 3 = 83(D) and 81(H) (or 8x16 + 1 = 129(D)). The advantages of using hexadecimal are (1) the ease of dividing large numbers into their 1-byte fragments and (2) the more convenient size with each hex-digit representing 4 bits. Note that the computer does **not** use hexadecimal numbers any more than it uses decimal; the assembler (or machine language monitor) converts hex. values into binary, and these are used. More expensive assemblers will accept decimal addresses and do all the conversions for us, but the Supermon family needs to be fed hexadecimal. By the way, it is usual to show decimal numbers just as is and precede hex numbers by "$".

To convert from Decimal to Hexadecimal, you can do successive divisions by 4096, 256 and 16, or you can use a look-up table, or BASIC-AID, or this little program:

```
10 HH$ = "123456789ABCDEF"
20 INPUT "ENTER DECIMAL NUMBER";D
30 IF D◄0 or D►65535 THEN PRINT "OUT OF RANGE.":GOTO
   10
40 PRINT"$";:FOR I = 3 TO 0 STEP -1
50 HEX$ = "0": DIV = 16 I: IF D◄DIV THEN 80
60 Z = INT(D/DIV): D = D-Z*DIV
70 HEX$ = MID$(HH$,Z,1)
80 PRINT HEX$;:NEXT
90 PRINT:PRINT:PRINT:GOTO 20
```

So far, we have been talking about PET/CBM models. For the unexpanded VIC, the beginning of screen memory is at 7680 or $1E00, and for the C-64 it's 1024 or $0400.

USING THE ASSEMBLER

Now let's make all this work. First load in your monitor/assembler program and RUN it. After a few seconds, you will see a display of the contents of the microprocessor registers, an address to call with a SYS command to get back to the monitor (write it down) and the cursor blinking next to a period. Let's try to put a heart on the screen using the steps outlined above, and let's store the program starting at memory location 830 ($033E). Begin with the instruction to load the value 83 ($53) into the accumulator. It looks like this:

```
.A 033E LDA #$53
1  2  3  4
1--Assemble
2--at this memory location
3--the instruction, Load the Accumulator with....
4--this numerical value (i.e. Immediate Mode Addressing)
```

When you press RETURN, the appropriate numeric code is entered in memory locations $033E and $033F, and the next usable location is displayed as:

```
.A 0340
```

Now complete the line as:

```
.A 0340 STA $8150   (CBM/PET)
.A 0340 STA $1E90   (VIC)
.A 0340 STA $0490   (C-64)
```

and press RETURN. In this case, the "address" of the instruction is a memory location, not a numerical datum, and this is indicated by the omission of the # sign. Specifying the actual location in which the value is to be stored (or from which it is to be retrieved) is called Absolute Addressing. The last line of the program is

```
.A 0343RTS
```

which gets us out of machine language (in this case).

Now type RETURN, X and RETURN to get out of the assembler. To run our tiny program, enter SYS 830 and expect to see a heart appear somewhere on the screen and the cursor to reappear. You can get back into the assembler by calling the SYS address you noted down earlier, then you can expand the program to put all kinds of symbols all over the screen. Or you could enter values in color memory at $9600 to $97FF (VIC) or $D800 to $DBE7 (64).

Although we can make things happen very quickly by building up long routines of this sort, the programs are needlessly long and inefficient, and the programming and typing are extremely tedious. What we need now is a way to get the program to take care of the repetition itself with something like the FOR...NEXT loop in BASIC. But that's for next time.

DEBUGGING programs is a little like swatting mosquitos. Eliminate one BUG and you may find 20 at its funeral.

Ylimaki

6502/6510 ML Instructions

By Vince Sorensen, Regina, Sask.

When I first read that the C64's new 6510 chip had extra I/O pins, I wondered if the 6510 would also have more ML commands, in order to access these pins. After investigating, I found that both the 6502 (in the VIC, PET, and previous Commodore efforts) and the 6510 have more usable commands than previously thought.

This obviously has occurred to more people than just myself. About a month after discovering a few commands, I came across an article in the October 1983 issue of COMPUTE! that detailed some of the commands that I had found, and some that I hadn't. So, I decided to combine both my work and the additional information provided by Joel C. Shepherd in COMPUTE!, and provide fellow users with a comprehensive list of unofficial 6502/6510 machine language commands.

Generally, each bit in a 6502 opcode represents a different instruction type or addressing mode. An opcode byte can be broken down with the three most significant bits representing type, and the other bits giving the mode. There are, of course, exceptions, but we can still postulate what the 6502 thinks it sees when it encounters an undefined number.

For a list of documented opcodes, find the MOS PROGRAMMING MANUAL or the C64 PROGRAMMER'S REFERENCE MANUAL. Any hex opcodes not given here, and not documented in one of those books, can be placed in one of the following categories: DETRIMENTAL (your machine crashes); NON-EFFECTIVE (has no effect, except to skip one to three bytes); INCONSISTANT (different results from the same parameters, repeatedly and randomly); and REPETITIVE (this command is identical to another command, including type and mode.)

Here are the unofficial OPCODES in the following format: HEX #3: (Mnemonic, Addressing Mode) Brief Description, Other Addressing Modes.

04: (NTW,implied) This byte and byte after ignored.
1B: (NTH,implied) This byte and two bytes afterward ignored.
07: (SLO, zero page) These four commands shift memory left then OR the accumulator with this memory.

Other modes:

0F (absolute)
17 (zero pg,x)
1F (absolute,x)
27: (RLA, zero page) These six commands roll a memory location left, then ANDs the contents of the accumulator with the result.

Other modes:

23 (indirect,x)
2F (absolute)
37 (zero pg,x)
3B (abs.,y)
3F (abs.,x)
43: (SRL, indirect,x) This command shifts memory right, then loads the accumulator with the result.
4B: (SRA, immediate) This command shifts the contents of the accumulator right, and then ANDs the result with immediate data.
47: (SRE, zero page) These four commands shift memory right and then EOR the accumulator with the shifted memory.

Other modes:

4F (absolute)
57 (zero page,x)
5F (absolute,x)
67: (RRA, zero page) These 6 roll memory right, and add with carry to the accumulator.

Other modes:

6F (absolute)
73 (indirect,y)
77 (zero page,x)
7B (absolute,y)
7F (absolute,x)
87: (AAX, zero page) These three commands AND the contents of the accumulator with those of the X register.

Other modes:

8F (absolute)
97 (zero page,y)
8B: (AAX, immediate) This command ANDs the accumulator, the X register, and immediate data.
A3: (LAX, indirect,x) These seven commands load both the accumulator and the X register from the same location.

Other modes:

A7 (zero page)
AB (immediate)
AF (absolute)
B3 (indirectly)
B7 (zero page,x)
BF (abs.,x)
C3: (DCP, indirect,x) These six commands decrease a memory location, then compare it with the contents of the accumulator.

Other modes:

C7 (zero page)
CF (absolute)
D3 (indirect,y)
D7 (zero page,x)
DF (absolute,x)
CB: (XAS, immediate) This command first ANDs the X register with 8, and then subtracts the data immediately following.
EB: (SOC, immediate) This command subtracts one from the accumulator, then carry, and then the data immediately following.

E3: (ISC, indirect,x) These seven commands increase a memory location by one, and then subtract the result from the accumulator, with carry.

Other modes:

E7 (zero page)
EF (absolute)
F3 (indirect,y)
F7 (zero page,x)
FB (abs.,y)
FF (absolute,x)

Note that the results for all of the above commands are stored in the accumulator except as follows: AAX results are stored in memory location given by data after AAX (immediate mode) results, and LAX results are placed in both the accumulator and in the X register. DCP results are shown in memory and in the processor's status byte. XAS results are put in the X register only.

Presently, only a few assemblers will accept these new mnemonics. These assemblers allow either new commands to be defined or .BYT commands which let the user put numeric data (in this case our new commands) into his program. However, everyone should be able to think of a way to get these new opcodes.

Until next time...happy programming!!!

CBM Conditional Assembly

By Mark Niggemann, Ames, Iowa

GOAL OF ARTICLE

To inform other programmers that use the CBM assembler on the C64 about an undocumented feature, that of conditional assembly.

The CBM assembler for the C64 is a fairly inexpensive, yet rather complete, assembly development system. It has to be one of the best assembly development systems for the price (under $20 in most places). Despite Commodore's good intentions on selling good software at a low price, they did leave out a very important fact about the CBM assembler on the C64. The CBM assembler on the C64 has what is called Conditional Assembly.

Conditional assembly is a very useful means of creating specialized coding using the same source file(s). For example, say, if you have to write a display driver for 40, 64 and 80 columns. Most of the coding for all of them is identical, with the exception of a small patch. Instead of having three separate source modules, you can use Conditional Assembly and include all three. If you use as a variable label COLWID, you just change whatever COLWID is and you have changed the configuration of the display driver. Conditional Assembly is also very useful in the expansion of MACRO's, but I'll let you discover more about that on your own.

HOW DO YOU DO IT?

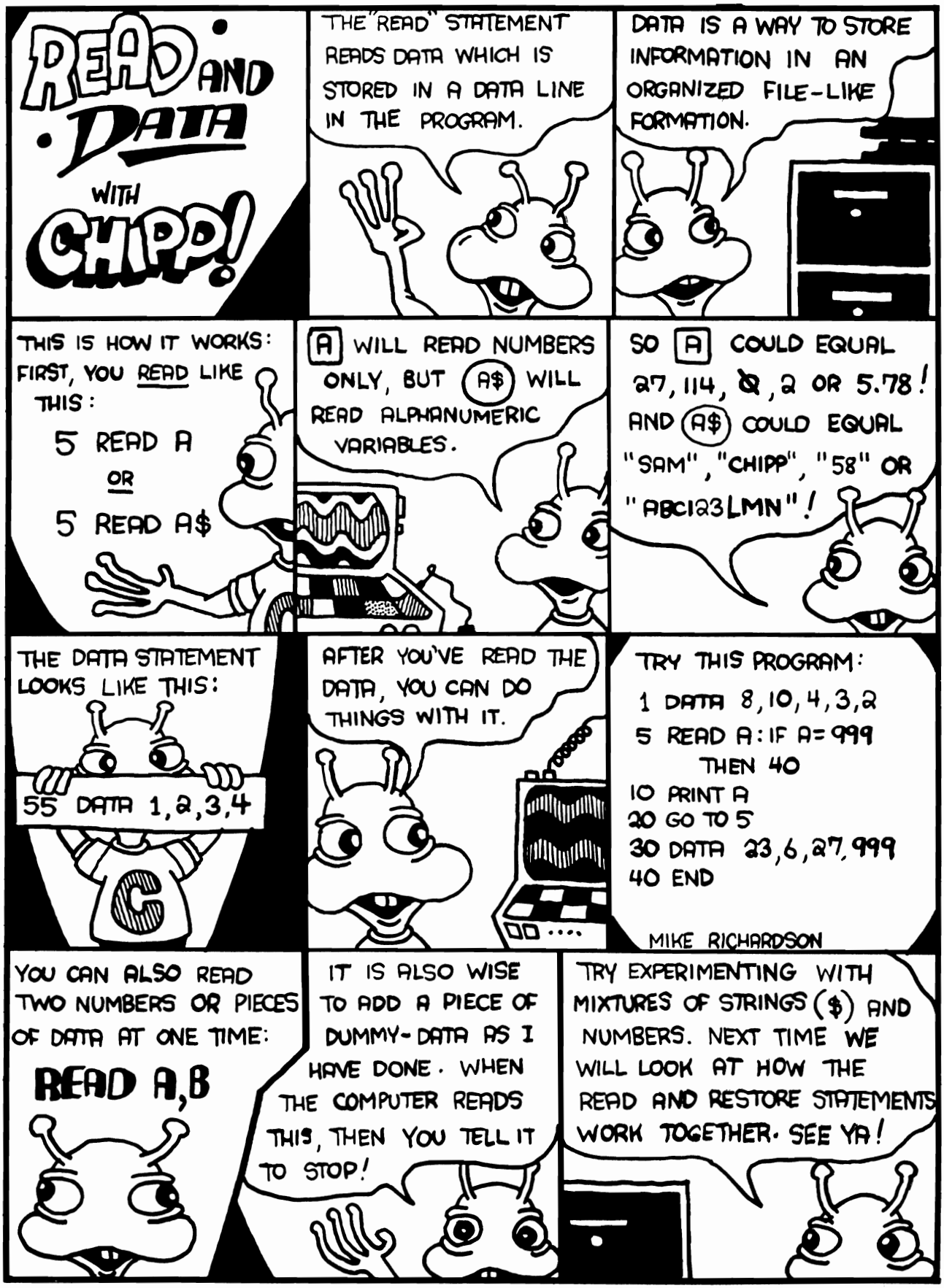
On the CBM assembler, you have two conditional assembler pseudo-ops, .IFE and .IFN. .IFE is IF Equal to zero. It means, if the label expression is equal to zero, it will assemble the conditional block of code. .IFN is IF Not equal to zero. If the label expression is not equal to zero, it will assemble the conditional block. A label expression can be just a single label or an expression of labels that are joined by + or -.

Examples:

```
.IFE COLWID-40  ◄
. ;The code would go
. ;right here instead
. ;of these dots
►
.IFN FLAT  ◄
.
. ;Ditto here
.
►
```

Notice the '◄' on the line following the label expression. This must appear there; otherwise, you will get an assembly error. Also note the '►' on the line after the last line of code in th conditional block. The '►' must be in the first column on the line after the last line in the conditional block. If there isn't one or you don't have it in the first column, then the assembler will crash. Other than that, there isn't much to using conditional assembly.

Conditional assembly is a very powerful tool in the bag of assembly language programming tricks, and it is well worth the effort of this article to bring to light what has been until now an unknown feature of the CBM assembler.


READ AND DATA WITH CHIPP!
THE "READ" STATEMENT READS DATA WHICH IS STORED IN A DATA LINE IN THE PROGRAM.
DATA IS A WAY TO STORE INFORMATION IN AN ORGANIZED FILE-LIKE FORMATION.
THIS IS HOW IT WORKS: FIRST, YOU READ LIKE THIS:
5 READ A
OR
5 READ A$
A WILL READ NUMBERS ONLY, BUT A$ WILL READ ALPHANUMERIC VARIABLES.
SO A COULD EQUAL 27, 114, Ø, 2 OR 5.78! AND A$ COULD EQUAL "SAM", "CHIPP", "58" OR "ABCI23LMN"!
THE DATA STATEMENT LOOKS LIKE THIS:
55 DATA 1,2,3,4
AFTER YOU'VE READ THE DATA, YOU CAN DO THINGS WITH IT.
TRY THIS PROGRAM:
1 DATA 8,10,4,3,2
5 READ A: IF A=999 THEN 40
10 PRINT A
20 GO TO 5
30 DATA 23,6,27,999
40 END
MIKE RICHARDSON
YOU CAN ALSO READ TWO NUMBERS OR PIECES OF DATA AT ONE TIME:
READ A,B
IT IS ALSO WISE TO ADD A PIECE OF DUMMY-DATA AS I HAVE DONE. WHEN THE COMPUTER READS THIS, THEN YOU TELL IT TO STOP!
TRY EXPERIMENTING WITH MIXTURES OF STRINGS ($) AND NUMBERS. NEXT TIME WE WILL LOOK AT HOW THE READ AND RESTORE STATEMENTS WORK TOGETHER. SEE YA!

Other Languages Programming

PAGE

Integer BASIC Compiler Hint 104
G.R. Walter, Proton Station, Ont.
An integer compiler gives you nearly machine language speeds without having to know machine language. This article gives you some hints on how to more easily use the compiler.

Logo for the Commodore 64 105
Dr. Efraim Halfon, Burlington, Ont.
Here is a very thorough description of that programming language educators hear so much about.

Simon's BASIC 109
Dr. Efraim Halfon, Burlington, Ont.
Here is a description of an extended BASIC language that is available from Commodore for the 64.

What Really Is CP/M? 116
Steve Rimmer, Toronto, Ont.
The only operating system available on many computers is thoroughly described.

CP/M Now A Reality With Commodore 120
Tony Ning & Rick Denda, Toronto, Ont.
An independent implementation of CP/M for Commodore computers is described.

CP/M On The C64 122
Fred Wallace, Windsor, Ont.
The Commodore implementation of CP/M is described.

CP/M File Transfer for C64 124
Wm. Kendall, Baltimore, MD
How to get CP/M files to download from any computer into your Commodore 64.

Integer BASIC Compiler Hint

By G.R. Walter, Proton Station, Ont.

This program is on The Best Programs Disk

The Integer BASIC Compiler is (for you readers) who don't know) a program which will take a program written in BASIC and convert it to machine language — which usually results in an 80-100 time speed-up. This is super, but the program does have one major problem for which I have found a partial solution. The problem is that your BASIC program has to be typed into the Integer BASIC Compiler's editor, with all the resultant typos and errors. Wouldn't it be great if there was some way of directly transferring your PET BASIC program into Integer BASIC? Well, now there is a way. The method I use goes as follows:

1) Take the PET BASIC program and, using some type of BASIC Aid, change all occurrences of ◄► to #, fix all other comparison operator references (◄=, =►, etc.), move all DATA statements to the beginning of the program (followed by a RESTORE), and change all function and array variables to the special format that the Integer BASIC Compiler requires. After this is all done, SAVE your changed program (in case something goes wrong).

2) LIST your program to disk, giving it some unique 4 or 5 character name.
.ie

```
OPEN8,8,8,"0:NAME,S,W":CMD8:LIST
PRINT#8:CLOSE8
```

3) LOAD and RUN the converter program (LISTed at the end of this article). When it asks you "NAME ?" just type in the name and press RETURN. While your program is being converted you will see printed on the screen what is being printed onto the disk (don't worry if you see some "funny" characters, that's normal). The conversion may take some time.

4) LOAD the now converted program into your Integer BASIC Compiler and add the set-up instructions (eg. the @STRINGS type commands).

Then you come to the step you always have to do : COMPile your program, test your program, and edit your program until it works bug free.

PROGRAM NOTES

The conversion program just converts a program ASCII file (the program listing on the disk) to the format required by the Integer BASIC Compiler. The format used on disk is :

line number (RETURN)

rest of program line (RETURN)

next line number ..... and so on until the last line number and last program line, which are followed by :
! (RETURN)
! (RETURN)

The conversion program also converts the special characters (such as comma - ",", etc.) to the appropriate Integer BASIC code.

The program was written on a PET and is set up for the way that the "green screens" list their programs. If you wish to use it on a C64 you will have to change it to take into account the different listing format. One example of this different format is that on the C64 there is no space printed before the line number, while on the green screens a space is always printed before the line number.

```
10 REM BASIC ASCII LISTING (ON DISK) TO INTERGER BASIC COMPILER CODE CONVERTER
100 INPUT"␣NAME ";N$
110 OPEN8,8,8,N$
120 OPEN9,8,9,"E "+N$+",P,W"
130 GET#8,B$,B$
140 GET#8,B$
145 IFB$<>" "AND (B$<"1"ORB$>"9") THENPRINT#9,CHR$(13)"!":CLOSE8:CLOSE9:GOTO220
150 PRINT#9,CHR$(13);:PRINT:IF B$<>" "THEN GOTO 165
160 GET#8,B$
165 IFB$<>" "THENPRINT#9,B$;:PRINTB$;:GOTO160
170 PRINT#9,CHR$(13);
180 GET#8,B$:IFB$=":"THENB$="♠"
190 IFB$=","THENB$="◆"
200 IFB$=CHR$(13)THENGOTO140
210 PRINT#9,B$;:PRINTB$;:GOTO180
220 END
READY.
```

Logo for the Commodore 64

By Dr. Efraim Halfon, Burlington, Ont.

Logo, the language developed at the Massachusetts Institute of Technology in the 1970's, is now available also for the Commodore 64. Up to now, it was only available for other microcomputers such as the Apple, Texas Instruments and Radio Shack. Logo is also well-known as turtle graphics, because of its graphics capability which uses a turtle as an indicator of where lines should be drawn. Over the years, Logo has become immensely popular, especially in the school systems, and now several clubs and magazines deal only with Logo (see Table 1 for some names, addresses and references). Logo, however, is not only turtle graphics; it is also mathematics, words and lists analysis, sprites, graphic characters and simple music. The whole 64K of memory are used by Logo with about 14K available to the user for storing the procedures (or programs).

WHY IS LOGO SO SUCCESSFUL?

The main feature of Logo is that the users, often children of ages three to 15, can teach the computer, or program one's ideas very easily with turtle graphics. The computer is not used as a driller, where the user is only a passive spectator feeding answers to the computer, but as a tool which is under complete control of the user. Children then can easily learn how to control the computer, and by doing things learn the basics of programming, without being afraid of being mistaken. In Logo, there are no mistakes, only learning through debugging. For example, to move the turtle forward, the command is FORWARD n, where n is a number. FD 100 (the abbreviated form) will move the turtle 100 steps.

The Turtle can be turned right with the command RIGHT 90, where 90 is the number of degrees in a right angle, or LEFT, or BACKWARDS, while writing a line (PENDOWN) or not writing (PENUP). Once these basic commands are learned, the next step is to draw figures, for example, a square. The user defines a procedure, called, for example, TO SQUARE, or TO B2D2, or any other name. Logo then enters edit mode. A program to draw a square would then be:

```
TO  SQUARE
FD 100
RT 90
FD 100
RT 90
FD 100
RT 90
FD 100
RT 90
END
```

A shorter way of drawing a square of any size, would be, however:

```
TO SQUARE :N
REPEAT 4 [FD:N RT 90]
END
```

The command REPEAT 4 means to do 4 times the commands in the square parentheses and :N is the length of the square side.

Once defined, the procedure SQUARE is now available to be used in other procedures. For example:

```
TO  FLOWER :LEAVES :N
REPEAT LEAVES [SQUARE :N RT 360/;LEAVES]
END
```

draws a schematic flower with :LEAVES number of schematic square leaves of length :N. With very few combinations of procedures, complex graphic figures can be created. For example, try PENCOLOR 0 FLOWER 90 90 PENCOLOR 1 FLOWER 10 30.

Usual projects include a face, a person, a house, a propeller, a flower, a car, etc. Incidentally, the name turtle originates from a mechanical device that was developed at MIT when the language was developed. At the time, early 70's, the microcomputers with today's capabilities did not exist, and therefore all commands were transferred to a turtle with wheels which roamed along the floor. This mechanical turtle is still used today in schools and it is particularly useful in the education of retarded children and children unable to control their body fully. By controlling the mechanical turtle via a simplified keyboard, the children can have control on the outside world, sometimes for the first time. Experiments along this line have proven very successful.

MATHEMATICS

Another aspect of Logo is its mathematical ability. Mathematical operations, such as addition or multiplication, can be integrated with turtle graphics to provide a visual relation between numbers and their geometrical meaning. Plotting of curves such as circles, paraboles and hyperboles take only a few commands. Analytical geometry thus becomes much easier to understand through continuous feedback between the user and the computer.

WORDS AND LISTS

Even though Logo is often associated with turtle graphics, its power also lies in its ability to handle words and lists in a manner that the computer responses seem to show intelligence. Several computers on the market cannot handle lists and words because of memory limitation. The Commodore 64 with 64K of memory is very apt to handle this part of Logo. Indeed, it would have been a pity if words and lists were left out.

SPRITES

The Commodore 64 is well-known for its ability to handle eight sprites at a time and for its ability to play music. Logo also incorporates this feature, even if in a limited way because of memory constraints. Each of the eight sprites, 0 to 7, can be defined and moved independently. Sprite 0 is the turtle shape. Logo incorporates a Sprite editor which can be used to design sprites. The Sprite editor is fairly simple to use and, once edited, the sprites can be saved in memory for future use. Animation of sprites can be done by modifying slightly the shape of a sprite and then by displaying them one at a time. Animation, however, limits the number of different available sprites since only eight can be kept in memory. As you can recall, the Commodore 64 in its regular form can store a large number of sprites in memory, even if only eight can be displayed on the screen. Logo uses much of the 64K of memory and therefore a compromise must be made in some applications between animation and a choice of different sprites. However, if in a program execution some delay is allowed, then new sprite shapes can be read from disk. Sprites are also a feature of the Texas Instruments microcomputer, which can handle 32 sprites at a time on the screen. However...the TI microcomputer is no longer produced.

MUSIC

Only one voice can be used at a time to play a tune. Music in Logo, however, is not used to produce complex and rich (three voices) melodies, as it is possible to do with the standard configuration and direct access to the music chip. The function of music in Logo is to teach music characteristics, such as pitch and tempo. More advanced users can also design their own sound envelope to simulate different instruments. Using Logo, children and other users can learn about notes, relation among notes, composition and musical phrases. Since only few commands are used, such as PLAY and SING, the user's attention can be focused on the music rather than on PEEK's and POKE's. Music in Logo is a special feature of the Commodore 64. As far as I understand, no other microcomputer has the same musical ability.

DOCUMENTATION

When you buy the Logo package from a dealer, you receive a book, two floppy disks, and, most important, a postcard to Commodore. The postcard can be used to request a backup copy of your Logo floppy disk, if by any chance it gets damaged. Price for replacement is $5.00 U.S. One floppy disk contains Logo and one contains a large number of indispensable Logo routines (back up this disk right away before doing anything else).

These Logo routines contain a large number of demo programs, a number of sprite shapes, and a number of utility procedures which make the life (and programming) of Logo users much easier.

Among these utility programs, you can find procedures to draw with the joystick, to edit sprites, to play music, to play games, to draw pictures, to understand the Logo manual with living color example programs. The development of these routines must have taken a lot of effort and I am quite happy with the results.

The Logo manual is very well-written. The editing was very accurate, and I found only three misprints. Reading is very pleasant, and all commands, primitives in Logo, are clearly and well-explained. The manual is very comprehensive, indeed, much more comprehensive than the manual of the Apple computer, even if the two versions of Logo are very similar. The only unavoidable drawback in appending machine language programs to Logo is that the machine language routines must share space with the sprites. The locations OCOO to ODFF are used by the eight sprites and the locations OC40-ODFF are available for machine language; thus, one or more sprites may have to be released if this feature is chosen. An assembler procedure is also included in the utility disk to help create fast machine language procedures. Overall, Logo is a very slow language, because of the large amount of preprocessing done to make the language easy to use. Speed, however, is usually not a consideration in Logo applications. The last part of the manual is dedicated to make the system flexible by using several options. The average user will not be concerned with these features at the beginning.

Commodore graphic characters from the keyboard are all available for use. The manual,

however, does not emphasize this important aspect. For this purpose, the user can use the procedure STAMFD :D :CHAR, where D is the distance the turtle moves and CHAR is the character which must be stamped. To let the Logo interpreter know that CHAR is, for example, a letter, a " must precede the letter.

Another useful editing command is SHIFT-INST which quotes the following character. For example, SHIFT-INST followed by CTRL-2 (white) will insert the special character for changing the color to white. Thus,

```
TO REV
PRINT "SHIFT-INST CTRL-2 HELLO SHIFT-INST CTRL-7
END
```

changes colors during execution and then returns to the default color.

COMMODORE 64 LOGO: GENERAL REMARKS

This version of Logo is based on the one originally developed for the Apple II and produced by Terrapin Inc. This version is better than the one for the Apple since it offers 29% more user memory plus some unique capabilities such as sprites, music and graphics characters. The graphics screen is in high resolution, with all the 16 foreground and background colors. Care, however, must be taken when choosing the different combinations of colors, since some may not mix well. The Commodore Reference Guide offers some suggestion to the best combinations. Text and graphics can be mixed on the graphics screen. The turtle can go 129 steps up before wrapping around, and 130 steps down before wrapping around the top. The NOWRAP command eliminates the wrap-around capability if so wished. The turtle can go 160 steps to the left and 159 to the right.

The advanced .OPTION command allows the user to control some of the ways the system operates. Most beginner users would not probably use this feature at the beginning, but it may be quite useful in some instances. Among .OPTION primitives, there are DEPOSIT (POKE) and EXAMINE (PEEK) commands to look at particular memory locations, and JOYSTICK which outputs a number that is the sum of the switch values, when the option N = 1 is chosen. This mode is documented in the Commodore Programmer's Reference Guide.

The high resolution graphics screen can be changed to DOUBLECOLOR mode that allows two colors per 8x8 pixel region, instead of just one. The resulting colors will be much richer and easier to see, but drawings are less precise because horizontal lines are thicker, i.e., horizontal resolution is reduced to half. Once a program has been run and a picture created, this can be saved on disk with SAVEPICT. Another useful command to use in graphic mode is the SPLITSCREEN command. The bottom lines (the number can be chosen arbitrarily with a maximum of 13) are used to display the commands while the turtle moves on the graphics screen. On the graphics screen, a useful feature to create the illusion of three-dimension is to use sprites to draw, for example, clouds, cars, trees, etc. The lower the number of the sprite the higher the priority of display and, therefore, one can program a cloud moving in front of the sun or a car driving by and in front of a tree or a house.

Many other system primitives are available in Logo. These primitives can be compared with those of other computers on the market (see BYTE issue, August 1982).

In conclusion, the Logo version for the Commodore 64 compares well with others on the market, and, given its words and lists, sprites, music capabilities and its high-resolution graphics, I believe it to be superior to all others. Users of all ages will enjoy its capabilities, the powerful ideas and its framework directed to problem-solving and computer literacy.

ACKNOWLEDGEMENTS

I would like to offer my most sincere and appreciative thanks to Mr. Laurie Fountain of Commodore Canada for his time and assistance. He gave me access to the Commodore Logo and its manual before marketing in Canada, and provided explanations of programming details. Thank you very much.

TABLE 1

References, Books and Magazines for teaching and using Logo

Abelson, H. and A. diSessa. Turtle Geometry, Cambridge, MA: MIT Press (1981)
Beardon, D. One, two, three, my Computer and Me: a Logo funbook for kids, Reston, VA: Reston Publishing Company (1983)
Beardon, D., K. Martin and J. Muller. The Turtle's sourcebook, Reston, VA: Reston Publishing Company (1983)
Burnett, J.D. Logo: an introduction, Morristown, NJ: Creative Computing (1983)
BYTE magazine, Logo issue, August 1982

Goldenberg, E. Special Technology for Special Children, Baltimore: University Park Press (1979)
Minnesota Educational Computing Consortium (MECC) Apple Logo in the classroom. MECC - Distribution Centre, 2520 Broad Dr., St. Paul, MN 55113
Papert, S. Mindstorms; children, computers and powerful ideas, New York: Basic Books (1980)
Thornburg, D. Discovering Apple Logo, Reading, MA: Addison-Wesley (1983)
Watt, D. Learning with Logo/Learning with Commodore Logo, New York: BYTE Books-McGraw Hill (1984, in press)
Young People's Logo Association, 1208 Hillsdale Drive, Richardson, TX 75081. This association is one of the leading groups in educational Logo. The YPLA has members throughout the world. Young people 18 and under can receive their newsletter, Turtle News, at no charge. YPLA asks adults to contribute US $25 per year to receive Turtle News plus the Logo Newsletter, which is oriented towards adults. YPLA also has exchange disks and tapes at US $10 each or at no charge when exchanged for a working program.
The National Logo Exchange, P.O. Box 5341, Charlottesville, Virginia 22905, publishes a non-commercial newsletter monthly from September through May at a subscription price of US $25.
Computer magazine has a regular feature called Friends of the Turtle, with the latest news on Logo.

I JUST DISPROVED EINSTEIN'S THEORY OF RELATIVITY

Simon's BASIC

By Dr. Efraim Halfon, Burlington, Ont.

When a 16-year-old teenager bought his Commodore 64 in England he wondered why Commodore had not provided a BASIC language that could handle high resolution graphics, music and text rather than having to rely on PEEK's and POKE's. He then set to work and developed over 100 new BASIC commands for the C-64. This addition will provide Commodore users with programming capabilities better than those available on the Radio Shack Colour computer and Texas Instruments among other personal computers.

The extra commands of SIMONS BASIC fall into twelve broad areas. The attached table 1, shows an abbreviated description of all commands. The programming aids facilitate BASIC programming, for example AUTO automatically generates program line numbers and RENUMBER automatically renumbers all the program lines. This renumbering does not include the GOTO or GOSUB numbers but SIMONS BASIC has the capability of calling subroutines by name (PROC command) and therefore this is not a problem. In the subroutines, variables may be made LOCAL so that the same name can be used as in the main program without having to change variables' names in the subroutine. The GLOBAL command restores the original values to local variables. The MERGE command can be used to merge two programs, one in memory and one saved on disk or tape. The OPTION 10 command highlights SIMONS BASIC commands while the program is listing. The OPTION command only works with the parameter 10 and other numbers do not seem to make any difference. The KEY command enables the user to program the function keys quite easily. I liked this feature very much since it makes programming much easier and concise.

In addition to programming aids there are program debugging aids such as the command TRACE that displays on the screen the number of the program line being executed. The command DUMP displays values of all non-array variables. These debugging aids are quite useful. For security-minded programmers the command DISAPA marks lines that should not be listed. The command SECURE executes the security part and lines marked for security can never be listed. The manual suggests that the programmer keeps for his own safety a non-protected version. The only disadvantage I found is that all lines that need to be protected must be marked with DISAPA which, for some users, may be the whole program.

Character strings can be manipulated with the INSERT, INST, PLACE, DUP and CENTRE commands. Other commands such as INKEY and FETCH provide control over which inputs can be accepted by the program. All these commands improve programming flexibility. For mathematical programs six new commands, MOD, DIV, FRAC, %, $ and EXOR can be useful. However, I do not expect that the average user would have much use for MOD and DIV (see Table 1) but they are nice to have for programmers accustomed, for example, to FORTRAN.

Two disk commands DISK and DIR are also provided. The DISK command saves the effort of programming the OPEN, PRINT# and CLOSE commands when some disk operations are required. Thus, disk initialization, formatting and file scratching can be performed with one command. The DIR command enables all, or a selective part, of a diskette directory to be displayed on the screen. This command replaces the LOAD "$",8 command.

The high resolution graphics commands are really excellent but for a lack of consistency on parameter order in the various commands. (Table 2). The high resolution commands allow standard high resolution and multi-colour modes.

In high resolution the screen is 320 pixels wide and 200 long, in multi-colour mode 160 pixels wide and 200 long. Plotting and background colours can be chosen and changed rapidly and easily, the HIRES and MULTI commands allow a rapid change between standard high resolution and multi-colour. Different parts of the screen can be in different colours and different modes. In high resolution mode different colours can be programmed in different parts of the screen so that high resolution plots with several colours are possible. Several standard geometrical figures can be plotted on the screen, rectangles, circular shapes and shapes of any form. Once drawn, these shapes can be PAINTed. The high resolution screen can also include text and the CHAR and TEXT commands print characters and character strings on the graphic screen, respectively. The CSET command with option 2 allows the display of the last high resolution screen; this feature is quite useful for games.

The only problem that I found is that the user cannot save on disk a high resolution screen once this is programmed. In fact, SIMONS BASIC could be used to program fast arcade games

which require several high resolution screens. At present only the last one can be immediately recalled with the command CSET 2. If another screen is needed in a game, it must be drawn anew. Drawing is fast but not immediate as it is required in an arcade style game. For example, when I was introduced to SIMONS BASIC the first time, it took me only about three minutes to program a high resolution screen for a game, which previously took me over ten agonizing hours with regular BASIC and PEEKs and POKEs. Unfortunately I could not take it home with me and this was disappointing. The fact is that to save memory space most of the RAM memory used is under ROM. For example, since the high resolution screen is under the kernal, this memory area can only be POKEd but not PEEKed, thus the inability of saving a screen. The rest of SIMONS BASIC is in the 8K reserved for the cartridge and in the RAM under the regular BASIC ROM. Thus, SIMONS BASIC only reduces the regular BASIC memory by 8K. Quite an accomplishment!

Not all graphics commands are for the high resolution screen, some are for the low resolution screen. For example the FLASH command flashes screen colours at variable speeds, from very slow to maddeningly fast. The same is valid for the border colour (BFLASH). The FCHR, FCOL, FILL, MOVE, and INV commands are used to fill areas of the screen with characters and colours and to move data from one part of the screen to another. I really enjoyed the SCROLL commands. The user can define several windows and in each window scrolling is allowed up, down, right and left. On a screen all four scrolling directions can take place simultaneously. Visually the scrolling capability is excellent.

Sprites and special characters can be easily programmed with the nine special commands. Sprites can be created within a program, stored and modified. User's specific graphic characters can also be easily programmed. Even if several software programs now exist on public domain to produce sprites, it is useful to have specific commands that can be easily used within a program. SIMONS BASIC also includes four structured programming commands, such as IF THEN ELSE, REPEAT UNTIL and LOOP. The structured programming part also includes commands which prevent a BASIC program from crashing by trapping program errors; ON ERROR GOTO, for example, helps in program debugging.

The five music commands are all that music programmers want to compose simple and complex melodies. No more PEEKs and POKEs and complex calculations to produce the appropriate notes with the appropriate tempo (the function keys take care of all timing and note duration). With SIMONS BASIC music composition was instantly open to me. The only objection that I have is that perhaps the commands, especially the MUSIC command, are too sophisticated. To play music the average user may want to use the public domain programs ORGAN and PIANO. The latter one especially allows one to play the melody and save automatically the notes on disk or tape. With the MUSIC command all notes and the duration of each note within each voice must be individually programmed and entered through the keyboard. However, for specialized applications and games where music is important, then the MUSIC commands are excellent. For example the PLAY command can be used to play the music while the program continues its execution or to stop the program execution until the music is finished, PLAY 2 and PLAY 1 respectively. I did not find much use for PLAY 0, which supposedly stops the music, but the last note continues on. When I wanted the music stopped in a program I preferred to use the VOL 0 command.

Finally, four commands, PENX, PENY, POT and JOY allow a program to read the coordinates of a light pen, the resistance of the paddle and the direction of the joystick. These commands greatly simplify programming games and graphic applications.

Overall I found SIMONS BASIC a very good addition to the BASIC commonly provided with the C-64. From now on complex programs can be developed in BASIC since the execution of most commands is at machine language speed.

The programming and debugging aids are quite easy to learn and use and I particularly enjoyed working with high resolution graphics with an ease never before obtained on the C-64. While SIMONS BASIC will be marketed by Commodore I understand that several independent software firms have their own versions of BASIC that they plan to market soon. Some will have some features similar to those of SIMONS BASIC, probably for high resolution graphics and music, but I expect that few will have all the comprehensive commands that this package has. My recommendation would be to use it only if you can use the special capabilities. That is if, you do not particularly enjoy POKEing and PEEKing, and if you want to reduce your programming time several folds, such as happened to me. A reduction of from ten hours to a few minutes is probably worth the expense of the cartridge. The manual is very well written, comprehensive and with several ex-

amples. The error messages are clear and informative.

TABLE:1: SIMONS BASIC Commands

Programming Aids:

KEY	to assign a command to a function key
AUTO	automatically generates program line numbers at a specified interval
RENUMBER	automatically renumbers all the program lines
PAUSE	pause number of seconds
LIM	to determine the number of the screen line on which the cursor is positioned
CGOTO	to compute the line number to which the program should branch
RESET	to move data pointers to a specified line of data
MERGE	to merge two programs
PAGE	to divide a program listing into 'pages' of n lines
OPTION 10	to highlight SIMONS BASIC command while program is listed on the screen
DELAY	to vary the rate of scrolling of a program listing
FIND	to search a BASIC program for a character string on display line where it occurs

Program Debugging Aids and Program Security

TRACE	to display the number of the program line being executed
RETRACE	to resume tracing after editing a program
DUMP	to display values of all non-array variables
COLD	resets the C-64 to the start of SIMONS BASIC
OLD	reverse NEW command
DISAPA	to indicate that the code in a program line is to be hidden
SECURE	to hide all program lines beginning with DISAPA

Input validation and text manipulation commands:

INSERT	to insert one character string into another
INST	to overwrite a string beginning at a specified position
PLACE	to determine the position of a string within a string
DUP	to duplicate a character string n times
CENTRE	to centre a character string on a screen line
USE	to format numeric data, i.e. to align decimal points
PRINT AT	to print a character string at a specified location
FETCH	to limit the type and number of characters for user input
INKEY	to test for a function key input
ON KEY	to branch to a specific point in a program
DISABLE	to terminate ON KEY command
RESUME	to reinstate ON KEY command

Arithmetic operators:

MOD(x,y)	to return the remainder when one integer is divided by another
DIV(x,y)	to return the largest integer which, when multiplied by y is equal or less than x
FRAC	to return the fraction part of a number
%	binary to decimal conversion
$	hexadecimal to decimal conversion
EXOR	to perform exclusive OR between two numbers

Diskette commands:

D I S K	to open a diskette channel and then close it when the operation is executed
D I R	to list some or all of a diskette directory

Graphics:

HIRES	to initialize high resolution graphics mode and select plotting colour and screen background colour
REC	to draw a rectangle
MULTI	to initialize multi-colour graphics mode and select three plotting colours
LOW COL	to change plotting colours
HI COL	to revert back to originally selected plotting colours
PLOT	to plot a dot

TEST	to return the state of a screen location, dot plotted or not
LINE	plot a line
CIRCLE	to plot a circular shape
ARC	to draw an arc of a circular shape
ANGL	To draw the radius of a circle
PAINT	to fill an enclosed area with colour
BLOCK	to draw a fully shaded block of colour
DRAW	to design a shape of any form
ROT	to rotate a shape
CSET	to select one of the character sets or recall and display the last high resolution screen
CHAR	to print single characters on a graphic screen
TEXT	to print a character string on a graphic screen
COLOR	to set screen background low resolution

Screen Manipulation

FLASH	to flash a screen colour at variable speeds
OFF	to turn off FLASH
BFLASH	to flash border screen at variable speeds
BFLASH 0	to turn off BFLASH
FCHR	to fill an area of the screen with a character
FCOL	to change a character colour
FILL	to fill a defined area on the screen with a specific character in a particular colour
MOVE	to duplicate a section of screen data on another part of the screen
INV	to inverse a specifed screen area

Scrolling:

LEFT, RIGHT, UP, DOWN	to scroll an area of the screen within a window in any direction. Also several parts of the screen can scroll in different directions at the same time
SCRSV	to store data from a low resolution screen on disk or tape
SCRLD	to display screen data previously stored
COPY	to produce a hard copy of a graphic screen
HRDCPY	to print low resolution screen data

Sprite and User-defined Graphics:

DESIGN	to allocate memory space for a MOB (moveable object block or a sprite)
@	to set up the design grid for MOB
CMOB	to set up colours for multi-colour MOB
MOB SET	to set a MOB, i.e. MOB initialization
MMOB	to display and/or move a MOB
RLOC MOB	to move a MOB between two screen locations
MOB OFF	to clear a MOB from the screen
MEM	to move a character from ROM to RAM
DESIGN	allocate memory for characters defined by user

Structured Programming:

IF THEN ELSE	If condition THEN true: ELSE false
REPEAT UNTIL	REPEAT loop UNTIL condition is met
RCOMP	to re-execute latest IF THEN ELSE test
LOOP EXIT	LOOP program loop EXIT IF condition true END LOOP

Program Procedures: to call subroutines by name rather than number

PROC	to label program subroutine
END PROC	end of a procedure (subroutine)
CALL	call procedure name, to continue program execution from a specified line of code
EXEC	to call a program routine and return to the line following the call when the procedure has been completed
LOCAL	to assign variables to specific program routine
GLOBAL	to restore original values to local variables
ON ERROR GOTO	line number traps program errors
NO ERROR	disables ON ERROR GOTO command
OUT	to re-enable '64' error handling routines

Music Commands:

VOL	volume level
WAVE	to set music voice type, synchronization and ring modulation

ENVELOPE	to define shape of sound played, attack, decay sustain and release
MUSIC	to compose music and save notes
PLAY	to play the music

Read Functions:

PENX	x coordinate of light pen
PENY	y cordinate of light pen
POT	returns resistance of paddle 0-255
JOY	test direction of joystick

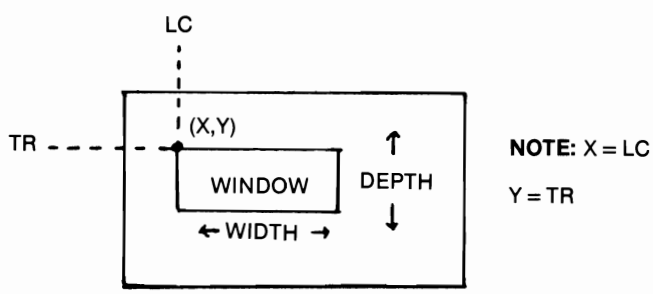
Error messages:

SIMONS BASIC has nine error messages to point out specific mistakes.

TABLE 2:
SIMONS BASIC Graphic Commands in High Resolution and Low Resolution Modes.

SIMONS BASIC

Syntax of Text Commands



Scroll A Window

UPB TR, LC, Width, Depth
Similarly for UPW, LEFTB, LEFTW, RIGHTB, RIGHTW, DOWNB, DOWNW

Note: The first two parameters are the coordinates of the top left corner of the window in reversed order.

Reverse A Window
INV TR, LC, Width, Depth

Fill A Window With Colour
FCOL TR, LC, Width, Depth, Colour

Fill A Window With A Character
FCHR TR, LC, Width, Depth, Character

Fill A Window With A Character In A Specific Colour
FILL TR, LC, Width, Depth, Character, Colour

Move A Window
MOVE TR, LC, Width, Depth, Destination Row, Destination Column

Flashing
FLASH Colour, Speed (OFF turns flashing off)

BFLASH Speed, colour1, Colour2 (BFLASH 0 turns flashing off)

Print At A Specific Spot
PRINT AT(x,y)"text" etc.

Syntax of Hires commands

Plot Types

HIRES MODE: 0 = clear dot
1 = plot dot
2 = reverse dot

MULTI-COLOUR MODE: 0 = clear dot
1 = plot dot(col.1)
2 = plot dot (col. 2)
3 = plot dot (col. 3)
4 = inverse dot col.

Turn On High Resolution Graphics
HIRES Plotting Colour, Background colour

Change to Multi-Colour Mode
MULTI Colour1, Colour2, Colour3

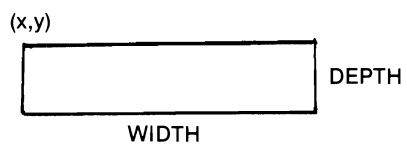
Change colour Registers
LOW COL Colour 1, Colour2, Colour3

Note: — In HIRES mode, colour 2 should be the same as the background colour, otherwise the entire 8x8 block in which plotting takes place gets changed to colour2. (This can sometimes be useful, e.g. drawing a solid 'thick' border).

— In MULTI-COLOUR mode, colour1 corresponds to plot type 1, colour2 corresponds to plot type 2, and colour3 corresponds to plot type 3.

Restore Original Plotting colours
HI COL

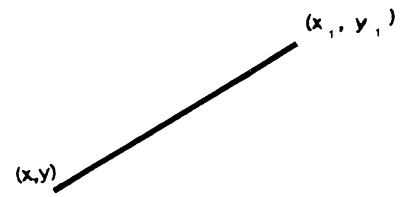
Draw A Rectangle
REC x,y, Width, Depth, Plot Type



Plot A Single Point
PLOT x,y, Plot Type

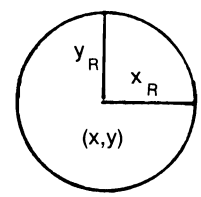
Test to See If A Specific Pixel Is On
TEST(x,y) [0 = pixel off: 1 = pixel on]

Draw A Line
LINE x, y, x_1, y_1, Plot Type



Draw A Circle
CIRCLE x, y, x_R, y_R, Plot Type

Note: For a true circle x_R should equal $1.3xy_R$



Draw An Arc
ARC x,y, Starting Angle, ending Angle, Increment, x_R, y_R, Plot Type

Note: Angles are measured clockwise with 0 degrees being straight up.

Paint A Region

PAINT x,y, Plot Type

Draw A Block
BLOCK x,y,x_1, y_1, Plot Type

Draw A Shape
DRAW 'shape string',x, y, Plot Type

ROT Rotation Number, Size (Rotation Number — rotates in steps of 45 degrees.)

Print text On Hires Screen

TEXT x, y, " control character character string", Plot Type, Size, Increment

CTRL A = uppercase/graphics

CTRL B = upper/lower case

Dr. Halfon is a research scientist with the Federal Government. He has been a member of Toronto Pet Users Group since November 1982, and bought a C64 in March 1983. In his work, he uses mainframe computers to develop simulation models describing the fate of toxic substances in the aquatic environment, Lake Ontario and the Niagara River. He has used his C64 to teach his wife and children about computers. He is a heavy user of the public domain educational programs provided by Commodore.

What Really is CP/M?

By Steve Rimmer, Toronto, Ont. (Reprinted by permission from Computing Now!, April 1983)

A very long time ago, eons measured in machine cycles, a company down in California that wasn't much of anything at the time released a disk operating system written for the 8080 processor called CP/M. Nobody was particularly sure what CP/M was, or what it was good for, or why they needed it right then because, in fact, you had to own a middle Eastern sheikdom to even afford a disk drive at that particular juncture of history. However, Digital Research put ads for the thing in Byte and Kilobaud and obviously managed to sell enough of it to stick around.

As the price of disks, and computer hardware in general, came down, and more people got into doing practical things with their computers above and beyond making the lights on the front panel count up to sixteen in binary, there came to be a need for a way to intelligently deal with disks. Early disk users were constantly being forced to reinvent the wheel of file handling and error trapping, which was a drag, and, as such, people started to buy disk operating systems. It was at this time that CP/M began to be recognized for what it truly is; a slow, archaic, poorly written piece of software which just lucked out and happened to do exactly what a lot of people happened to want to do. The lads at Digital Research began to take in money.

There are two groups of people in the world as regards CP/M. First off, there are the enlightened few. Secondly, there are about four billion examples of the totally mystified. If you are within the latter contingent you might want to read on and see what all the furor is about.

CENTRAL CONTROL

A good way to explain exactly what CP/M is..somewhat...is by using a PET. A PET can't actually run CP/M, as it has entirely the wrong processor and is actually quite aside from the whole topic. However, ignore that. It happens that machine code programs which are written on one PET (or, in fact, any CBM computer) will run on any other type of PET due to the existence of a ROM architecture called a Kernel. This is also referred to as a "jumbo jump table".

A jumbo jump table is a minor waste of memory which lives up at the very top of the CPU's address range, up in the FF's. All it is is a great string of JMP instructions to other routines in the ROM which could have been jumped to directly. However, as it is, these routines are at different locations in every different permutation of PET that's been released. The locations of their corresponding JMPs in the jumbo jump table, however, are all the same. Thus, for example, a programmer can write a program which jumps to location $FFD2 to print a character on an old ROM 8K PET and know that it will run just as well on a brand new VIC-20. All the useful I/O is handled through the jumbo jump table.

This is what CP/M does, on a grander scale. It consists of two bits, primarily. First, there's the CCP, which takes what it is given, both in terms of instructions from the human world and calls from machine language programs and deals with them. Secondly, there is the BIOS. The CCP is the same on everybody's system, and, whenever it wants to communicate with some peripheral, like the screen or a disk, it flags a character at the BIOS and says something like "This goes to the screen. Where the screen happens to be is your problem. I'm going for a nap."

The BIOS is system specific. It is written especially for whatever computer is using it. It knows which ports have which peripherals and how to deal with all the I/O. It is, in the normal course of events, the only thing which needs to know what system a program is running on. The actual software can go from an Osborne to a Multiflex to a Xerox to a Northstar and on and on without ever having to have a single byte changed.

Highly useful stuff, this, and it's all the theoretical headbending we're going to get into. For, you see, CP/M does a great deal more. It's not just a peripheral handling routine. When you buy a CP/M package you get between one and two dozen useful little programs, called utilities, and, as the documentation on them isn't all that explicit as to exactly what they're for, it may be worth a dig to find out just what all the file names do when you let 'em rip.

COMmunism

When you turn on your computer and load up the CP/M disk, the machine will start itself up, called a "cold boot", print a copyright notice and give you an **A**, which is called a prompt. It means that you are "logged on" to disk drive A. This, in turn, means that anything you do will happen to the files on disk drive A unless told otherwise. A bit of a waste if you've only got one drive on your system, I suppose...

If you type a B and a carriage return you'll be logged onto disk drive B. A letter with a colon after it always refers to a disk drive.

If you change the disk in the drive you are logged onto, you will have to get the attention of the computer again and tell it that you've upset its precise little environment by causing it to execute a "warm boot". A warm boot is what happens whenever the machine wants to return to just being logged onto a disk, and looking at a prompt, after running a program. Since you haven't run a program you'll have to do it by hand by typing a CTRL C. This will give you a new prompt on a new line.

A CTRL C is just a way of making the machine ignore the fact that what it's looking at isn't exactly what it was expecting, which is what it will see if you've swapped disks unexpectedly.

There are a few other CTRL codes that CP/M will recognize. CTRL H is the backspace, and is probably generated without your being aware of it by the DEL key of your computer. CTRL X will wipe out a whole line of text without backspacing to the prompt by hand. CTRL R will retype a line, which isn't very useful on a screen, but was good when computers used teletypes. As was mentioned previously, CP/M has its archaic bits.

CTRL P echoes everything on the console out to the printer. Hitting it once toggles it on, and hitting it a second time shuts it down again. CTRL S stops whatever is happening on the screen until the CCP receives a second CTRL S... essentially a "HOLD" key.

THE BUILT-INS

CP/M can do five things all by itself, none of which are much good all by themselves, but they do take on enormous proportions later on. Specifically, it can execute the built in commands ERA, SAVE, REN, DIR and TYPE. These are **erase** a file from the disk, **save** some pages of memory as a file on the disk, **rename** a file on the disk, put up a **directory** of the files on the disk and **type** a file from the disk onto the console...which is what CP/M likes to call the screen.

First off, it is useful to note that when you put a file on the disk with CP/M...a file is anything that's stored, such as a program, a text file and so on...the system stores the file proper somewhere, but, before doing this it puts the name of the file and a pointer to where it will be on the disk into what is called the directory track. This means, for example, that when you tell the CCP to get you a particular file, it doesn't have to go through the whole disk...which would take many minutes...but just to scan through the directory and get the location of what you're after. This also means that erasing a file just involves destroying its name and its pointer on the directory track. Later disk writings will overwrite its actual contents as the space it occupied before erasing won't be protected by pointers.

DIR produces a directory simply by printing up the directory track in an attractive format.

There's an interesting thing about DIR, though. It can use wild cards. Wild cards are called "ambiguous specifiers".The wild card symbol is an asterisk. Whenever you use a wild card you are telling the CCP "this part of whatever I'm talking about can be anything that will fit." File names in CP/M consist of eight letter names left of the period, a period (you might have guessed that) and a three letter extension. The extensions are meaningful; we'll get to that. If, for example, you wanted to see what files were on the disk which had the extension .COM, you could use a wild card with DIR. DIR* .COM refers to all .COM files. If you wanted to know what files began with the letter Q, you could say DIR Q*.*.

Wild cards work with DIR and ERA. If you ask ERA to use *.*, which means all files with names (all files), it will ask you if you really want to go ahead and kill off everything on the disk.

The built in commands, as is usually the case with all CP/M programs which relate to the disks, can also use disk specifiers. For example, if you wanted to know what files were on disk B you could log onto it and then call for DIR, or you could say DIR B:, which would look at B while still on A. You can also say B:DIR, which is not quite the same thing, as it is saying "log onto disk B for a second and do a DIR".

All the other commands associated with CP/M are called **transients**, as they get loaded into RAM, executed and then trashed. A transient is called a COM file. The CCP is set up so that the only kind of file it will recognize as a runnable program and thereupon try to execute is one with the extension COM.

A COM file can be invoked...a fancy way of saying "run"...by typing its name. PIP COM is a runnable COM file. To do a PIP, you'd just type PIP and a carriage return. Now, as to why anyone would want to PIP...

PIP AND OTHER WONDERS

When you get your CP/M disk, it will have a number of programs on it, as we've said, and none will immediately make sense because they're not really supposed to. Exactly what you get will vary with the supplier of your CP/M package. CP/M suppliers are software houses which, essentially, take the generic CP/M package, write BIOS's to suit specific popular computers and then package the whole works up in a reasonably understandable fashion. Among these software houses are Lifeboat, National Multiplex/Pegasus, Pickles and Trout and Magnolia. Most of these suppliers also add their own utilities to the pot, which will confuse things.

The most useful program on the disk is called PIP, which stands for Peripheral Interchange Program. Its primary use is in copying disk files, but it has a large number of options available on it. We won't look at them all, as some are really obscure, but it will be useful to understand exactly what PIP can accomplish.

PIP is set up to provide communication between any peripheral on a system, and CP/M treats virtually all I/O as peripherals. This includes the disk drives, the printer, the screen and keyboard and any ports. It is an interesting bit of prehistory, actually, to see what CP/M calls its peripherals...excluding the drives there are CON, for console, LST, for the list device, PUN, for the card punch and RDR for the card reader! What, no card reader on your system? It doesn't matter...nobody else has one either. These virtual devices are just references, and can be assigned to any ports you like.

The CON is usually assigned to the keyboard and screen. The LST is the printer. The PUN and RDR become the serial port and we're out of the vacuum tube age.

PIP can, first off, be used just to communicate between disk drives, which is what it defaults to doing if it isn't told to work with one of its assigned peripherals. To move a file from one drive to another, one can say PIP A:=B:FILENAME. One can also change the name of the file in the process by specifying a destination file name for PIP to load the data into, i.e., PIP A:WOMBAT. DOC=B:PENGUIN DOC. If both disks involved happen to be the same, you'll copy the file on the same disk.

If there happens to be a file called WOMBAT DOC on disk A prior to PIPing, most versions of PIP will cheerfully over-write it. Some tell you about it and inquire after your feelings on the matter.

In the same way, PIP can be used to communicate with its peripherals. For example, you could say PIP PUN:=A:WOMBAT DOC to send a file out to the serial port. In this way, files can be PIPed between two machines which don't have other, fancier communication software.

PIP has a number of options which can be selected by toggles. Toggles are enclosed in square brackets after the stuff we've just been looking at and are valid for the one command line only. There are quite a number of these things, but only a few are immediately useful. To wit, these are V,E,F and Z.

All PIPing should be done with V toggle in use, as it causes PIP to verify what it copies. Thus, moving files around should be done as PIP A:=B:WOMBAT DOC[V] to insure that what you send is what you wind up with. The E toggle causes whatever is being PIPed to be echoed up to the CON device, i.e., the screen. The Z and F toggles are used together, and have one practical function of stripping off the high order bytes in text files created by some word processors, such as Wordstar.

There is actually a heap of other things that can be done with PIP, but these functions tend to be more specialized, and, as such, can be dug out of the Digital Research manual if and when they're required.

Another useful disk utility is STAT, which is very much less complicated than is PIP. If you type STAT, you will find out how much free space is on the disk in question.

ASM is a Z-80 assembler. it takes a text file of Z-80 machine code mnemonics and does the first pass of an assembly which will eventually produce a COM file. These text files are identified by the extension ASM. you may have DUMBTERM.ASM on your disk. If you were to type ASM DUMBTERM.ASM, you would be on the way to getting DUMBTERM.COM...which is probably a bit pointless, as this file is usually already provided. However, it's the idea that counts.

ASM also checks for compilation errors.

It is beyond the scope of this article to get into writing Z-80 ASM code. Suffice it to say that, once you have done so, or obtained some from some other source (infinitely easier), you can ASM it and, if it is error-free, you will have, along with your ASM file, several other files with the file

name of your original ASM file but the extensions PRN and HEX. PRN is a second text file which has the original ASM file's statements plus the resulting object code plus any ensuing error messages. HEX is just the object code in textual form.

The HEX file contains a bunch of hexadecimal numbers which must be converted into actual bytes. This is done by a second utility called "LOAD". If you type LOAD WOMBAT you will get a file called WOMBAT.COM, which can be executed. The PRN and HEX files can be erased afterwards.

DDT is another assembly language tool which, once again, is too heavy to cover in this article. Basically, it is a very complex run time environment simulator which permits the programmer (yourself) to execute programs in a controlled setting so you can watch them and keep them paranoid. It has a dis-assembler for looking at little bits of object code and a simple interpretive assembler for doing "patches"..quick fixes on your program.

SYSGEN is the system generator. It is primarily useful when you are making up new disks. It can copy the system from an existing disk onto a new one. Depending upon whose SYSGEN you use, you may be able to specify a number of the parameters involved in this transfer. SYSGEN can also take a system out of memory and install it on a disk, which permits one to modify the CCP and then install it as a working system on a new disk.

FORMAT sets up the data storage format for a new disk. In most cases, you can specify single, double or extended density formatting. There is an obvious trade-off here; high density means more data on the disk, but it also means that each track occupies less area, which means that glitches that arise in the magnetic surface are more likely to cause later problems. Extended density disks will tend to have shorter useful lives for this reason, and very cheap disks will often not be much good formatted at high densities. Disk errors will show up as BDOS ERROR, WRITE ERROR, READ ERROR, and so on.

DUMP is a program which will take any sort of file and display it on the screen. On the left-hand side of the display will be the file in object form, a bunch of hex numbers. In the more useful versions of DUMP, the right-hand side of the screen will have ASCII characters for all the printable hex values which permits one to get oriented amidst the stream of data flowing by. DUMP is executed by typing DUMP WOMBAT.COM, or whatever.

FURTHER ADVENTURES

Knowing how CP/M works is only half the battle. You can actually get that, for the most part, from the Digital Research books if you are prepared to dig a little. Well, a lot then. However, there are a number of conventions which have grown up around CP/M in the years since it was first handed down from on high, and these are very useful to keep in mind.

First off, it will be noted that one Digital Research transient which we haven't looked at is the one called ED. ED is described as a "powerful contextual editor". ED is too archaic even to contemplate using, and is best ignored if you have something better to hand.

Wordstar is coming to be pretty well universal on CP/M systems. It's expensive but worth it, as it does double duty as a word processor and program editor. The D option does letters and text, providing justification and generally messing things up with control characters. N is for non-documents, such as ASM files.

You can't Wordstar with a COM file. You also can't TYPE one.

Files come in all types. Wordstar consists of WS.COM, which is what you type to get it going, plus two or three OVL files, depending upon which version you have. These contain the overlays, which include the menus and certain optional routines which get called in upon command. Wordstar also creates files, these called BAK files, which contain the version of the document being edited prior to editing.

BAS files are programs created by MBASIC, the CP/M version of Microsoft disk basic. MAC files are ASM files for a different assembler program, called MAC. MAC itself produces a different type of file called SYM, for symbols. MAC'd files are still LOADed in the usual way.

Then there are C files, for files to be used with a C compiler, usually BDSC. DOC files refer to on-disk documentation. If, for example, you get a file called PROGRAM.COM and another called PROGRAM.DOC, the COM file is to be run and the DOC file is to be TYPEd to see what's going to happen. Tarry not over a program called DOC.COM, which is a utility.

There are also OBJ files, for object code. These are just COM files that won't run because the CCP wants to see the word COM. These can be renamed into COM files and executed.

$$$ files are encountered only when something has gone wrong. For example, if you try to PIP a file onto a disk which has a bad sector or insufficient room to accommodate it, you will get a file with the right name but a $$$ extension, indicating that it's a disaster. $$$ files frequently occupy no space at all on the disk, being there only as indications that something is amiss.

Lastly, there are QQQ files, or squeezed files. If a normal file with an extension is squeezed, the middle letter of the extension will be changed to Q. If it had no extension, all three letters become Q. However, it's interesting to note that a squeezed file retains the name of the original unsqueezed file at the beginning of it, and even if the squeezed version is renamed prior to unsqueezing, the straight file will come out with its original name and extension.

Squeezed files are files that have been specially compacted to take up the minimum amount of room.

IN DISKguise

This is certainly not all there is to CP/M; it's a vast and complex operating system. However, these have been some of the more useful conventions and general bits about the system which should make getting into it a little easier. Many disk operating systems have tried to improve on this aging beast, but none has even begun to approach its widespread use. There are more sophisticated programs available for CP/M than for practically any other small operating system, and, while it's far from optimum in many respects, its very lack of specialization has made it suitable for a huge number of applications.

You just have to ignore the references to card punches.

CP/M Now A Reality With Commodore

By Tony Ning & Rick Denda, Toronto, Ont.

Good news for all you Commodore enthusiasts, CP/M is now available on 4000/8000 series machines via the Madison Z-RAM board. In case you are wondering why there is so much interest in CP/M, quite simply, it is currently one of the most popular Operating Systems available for micro-computers. CP/M will open doors to a range of good software that easily could multiply the number of applications available for your Commodore systems ten-fold. Furthermore, since Commodore recently announced CP/M compatibility options for the C-64 and most of their next generation computers, it would be timely to describe CP/M in some detail.

WHAT IS CP/M?

CP/M stands for Control Program for Microprocessors, a fancy term to describe an Operating System. At the lowest level, all operating systems perform the same tasks: get data from the keyboard, print information, and handle disk activities.

One major problem in running software on various computers, was that many had unique Operating Systems, since initially virtually all micros were developed independently. Thus, the user was generally limited to software created for a particular machine. Digital Research recognized the shortcomings and developed CP/M in an attempt to standardize an operating system for small computers. Since then, CP/M has blossomed into the most widely accepted and used operating system in the world for micros.

CP/M is essentially the same on all machines although there may be small variations e.g. version 2.2,3.0, etc.. It is likely that programs written under CP/M can run on machines which support it. That's quite incredible. The result, more software has been written in CP/M than any other Operating System in the micro-computer industry today.

CP/M ON COMMODORE

CP/M usually comes in the form of programs that reside on disk and are loaded into a specific memory location in the computer. In the case of Commodore computers, there are two hardware requirements that are not present for CP/M. Since CP/M was originally designed around the 8080 microprocessor and Commodore equipment uses either a 6502, 6510, 6809, etc.., the instruction sets are not compatible, thus an 8080/Z80 microprocessor must be added to the Commodore computer. The second requirement is that

the system must have at least 48K of user memory or RAM. For the 4032, 8032, and SuperPET, CP/M can be incorporated by using an add-on CP/M board. The most popular board is the Madison Z-RAM board which is available from most Commodore dealers or you may contact the Canadian distributer, Computer Workshops Ltd, 465 King St. E. Unit 9, Toronto, Ont. M5A 1L6, phone (416)366-6192. The Madison board consists of a Z-80 microprocessor (8080 compatible), a 6502 processor and 64K of additional RAM. When used in conjunction with a 4032, 8032 or SuperPET, a total of 96K usable memory is available to support programs requiring 96K RAM like VisiCalc 96, WordPro 5 Plus, and Silicon Office. The Madison Z-RAM board sends a standard RS-232 signal through the user port and will support RS-232 printers,modems, and other RS-232 peripherals.

Please note if you are planning to install the Z-Ram board primarily for the use of CP/M, you should have 80-column screen since most CP/M-based programs require an 80-column output. If you have 4000 series PET, I suggest you contact your local Commodore dealer who will be able to install an 80-column upgrade provided you have a 12" screen version. The MADISON Z-Ram board can be installed without special tools or great expertise and comes complete with detailed installation instructions as well as a comprehensive manual on CP/M . The Z-Ram board mounts inside your computer directly under your CRT(monitor), in fact it utilizes its mounting screws.

A decision as to whether or not to add CP/M to your Commodore computer should probably be based on the following criteria:

1. Are you now, or in the near future, planning to run programs under CP/M?

2. Do you require 96K of RAM and don't mind paying an additional $245.00 to obtain CP/M compatibility (difference between the COMMODORE 64K memory expansion and Z-RAM board cost)?
3. Do you plan to write programs that you may wish to run on the new generation Commodore computers which will support CP/M?

4. Do you wish to have additional language capability such as: COBOL, FORTRAN, PASCAL, PL/1, APL, C-BASIC?

If you answered yes to any or all of the above, you should probably consider buying the Z-RAM board.

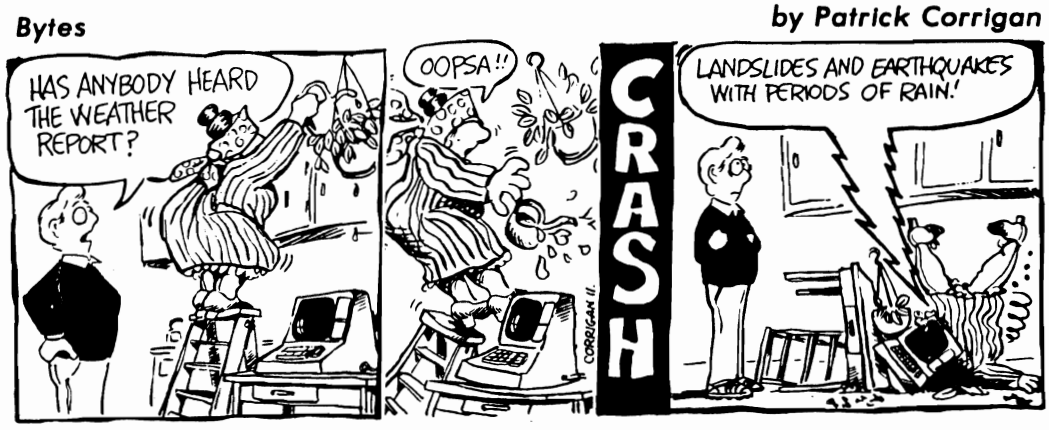
As far as CP/M for the Vic and the C-64 is concerned, there will be a CP/M card available soon, and I will keep you posted on further developments; or you may call me at (416)366-6192.

In the seven years that CP/M has been around, thousands of programs have been created that run under it, written by over 100 companies. Applications range from languages, development utilities like assemblers, to application programs such as ACCOUNTING, DATA-BASES, FINANCIAL PLANNING, and WORD PROCESSING. There is also a range of vertical application packages available under CP/M which currently may not be available in Commodore compatible software.

P.S.: GOOD NEWS

CP/M plug-in module, also providing 80 columns of display, is now available for the Commodore 64.

Reference for the CP/M board and CP/M for C-64 includes A Word Processor at $399 (Canadian). Both are available from Marketron, 465 King St. E., Unit #9, Toronto, Ont. M5A 1L6, or by contacting your local dealer.



CP/M On The C64

By Fred Wallace, Windsor, Ont.

CP/M IS A REGISTERED TRADEMARK OF DIGITAL RESEARCH

Few accessories from Commodore have been as eagerly anticipated as the CP/M package. Like so many programmers, I use CP/M daily and could never really understand why it was so long in being made generally available to us Commodore types. After all, I first saw one of Commodore's demo machines running it almost a year ago. And doesn't each and every C-64 carton promise its availability? Nevertheless, it's out now, and with visions of being able to run all my "big machine" FORTRAN programs in color, I eagerly rushed out for an early copy.

The package supplied by Commodore consists of three major parts: a somewhat oversize cartridge which plugs into the expansion slot at the rear of the C-64, a diskette in 4040/1541 format containing the operating system and skeleton programs, and a manual. Installation was no problem as the instructions given were complete and correct: you just plug the cartridge in and then use the familiar

```
LOAD"*",8
RUN
```

sequence to start the program.

In my case, however, this marked a substantial gap in my testing. I could not get the system to go, and investigation proved that the problem was caused by several bad sectors on the diskette. They would not read even on another 1541. The dealer exchanged it for me, and then I found that while this one would run, a couple of the important programs supplied on the diskette would not. Back again! The third copy eventually proved to be readable, but despite claims from the dealer involved that he was not having a problem with them, I consider two bad out of three to be a pretty rotten track record. In any case, protect yourself by making sure you purchase from a dealership which is going to be able to help in case something like this happens to you.

WHAT'S CP/M?

But just what is CP/M and why would you want it on your C-64? CP/M stands for Control Program for Microprocessors and it is an operating system which was introduced a number of years ago by Digital Research. It was among the first serious attempts to create a "machine independent" programming environment for small machines: that is, a given program could be run, theoretically, on any micro using an Intel-type processor without modification. The operating system would take care of the differences between the machines and permit the program to do its job without having to worry about details such as whether the peripherals were memory or I/O-mapped.

The attempt was successful to the point that there are now thousands of programs which are "CP/M compatible", and that is of course the attraction. After all, nobody really wants to re-invent the wheel if all that is needed is transportation. The sole catch in the procedure is that you are not processor-independent: you can only run CP/M programs on a processor which is or has a compatible instruction set to the Intel 8080 or 8085, or the Zilog Z-80.

That's why we need the cartridge: it contains a Z-80 chip with support circuitry to enable it to share the memory, peripherals, and busses of the 6510 contained in the C-64.

HOW IT WORKS

Although the two processors are both "on the buss" (as long as the cartridge is plugged in), the circuit is arranged so that only one of them can actually be alive at any one time. The C-64 powers up in the normal "BASIC" mode as usual, and in fact the cartridge could be left plugged in all the time so long as a program or game does not accidentally trip the location which transfers control over to the Z-80. Code for the 6510 and for the Z-80 can be freely mixed in memory so long as one doesn't try to run the other's code.

The LOAD/RUN procedure brings into memory a small machine-language program whose sole purpose is to bring in from the diskette the rest of the code - some of it for the 6510 and some (most of it) for the Z-80. When the procedure is complete, things have been arranged so that the Z-80 is now the "main" processor and the 6510 is used to handle I/O procedures through the KERNAL routines. From this point on the C-64 acts as if it were a terminal connected to a Z-80-based micro system.

UP AND RUNNING

For those already familiar with CP/M, the running system is by and large a faithful reproduction of others you have used, with some notable exceptions. The most immediately noticeable is the limitation imposed by the 40-column screen. Most CP/M programs assume 80 columns and may need modification. In addition, DO NOT automatically assume that you can simply run out and purchase an 80-column adapter or program for use with CP/M: most will not work since they rely upon having access to memory areas which are now the domain of the Z-80 chip. Ask the supplier if he can certify compatibility.

The second major difference is speed. CP/M makes heavy use of the disk, especially during such activities as the program development cycle, and the 1541 will be a real disappointment to most people here. Somehow it doesn't seem that slow when it's running BASIC stuff, but when you have to wait nearly 25 seconds for a simple WARM BOOT (a kind of master reset required at the end of just about every CP/M program's execution), it does tend to wear a little thin. Fortunately, the package has been designed to work with the IEEE adapter and associated fast disks, and I have a feeling any serious CP/M'er is going to want one of those really quick.

But the real crunch comes when you want to use commercial software. After all, we now have our machine running a "universal" operating system - let's go get some of those fancy programs that are for sale by the hundreds in the magazines and start running them! Great idea, except the Commodore disk format is different from other 5 1/4" formats, and is not supported by other manufacturers. Of course, this is a temporary barrier, but is a real factor when the inevitable question "What do I do with it now?" is asked. Until Commodore and third-party vendors catch up, only programs entered into the machine by hand are available.

PLUSES AND MINUSES

Devoted Commodore users will miss the screen-edit functions for program creation and modification. ED, the standard CP/M editor which is supplied on the system diskette, is line-oriented and seems archaic after using the full-screen BASIC editing. A full-feature screen edit will doubtless be among the first accessory programs to be made available.

My compliments to the Commodore people for taking such thought with the FUNCTION keys.

Under CP/M, each key can be "firm-programmed" to stuff a text string into the keyboard buffer, even including the carriage return. They come already set up with such universally-useful strings as "STAT *.* ‹CR›" , which at least for me save quite a bit of typing.

The manual supplied with the package is unusually high in detail content: an acknowledgement I suppose of the fact that the majority of people buying it are already quite computer-literate and will want to immediately start poking around in its innards. There are listings of some of the key software, which should enable customizations to be made. But the edition I received contained a polite note apologizing for the fact that some 14 pages (including the module schematic) had been selectively removed by Commodore.

I was somewhat disheartened to discover that not all of the C-64 kilobytes of memory available can be utilized by CP/M — this is mostly due to the memory sharing described earlier: some memory must be reserved for functions which the 6510 performs. In fact, though, the 48K which is available (44K if you have the IEEE adapter) is sufficient for all but some of the more complex business and scientific software.

The diskette format has been cleverly worked out so that it is not possible, without some really deliberate work, to accidentally overwrite any areas while running the C-64 in its BASIC mode: the diskette allocation map has been prewritten to make it appear to be full at all times. The minus comes when you discover that the normal 170K which is available to BASIC has shrunken to 136K under CP/M. Again, some space was required to store operating-system-related material.

My greatest disappointment, however, was in discovering that one of the devices NOT supported by CP/M at the time of this writing was the serial line (RS-232 port). I have another machine sitting next to the C-64 just loaded with CP/M software, and all I need to transfer it is a serial link. I also make use of a number of bulletin boards, and all I need to access them under CP/M is a serial link. This hole in the machine is primarily a hardware limitation and has caused a remarkable flurry of activity on the data services (notably Compuserve) so I fully expect there to be a solution very quickly from one of the midnight-oil crew. In the meantime I'm working on my own solution short of typing in megalines of code.

IN CONCLUSION

If you've always wondered what CP/M is, and already have a C-64 and disk drive, this package plus one or two of the excellent texts on the subject is a really economical way to learn (I've listed a couple of my own favorite books below). Similarly, the combination is useful for development of small Z-80 programs, especially if you already have a PROM burner on the unit. As a serious contender in the market for experienced CP/M users, though, it will not be terribly useful until a means of intermachine transfer of programs is made available, despite the attraction that color and sound may have as enhancements to CP/M programs.

TEXTS ON CP/M

1. Cortesi, David E. *Inside CP/M, A Guide for Users and Programmers*

2. Zaks, Rodnay *The CP/M Handbook with MP/M*

CP/M File Transfer for C64

By Wm. Kendall, Baltimore, MD.

This program is on The Best Programs Disk

Fellow purchasers of the Commodore 64 CP/M cartridge probably experienced the same frustration that I did when I unpacked the cartridge and disk, booted CP/M, and found that there were no programs to run except those of the CP/M system itself. Here is a seat-of-the-pants procedure I have found that makes it possible to download hex files from any CP/M computer with a modem. If you have the computers side by side you can just hook them together with an RS232 connector and do without the modem.

I wrote a simple BASIC program to poke the ASCII code for each character into memory as it comes from the RS232 buffer. The C64 is in normal 6510 operation but has the CP/M cartridge plugged into its port. After the file is all in the memory I put in the CP/M disk and load CP/M. Then I move the down-loaded file into the CP/M transient program area (TPA) with DDT (the CP/M machine language monitor), and save the file on the CP/M disk.

Now for the details! The equipment I use is an OSI Challenger with a D&N Z80 board. There are many CP/M modem programs which will upload files. Modem 7 and Modem 705 are public domain programs and Mite, Ascom, and Crosstalk are some of the commercial ones. Any will do the job.

Binary (COM) files under CP/M must be converted to HEX files before uploading from the CP/M computer. This is necessary because the C64 RS232 port only receives 7 bit code. There are many programs which do this conversion. The one I use is called COMHEX.COM. It reads a COM file and outputs a HEX file on the same disk. Example: If you had a COM file named TPUG.COM and COMHEX.COM on drive A, you would type COMHEX TPUG.COM to get your hex file. The output file would be named TPUG.HEX.

It's a lot easier to move a file in memory if you have a programming calculator such as the TI Programmer. It helps a lot in figuring the size of the file in memory and how many pages to save under CP/M. I think that's the last of the equipment except for pencil and paper.

Now you're ready to go. My program sets up the C64 terminal with mark parity, 300 baud, 1 stop bit, 1/2 duplex, and 3 line handshaking. Set up your CP/M terminal the same way. If you wish you can use full duplex at the CP/M terminal to watch the file go by. If you have trouble with the RS232 buffer filling up on the C64, remove PRINT A$ from line 140. I set the CP/M terminal to transmit a 'G' (hex 47) at the end of the file. This character is not used in hex files, and when line 45 finds a G, it signals the end of the file and closes the terminal. When the transmission is finished, load CP/M. When the A prompt comes on, type DDT. Your program starts at $1400 in memory ($2400 on the Commodore system is $1400 for CP/M). Check the beginning to make sure the first characters came through correctly and, if necessary, correct with the S command of DDT. I find that the first character is often changed. Don't ask me why; I just work here!

Next, find the end of your file and move it to the beginning of the TPA ($0100). You can find the size of the file with XDIR or STAT on the CP/M computer. If for example your file goes from $1400 to 2FFF, you would type M1400 2FFF 0100 to move the file to the beginning of the TPA. Now all you have to do is reboot (control-C), figure out how much memory to save (the CP/M manual tells how), and save your hex file. For the above exam-

ple you would type SAVE 28 TPUG.HEX. (All file names to be used by the CP/M LOAD utility must end in .HEX). Type LOAD TPUG.HEX and the output will be an executable COM file stored on your disk and ready to go!

One final complication. When a binary file is converted to hex the file becomes much larger. If your hex file is larger than 16K it will get messed up by BASIC garbage. You will have to split your original COM file in two before changing into HEX files. Save the two parts under two different names, for example, TPUG.COM could be split into TPUG1.COM and TPUG2.COM. Be sure to split at a place easy to find (words are the easiest) and take notes on what you did so you're sure of getting the whole file. After converting to HEX and downloading, the 2 files can be recombined with DDT by loading the second half, moving it up in memory, loading the first half, and then moving the second half down until the splice is perfect. This is not as hard as it sounds and if you goof it up you can always try again — your HEX files are already on your disk. The main problem that has plagued me is not saving enough pages for my hex file on the CP/M disk. Save an extra page to make sure. If you get an error message when using LOAD, you probably chopped off the end of the file or forgot to correct the first character. If you have a problem you can look at your HEX file by using DUMP TPUG.HEX or TYPE TPUG.HEX, or by changing the name to TPUG.COM and loading it with DDT. Then you can correct it or add characters to the end if necessary. Don't forget to change the name back! A machine language program to poke the file into memory would probably handle larger files; if one of the other TPUG members writes one I'd appreciate a copy! Somebody will eventually figure out a modem program for C64 CP/M, but it looks to me like we have a chicken and eggs situation right now. Until the CP/M cartridge is popular nobody's going to make any money selling a terminal program, and nobody in his right mind will buy the cartridge if he knows there's no software available. Present company excepted, of course!

C64 CP/M DOWNLOAD

```
100 GOTO210
110 GET#2,A$:IFA$ = ""THEN110
120 IFA$ = "G"THENPRINT"FILE ENDS AT ";AD:CLOSE2:END
130 T% = ASC(A$)
140 PRINTA$;:POKEAD,T%:AD = AD + 1
150 SR = ST:IFSR = 0THEN110
160 PRINT "ERROR"
170 IF SR AND 1 THEN PRINT"PARITY"
180 IFSRAND2THENPRINT"FRAME"
190 IFSRAND4THENPRINT"RECEIVER BUFFER FULL"
200 IFSRAND128THENPRINT"BREAK"
210 OPEN2,2,3,CHR$(38) + CHR$(176)
220 AD = 9217
230 GET#2,A$
240 PRINT"PRESS RETURN WHEN READY"
250 GETA$:IFA$ <> CHR$(13)THEN250
260 PRINT"(CLEAR)READY TO RECEIVE DATA":GOTO110
READY
```

C64 CP/M DOWNLOAD WITH REM STATEMENTS

```
90  REM *CP/M- C64 DOWNLOAD*
100 GOTO 210 : REM INITIALIZE
110 GET#2,A$:IFA$ = ""THEN 110: REM GETS A CHARACTER
FROM BUFFER—WON'T TAKE NULLS
120 IFA$ = "G" THEN PRINT "FILE ENDS AT
";AD:CLOSE":END: REM G IS EOF SIGNAL
130 T% = ASC(A$): REM GETS ASCII CODE
140 PRINTA$;:POKEAD,T%:AD = AD + 1: REM POKES
MEMORY AND INCREMENTS ADDRESS
150 SR = ST:IFSR = 0THEN110: REM ERROR CHECKING-IF OK
GOES BACK TO 110
160 PRINT "ERROR"
170 IF SR AND 1 THEN PRINT"PARITY"
180 IFSRAND2THENPRINT"FRAME"
190 IFSRAND4THENPRINT"RECEIVER BUFFER FULL"
200 IFSRAND128THENPRINT"BREAK"
210 OPEN2,2,3,CHR$(38) + CHR$(176): REM 3LINE, MARK
PARITY, 1 STOP BIT, 300 BAUD
220 AD = 9216: REM THIS IS WHERE PROGRAM STARTS IN
MEMORY-$1400 FOR CP/M
230 GET#2,A$: REM TURNS ON RS232
240 PRINT"PRESS RETURN WHEN READY"
250 GETA$:IFA$ <> CHR$(13)THEN250: REM WAITS FOR
RETURN
260 PRINT"(CLEAR)READY TO RECEIVE DATA":GOTO110
R E A D Y
```

One has to be a sailor of the I.C.'s to fathom the VIC-64. It's even named after a navy man, a COMMODORE. He's in charge of a whole fleet of CHIPS.

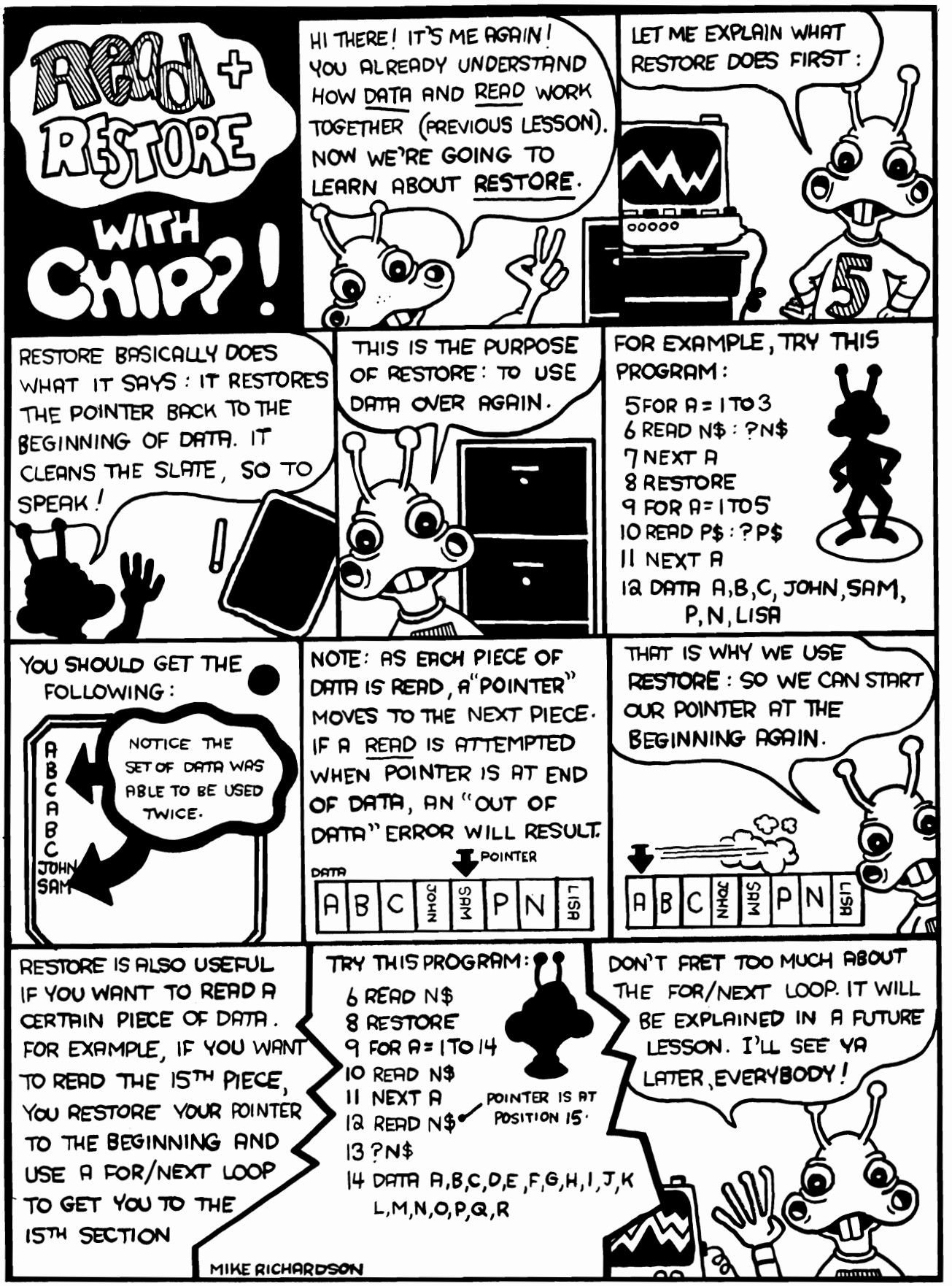
— the 6510 CHIP is sort of a c.p.U-Boat. Its manoeuvres are called SUB-routines.

— the 6566 Video Chip carries the fleet's colours. It's a SPRITE for sore eyes!

— watch out for waves around the SID CHIP. Listen to its beautiful sounds, but don't get lost in the high C's.

— follow a CURRENT back to PORT. Don't collide with any FLOATING numbers.

— Ylimaki


READ + RESTORE WITH CHIP?!
HI THERE! IT'S ME AGAIN! YOU ALREADY UNDERSTAND HOW DATA AND READ WORK TOGETHER (PREVIOUS LESSON). NOW WE'RE GOING TO LEARN ABOUT RESTORE.
LET ME EXPLAIN WHAT RESTORE DOES FIRST:
RESTORE BASICALLY DOES WHAT IT SAYS: IT RESTORES THE POINTER BACK TO THE BEGINNING OF DATA. IT CLEANS THE SLATE, SO TO SPEAK!
THIS IS THE PURPOSE OF RESTORE: TO USE DATA OVER AGAIN.
FOR EXAMPLE, TRY THIS PROGRAM:
5 FOR A=1 TO 3
6 READ N$:?N$
7 NEXT A
8 RESTORE
9 FOR A=1 TO 5
10 READ P$:?P$
11 NEXT A
12 DATA A,B,C,JOHN,SAM, P,N,LISA
YOU SHOULD GET THE FOLLOWING:
A B C A B C JOHN SAM
NOTICE THE SET OF DATA WAS ABLE TO BE USED TWICE.
NOTE: AS EACH PIECE OF DATA IS READ, A "POINTER" MOVES TO THE NEXT PIECE. IF A READ IS ATTEMPTED WHEN POINTER IS AT END OF DATA, AN "OUT OF DATA" ERROR WILL RESULT.
DATA
POINTER
A B C JOHN SAM P N LISA
THAT IS WHY WE USE RESTORE: SO WE CAN START OUR POINTER AT THE BEGINNING AGAIN.
A B C JOHN SAM P N LISA
RESTORE IS ALSO USEFUL IF YOU WANT TO READ A CERTAIN PIECE OF DATA. FOR EXAMPLE, IF YOU WANT TO READ THE 15TH PIECE, YOU RESTORE YOUR POINTER TO THE BEGINNING AND USE A FOR/NEXT LOOP TO GET YOU TO THE 15TH SECTION
TRY THIS PROGRAM:
6 READ N$
8 RESTORE
9 FOR A=1 TO 14
10 READ N$
11 NEXT A
12 READ N$
13 ?N$
14 DATA A,B,C,D,E,F,G,H,I,J,K L,M,N,O,P,Q,R
POINTER IS AT POSITION 15.
DON'T FRET TOO MUCH ABOUT THE FOR/NEXT LOOP. IT WILL BE EXPLAINED IN A FUTURE LESSON. I'LL SEE YA LATER, EVERYBODY!
MIKE RICHARDSON

General Hardware

PAGE

Radiation Hazard 128
Dr. Piasecki, Oakville, Ont.
If you use your computer with an old color television and you read only one article, then read this one.

C64 Link 130
Todd Hamilton, Highland, MI
The C64 and VIC do not come with an IEEE port. This device gives you access to both IEEE and other configurations, plus a number of advantages.

80 Columns for the VIC 131
Steve Garmon, Houston, Tex.
A review of the DATA-20 unit that gives you 80 columns on your VIC.

Radiation Hazard

By Dr. G. Piasecki, Oakville, Ont.

SUMMARY

The possible hazard of using an older color television set as a monitor is discussed. The group of people most likely to be exposed to this type of hazard are your readers. The reason for submitting this article is to try to inform persons at risk and I felt that this could be best done through a computer magazine.

ARTICLE: RADIATION HAZARD OF VIDEO SCREENS

As a physician I receive by subscription the *New England Medical Journal*. In the September 30, 1982 Issue there appeared in the Correspondence section a letter on the above subject.

It came from the Veterans Administration Medical Center, Washington, DC 20422. It was authored by David J. Nashel, M.D., Louis Y. Korman, M.D., and John O. Bowman, M.S.

As a physician, it is important for me to be aware of the possibility of disease arising from certain patient circumstances.

However, more important is the possibility of prevention of disease. With the appearance of the new computers (COLOR but without the color monitor), your readers, if they use old color television sets as monitors, would be the group at risk. To make this group aware of this risk is the main purpose of this article.

The following material and the References come from the authors mentioned above.

Although it is generally agreed that the video display terminal is not a major source of radiation for the user (1), field surveys of older color television sets (2-4) have indicated that 2.33 to 16.2% of receivers at some surface point exceed 0.5 milliroentgens (mR) per hour, which is the limit for emission set by the Food and Drug Administration, Bureau of Radiologic Health (5). From 1960 until January 15, 1970, when emission standards for color television sets were adopted, 25 million sets were produced. Since almost all these sets were manufactured in the late 1960s and the average life of a tube-type television is 11 years (Gerson R: personal communication), many of these receivers are still in operation.

Since radiation intensity is a function of distance from the emitting source, the spatial separation of viewer and display screen is extremely important. At average viewing distances (165 cm for children and 250 cm for adults), the estimated annual radiation dose from older color television sets appears to be within the accepted limits (2). However, users of microcomputers tend to position themselves closer to the display screen, thereby increasing their radiation exposure. To estimate the average annual radiation dose from an older color television set used as a display screen, the standard computational formula was used:

$$D = 1.13XTdF,$$

where D = estimated average annual dose (millirems per year), X = exposure rate at 5 cm from front face of the picture tube (milliroentgens per hour), T = annual viewing time (hours per year), d = distance factor, and F = depth dose factor. The value chosen for exposure rate (X) measured at the front face of the picture tube is 2.7 mR per hour. This was the average dose of radiation recorded from color television sets that exceeded the accepted emission standard in a survey done in metropolitan Washington, D.C.(2) Viewing time (T) is conservatively estimated at two hours per day, or 730 viewing hours per year. Using data from Wang et al.,(6) a distance factor (d) of 0.5 was obtained at a viewing distance of 40 cm; the depth dose factor (F) is 0.70 for the juvenile thyroid and 0.80 for the lens of the eye. Using these values, the estimated average radiation dose to the thyroid would be 779 mrem per year, and the dose to the lens of the eye would be 890 mrem per year.

These calculations suggested to the authors that youngsters using OLDER COLOR TELEVISIONS for display screens may be at risk for radiation exposure far in excess of the National Council for Radiation Protection and Measurement's recommendation of 100 mrem per year for persons under 18 years of age.(7) In order to decrease the posssibility of excessive radiation exposure, the authors suggested that only newer color television receivers (those manufactured after January 15, 1970) be used as display elements for computer function.

REFERENCES

1. Food and Drug Administration, Bureau of Radiologic Health. An evaluation of radiation emission from video display terminals. February, 1981. (Report 81-8153).

2. Neill Rh, Youmans HD, Wyatt JL. Estimates of potential doses to various organs from x-radiation emissions from color television tubes. Radiol Health Data Rep. 1971; 12:1-6

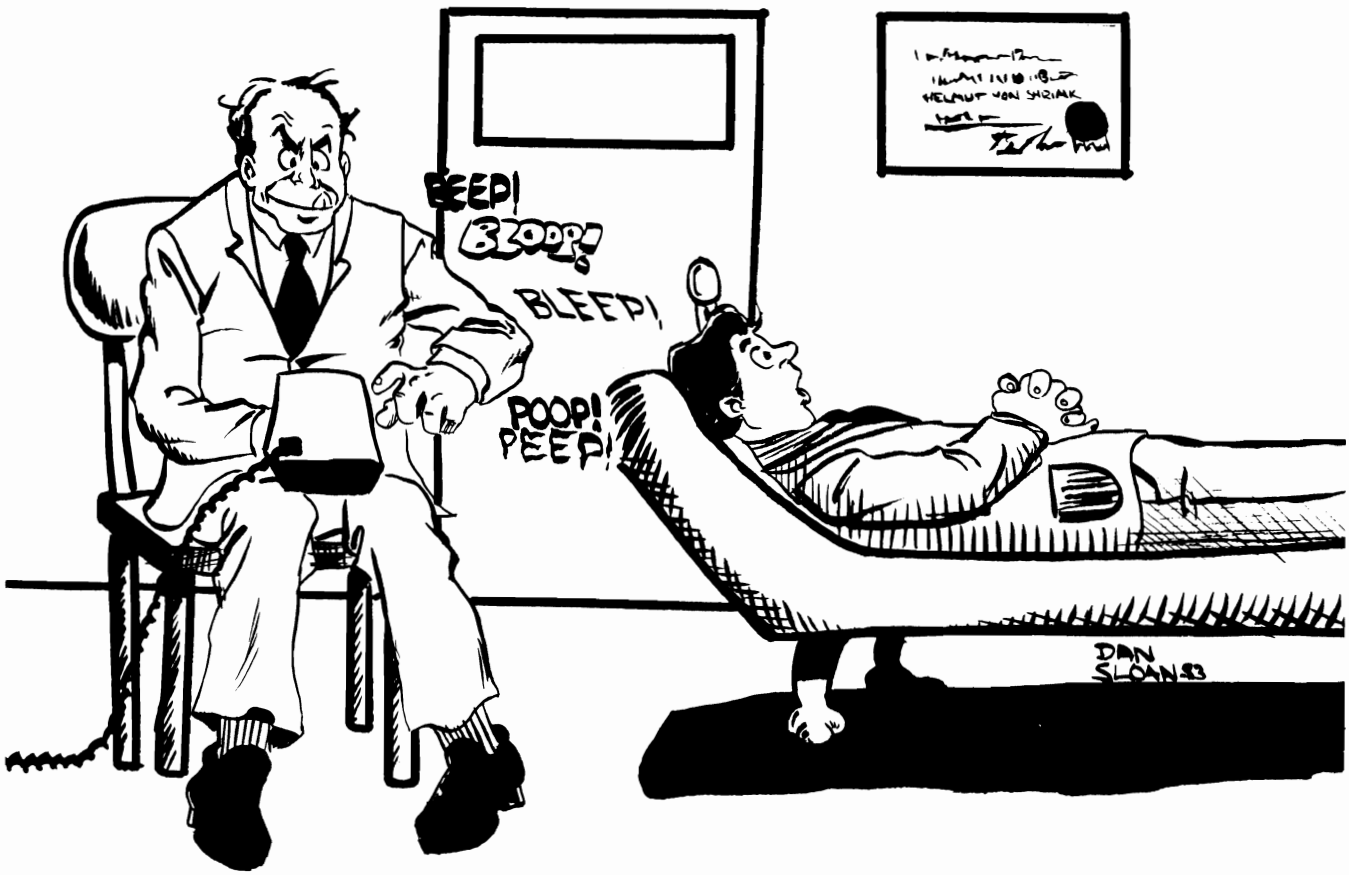
3. Radiologic Health Program, Commonwealth of Puerto Rico Department of Health. Results of a survey of x-radiation from color television receivers in the metropolitan area of San Juan, 1969-1970. Radiol Health Data Rep. 1971; 12:547-51.

4. Becker S. Results of a follow up radiation survey on color television sets, Suffolk County, New York. Radiol Health Data Rep. 1971; 12:457-8.

5. Public Health Service, Bureau of Radiologic Health. Regulations for the administration and enforcement of the Radiation Control for Health and Safety Act of 1968, OBD 71-1 (March 1971).

6. Wang SP, Savic S, Hersh H. Spectral and spatial distribution of x-rays from color television receivers. Technical papers, Conference on Detection and Measurement of x-radiation from Color Television Receivers, March 28-29, 1968. Washington D.C.: National Center for Radiological Health, 53-72.

7. National Council on Radiation Protection and Measurement. Radiation protection in educational institutions. July, 1966. (Report no. 32) 7-8.



DOC, PEOPLE TELL ME I'M CRAZY WHEN I SAY THIS PERSONAL COMPUTER TREND IS GETTING OUT OF HAND. WHAT DO YOU THINK?

C64 Link

By Todd Hamilton, Highland, MI.

The Commodore 64 Computer is not designed to communicate to IEEE-488 devices. Peripherals such as the 4040 disk drive and 4022 printer normally used on PETs will not connect to the C-64.

RTC of Canada markets the C-64 link to interface between the C-64 computer and either parallel, serial or IEEE devices.

After four months of use, my conclusion is that the C-64 Link is a very fine product. My hardware-test experience includes a C-64 Computer, C-64 Link, 1541 Disk Drive (serial), and a CBM 2022 Printer (1EEE). To date, no parallel devices have been used.

The C-Link switches modes with either immediate keyboard commands, "IEEE(CR)", or software commands, "POKE 820,3". The C-Link allows the user many of the BASIC 4.0 commands. However, only the BASIC 4.0 commands understood by your peripheral will work, ie. back-up will not work with the 1541 disk drive. The C-Link allows the user to call the on-board monitor, "MONITOR", at any time and to return "X" to BASIC. It also has "Modem" software, but mine is yet to be exercised.

After two and one-half months of operation, the C-Link failed to communicate to the IEEE printer, "Devices not ready". The link was mailed back to RTC for repair. The C-Link was returned 10 days later, repaired and ready to go. The service was excellent.

After a total four months of service, the C-Link met my objectives of being able to communicate with an IEEE printer and offered useful new features. However, there is room for improvement. The documentation that comes with the C-Link is only adequate. It lacks details in the BASIC 4.0 commands and I/O capabilities of the C-Link itself. It is not clear which BASIC 4.0 commands will work, and lacks detailed examples for 1541 users with one or two units. The information on software switching of the I/O devices is included in the manual, but little reference is made to it and NO examples are included. The monitor is a definite plus, but it is a very elementary monitor. More capability added to the monitor would make it a very powerful tool.

The C-Link comes with software to relocate its code to prevent interference with BASIC, etc. However, some machine language programs interfere with C-Link software anyway. The end result is many removals and reinstallations of the C-Link.

Enhancement of the C-Link could be provided with switch selectable options:

1. Code location (change memory blocks used)

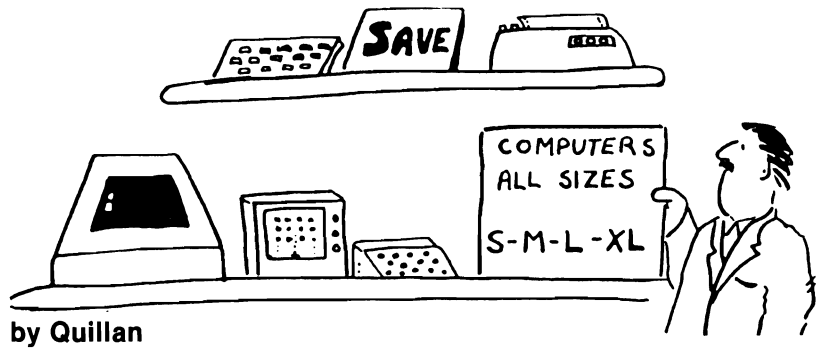
2. Primary device (presently power-on is IEEE. Serial 1541 requires pressing the C key on power-up)

3. Disable the C-Link without removing it

4. Reset the computer. This would allow moving the code, disabling the link, etc.

The operation of the link could also be improved with a device not ready response. Presently, if a command is sent to a IEEE device (the printer) and that device is off, the program locks up with the screen blank or appears to transmit data, but nothing happens. A gentle reminder "device not ready or present" and a return to the ready mode would be a help.

The C-64 is a fine product and if you are thinking of borrowing mine, you can forget it. I use it every day!



by Quillan

80 Columns for the VIC

By Steve Garmon, Houston, Tex. **(Reprint from CHUG)**

Because I write a lot of articles, I have been interested in word processors. There are quite a few available for the VIC but most are subject to VIC's major word processing limitation, its 22-column screen. There is no way you can visualize what your heavily formatted text will look like on a printer which prints the standard 80 columns when you are judging it against a 22-column screen display. This limitation led me to investigate 40/80-column adapters. I stumbled across one while shopping at *RAVE*, a popular electronics discount store located in Houston, Texas. They were offering *DATA-20*'s "Display Manager", a 40/80-column adapter for the VIC, at only $79.00. At that price, I couldn't pass it up. Admittedly, not all computer hacks would be interested in spending as much money on an adapter as they did for their computer but it's worth it if you are into word processing and instrument control and you want your applications to be greatly enhanced by the ability to display more than 22 characters per row across the screen.

The DATA-20 adapter really is a well made product. Standard television output offers an adequate display for 22 or 40 column modes; however, a video monitor is recommended for adequate resolution when using the 80 column mode. The 22 column display is still available even when the cartridge remains plugged in but it requires its own TV set or monitor.

The adapter is a plug-in cartridge which has a connector on the end of it exactly like VIC's video connector. To use the cartridge, you must plug the video cable into the cartridge rather than the VIC. The cartridge contains a 2K EPROM which in turn contains all of the necessary programming for the 40/80 column modes. It also has a 6845 video controller chip which does all of the hard work of formatting displays. An excellent feature of this product is the availability of the full Commodore character set (unlike some competing products). On power-up, the cartridge takes over and sets the display mode according to operator selection. To enter the 80-column mode you merely hold down the RUN/STOP key while turning on the computer. Otherwise, power-up defaults to the 40 column mode. Pressing function key F7 can also invoke the 80-column mode and pressing F5 will return you to the 40-column format.

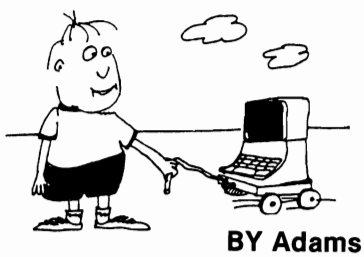
The cartridge uses memory in block 5. The operating system for the video display is located from $A000 to $AFFF and the video display is located from $B800 to $BFFF. 2K of video memory is provided so that there is no overhead requirement from your expansion memory. The DATA-20 cartridge makes provision for 8K of memory expansion to be located either at $2000 (block 1) or $6000 (block 3).

There are several differences which will have to be learned in using this cartridge. The standard VIC will allow 88 characters per program line but the Display Manager cartridge will allow only 80 in the 80-column mode and 40 in the 40-column mode. On the standard VIC the cursor is turned off while a program is running but, with the DATA-20 40/80-column adapter, the cursor can be toggled off or on.

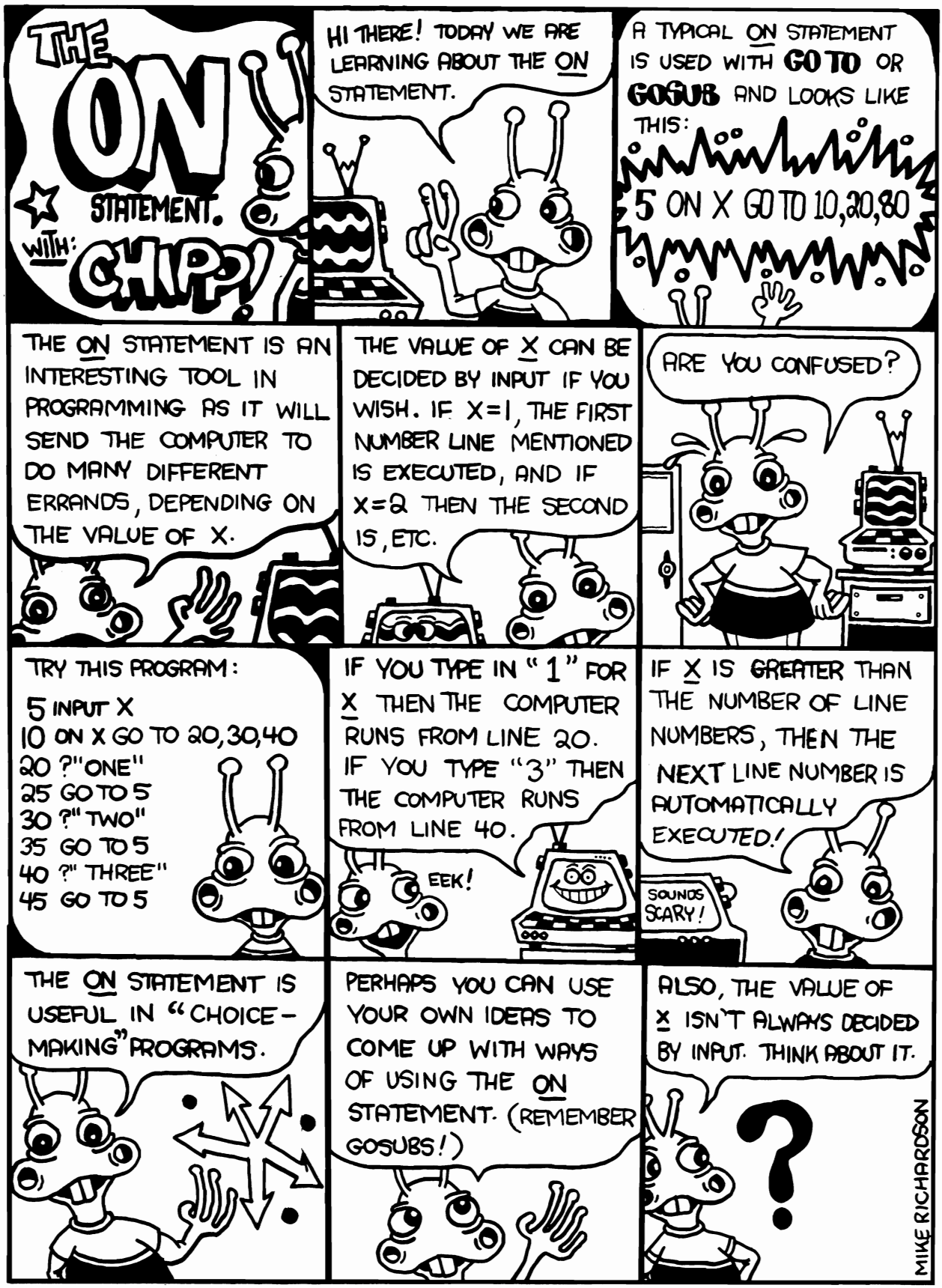
Here's a surprise: terminal emulator software is built into the cartridge and can be accessed by simply opening an RS-232 channel and then pressing F8 of the User Function Keys. The terminal emulator alone could justify the cost of the cartridge. Also included is a screen dump feature which allows dumping the contents of the screen. This feature can be accessed from either a BASIC program or from the keyboard.

The most impressive feature of this package is a free word-processing program. To me, it proved to be the major selling point. Most free software I have seen in the past has been inadequate, but this package offers a notable exception. This word-processor includes most of the prominent features offered by the more expensive programs but the major deficiency is the inability to format for right justification. Essentially, this software was written for the 80-column mode, so only a good monitor can do it justice.

The ease of use of this package has inspired me to do a lot more writing, hence this article and the probability of more articles to come. I can hardly say enough about this product. It has been one of the wisest purchases I have made for my computer to date. What comparable product can beat it for price, features, and performance?



BY Adams


THE ON STATEMENT. WITH: CHIPP!
HI THERE! TODAY WE ARE LEARNING ABOUT THE ON STATEMENT.
A TYPICAL ON STATEMENT IS USED WITH GO TO OR GOSUB AND LOOKS LIKE THIS:
5 ON X GO TO 10,20,80
THE ON STATEMENT IS AN INTERESTING TOOL IN PROGRAMMING AS IT WILL SEND THE COMPUTER TO DO MANY DIFFERENT ERRANDS, DEPENDING ON THE VALUE OF X.
THE VALUE OF X CAN BE DECIDED BY INPUT IF YOU WISH. IF X=1, THE FIRST NUMBER LINE MENTIONED IS EXECUTED, AND IF X=2 THEN THE SECOND IS, ETC.
ARE YOU CONFUSED?
TRY THIS PROGRAM:
5 INPUT X
10 ON X GO TO 20,30,40
20 ?"ONE"
25 GO TO 5
30 ?"TWO"
35 GO TO 5
40 ?"THREE"
45 GO TO 5
IF YOU TYPE IN "1" FOR X THEN THE COMPUTER RUNS FROM LINE 20. IF YOU TYPE "3" THEN THE COMPUTER RUNS FROM LINE 40.
EEK!
IF X IS GREATER THAN THE NUMBER OF LINE NUMBERS, THEN THE NEXT LINE NUMBER IS AUTOMATICALLY EXECUTED!
SOUNDS SCARY!
THE ON STATEMENT IS USEFUL IN "CHOICE-MAKING" PROGRAMS.
PERHAPS YOU CAN USE YOUR OWN IDEAS TO COME UP WITH WAYS OF USING THE ON STATEMENT. (REMEMBER GOSUBS!)
ALSO, THE VALUE OF X ISN'T ALWAYS DECIDED BY INPUT. THINK ABOUT IT.
?
MIKE RICHARDSON

Disk/Tape Drives Hardware

PAGE

Storage Concepts 134
Robert Dray, Peterborough, Ont.
The basic concepts of how to store data on tape and disk.

Datasette Micro-Surgery 135
Doug Drake, Ridgeland, WI
If you're having trouble with your datasette, this may be the solution.

The Lowly Cassette 137
Mayland Harriman, Pt. Arthur, Tex.
Some good arguments for sticking with a datasette and foregoing a disk drive.

Cleaning and Maintenance 138
Ian Wright, Toronto, Ont.
Some of the things the manufacturers don't tell you. It may not be so difficult as you think.

Disk Myths You Should Know About 139
Tom Van Flandern, Washington, D.C.
Yes, you can use both sides of your diskette. There are pros and cons.

Detecting Disk Format 142
Elizabeth Deal, Malvern, PA
Which disk operating system was that disk written on? Sometimes it is useful to be able to tell from within a program and this article explains how to do it.

Reading The Error Channel In Direct Mode 143
Elizabeth Deal, Malvern, PA
The red light is flashing on the disk drive and the computer screen says READY. This tells you how to find out more about what went wrong.

A Fix For The 1541 143
Elizabeth Deal, Malvern, PA
That expected compatibility with the 4040 drive just isn't there. So here is what you do now.

Using the 1541 Backup 144
David Bradley, Toronto, Ont.
When you have only one disk drive it is difficult to back up a disk. This article explains the simplest way to do it.

Keeping Track of Your Disks 145
Joel Meers, Kingston, Ont.
When you have fifty diskettes and about 20 programs on each one of them it can be difficult to find that one in a thousand program. This will help.

Storage Concepts

By Robert Dray Peterborough, Ont.

DATA STORAGE

Information is stored in the form of "files" on magnetic media, (cassettes or diskettes), and there are two main types of files: program files and data files. A file may be thought of much as you would a physical filing cabinet in which you open the file and add or remove some of the records, and then close the file. The major unit is called a file, and these are divided into smaller units called records, which in turn are divided into fields. If the file contained student information, then one record might have all of the data on a specific student, and the fields within that record would be the specific data on that student. The size of the single file is limited only by the amount of space on the device that contains the file, although the size of records or fields may be limited depending on the type of file which has been formed.

When you save a program, you are creating a program file, and the limit on the size of these files is obviously the amount of memory in your computer. When you load this type of file into your computer, the computer sorts out the file and puts everything in the correct places in memory. With data files, you must decide on the structure or arrangement of data in the file, and you need to know this arrangement to be able to retrieve the data. It's just like going into a new situation and trying to use someone else's filing system. Until you find out how it's organized, you cannot use it to file or retrieve data efficiently.

When one starts to use cassettes and/or disk drives in programming, life becomes a little more complicated. To output data to either of these devices, you must open a channel. The channel is given a specific number (1-255), which is called the "logical file number", and is opened to a specific device. Cassettes are given device numbers 1 and 2, for the two cassette ports, while the disk drive is given device number 8. Either of the physical device numbers may be changed by altering the wiring in the particular piece of equipment. The command "OPEN 4,4" opens logical channel #4 to device number 4 (the printer), and the command "print#4,a$" will send the data in the variable a$ to channel #4 which was previously opened to the printer. In a similar manner data may be sent to or received from disks and cassettes.

Because of the nature of the cassette tape, the data stored on it must be of a sequential nature, that is, one item is stored after the next, and to read item 34, you must read items 1 to 33. Each file on cassette begins with a header and ends with an end-of-file mark. The end-of-file mark causes the status to register as 64, and so you can use this to check for the end if you are reading data from a cassette file. The operator (that's you) must assume the responsibility of setting the tape recorder so that it is in the proper position to store the file.

The format for opening a cassette file is the following:

OPEN lf,dn,sa,"name"
Where: lf is the logical file number of your choice (1 - 255)
dn is the device number - 1 or 2
sa is the secondary address to specify operations
(0-read only, 1-write only,)
"name" is any 16 character name of your choice

DISKETTE STORAGE

A diskette is made of a polyethylene derivative and coated with a magnetic recording emulsion, and is stored in a protective envelope. An oval slot allows the read-write head of the drive unit access to the magnetic surface. There is another small hole in the envelope, called the index hole, which will line up with a similar hole in the diskette. A drive unit that uses this hole to position itself is said to be "hard sectored", but the Commodore drives ignore this hole and are called "soft sectored" because they line up with an electronic mark that is written on the diskette when you "Header" or "New" it. The diskette rotate at 300 rpm and have a useful life of $3x10^6$ passes per track, which is about 7 days' constant running.

The surface of the diskette has been divided into a number of concentric circles called tracks. The number of tracks depends on the disk drive, for example the 4040 sets up 35 tracks while the 8050 will set up 77. These tracks are then divided into sectors, with each sector capable of storing 256 bytes of data. Most disk drives have the same number of sectors in all tracks, which makes the inner tracks crowded and leaves gaps in the longer outer tracks. The CBM system puts more sectors in the outer tracks (21 as opposed to 17 on the inner ones), and in this way uses more of the diskette surface. The 4040 disk drive has a total of 690 sectors (blocks), while the 8050 has 2087. This gives the user 170,180 or 527,812 bytes of storage per diskette.

Files are not stored sequentially on a diskette but use a different method, which means that the disk

drive can go directly to a specific file without first reading all the previous files. This makes disk operation much faster, but there must be a very good record-keeping job done to keep track of where everything is.

Two of the tracks are used by the system for indexing the diskette. The disk drive sets up a Block Availability Map (BAM), for each disk which indicates which sectors are free, and then stores this on track #17 (on 4040 disks). Track #18 is used to store the directory. When a file is saved to disk, it must be given a unique name which is then stored on track 18 in the directory. Along with the name is stored the address of the track and sector where the program is stored. The program is not stored sequentially on tracks, since the space available for the program may have been created by erasing several smaller ones.

The directory contains the address of the first part of the program, and the last item in that sector contains the address of the sector containing the next portion of the program. In this way the disk drive locates your file sector by sector, when you instruct it to load or read the file. The disk drive has DIRECT ACCESS to each of the files, although once it has found the file, it must read it sequentially. (This is not true for all types of files).

One disadvantage of disks is that, if track #18 becomes damaged, the drive unit would not be able to find any of the files on the diskette. If you read the error channel and see number 18, then you know that the directory track has a problem.

Best of The TORPET Hardware — Disk/Tape Drives

Datasette Micro-Surgery

By Doug Drake, Ridgeland, WI

THE PROBLEM

After many months of faithful service, my Commodore Datasette suddenly started acting uncooperatively. Up to that point, I'd been amazed at the tape unit's consistency and reliability, but suddenly these past achievements meant nothing — I was ready to toss it out the window.

Perhaps the only thing that held me back was that I'd just switched from a VIC to a 64. My first 64 was defective, and I'd had to return it for another one. Perhaps there was some obscure problem in my second 64 as well? There was just enough doubt in my mind to let me step back and analyze the situation.

My patience paid off in the form of a simple solution. If you have the same problem, perhaps this will save you a trip to the repair shop.

The problem was that the Datasette was shutting itself off. I would begin a LOAD, and everything would progress normally for a while. When the tape stopped (so the 64 could display 'Found Program') and then started again, an ominous clicking noise would begin. After a few seconds, the PLAY button on the Datasette would pop up, and the tape would come to a screeching halt.

Needless to say, the LOAD was unsuccessful. When I held the PLAY button down by brute force, the tape ran a little longer, but the result was the same — an unsuccessful LOAD.

I began my effort to solve this problem with the standard Datasette maintenance operations. I gave the unit a thorough cleaning to remove all the dust, cleaned the heads with a commercial cassette head-cleaner, and moved the unit as far as possible from the monitor.

When these efforts were unsuccessful, I opened up the unit and lubricated (very lightly) all of the appropriate moving parts. The problem remained.

With that, I decided to shift gears a bit, to focus on defining the problem rather than blind attempts to solve it. Since the stopping and starting of the Datasette motor was obviously a contributing factor, I wrote a short program to write and read a data file. The file was long enough to force the motor to stop while the cassette buffer emptied.

FINDING A SOLUTION

I studied the still-assembled Datasette's innards from top to bottom as my test program ran. I found a series of levers and springs which caused the clicking noise I'd heard. These were connected to a small, white piece of plastic located

on the top side of the mechanism, between the Erase Head and the Read/Write Head (see Diagram 1).

This plastic piece moved forward, toward the tape, along the Read/Write head. It slipped into a small slot in the cassette. When I pushed the PLAY button (without a cassette in the unit), and then pushed backwards on the plastic piece with a pen, the unit shut itself off! The purpose of this set-up is to turn the Datasette off at the end of your tape.

I tried my test program once again. But this time I pushed that plastic piece toward the tape as the data was written, and this time it worked! I tried to read the data, again applying pressure, and again I was successful.

Obviously, it was time for some Micro-surgery.

The only tools required for this procedure are needle-nose pliers, a small Phillips-head screwdriver, and a flathead screwdriver. First, unplug the unit from your computer, turn it over and remove the four Phillips screws from the bottom. Turn it right-side-up and lift off the top cover. Now, simply grab the offending plastic piece with the pliers, and push down on the metal base below with the flathead screwdriver. Slowly and gently, move the pliers back and forth as you pull upward. Be sure you push down with the screwdriver so you don't bend the metal base.

It took me about five tries of sliding the plastic piece upward, little by little, before it finally popped off. Be gentle and patient. Now re-assemble your Datasette, making sure the cord in the back fits into its slot when you replace the cover.

After this minor surgery, my Datasette once again works just fine!

As to whether the piece I removed is really necessary, I think not. I examined some other cassette tape players and they didn't have anything similar, so it isn't needed to keep the tape aligned on the heads. My unit won't automatically shut itself off at the end of a tape anymore, but how often do you get all the way to the end of a tape anyway? Plus, you can always write a file with an EOT marker at the end of the tape (Secondary Address 2), and the computer will stop the tape motor for you.

Thanks to micro-surgery, my Datasette is once again its old, reliable self.

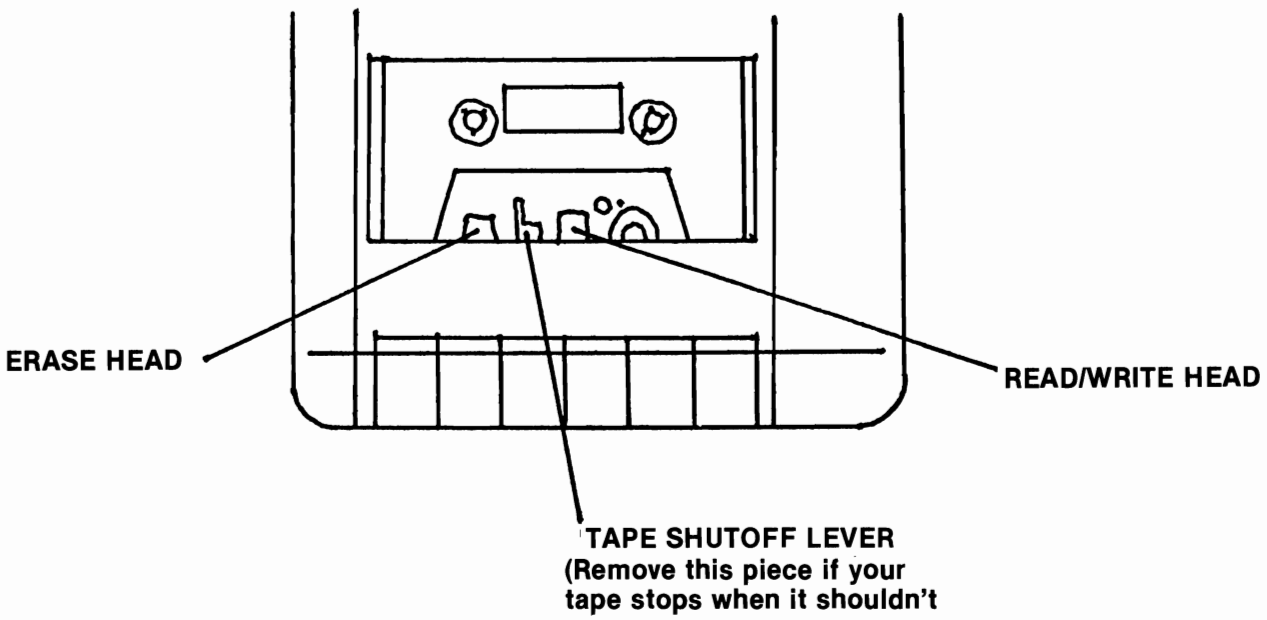


DIAGRAM 1

The Lowly Cassette

By Mayland Harriman, Pt. Arthur, Tex.

Are you embarrassed because you only have cassettes for programs? Does it seem like everyone else has dual disks and looks down on you being backward? Do you get a feeling of inferiority when you pick up a computer magazine or newsletter and the writers are ganging up to make you feel bad when they write that cassettes and tapes are no good and that they predict that most computer stores will not even carry them in the near future?

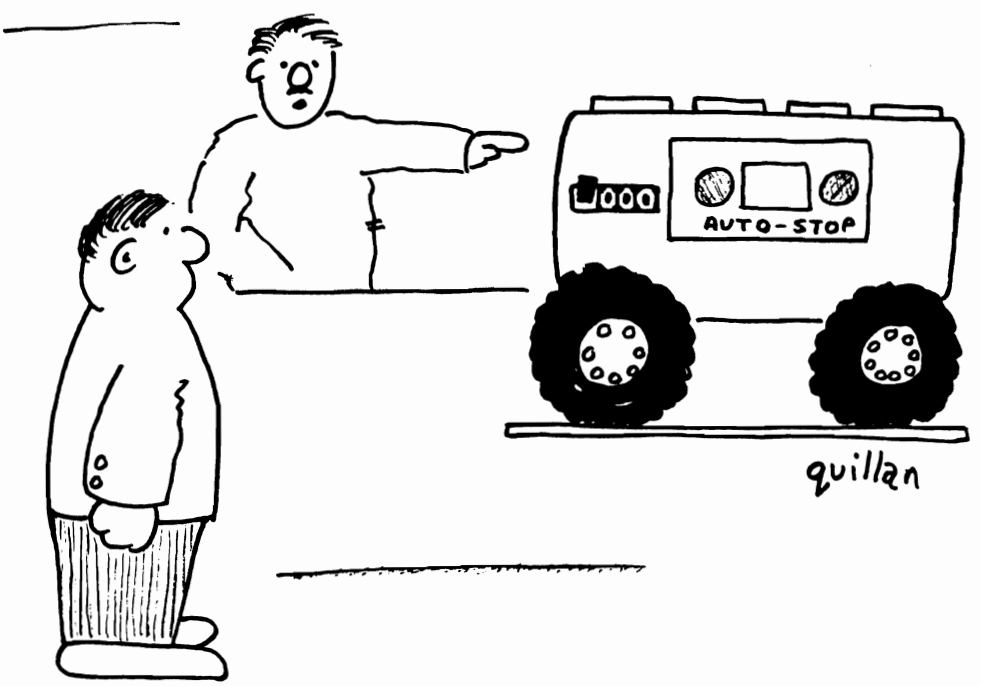
Take heart old tape friend, you need to be aware of a few things that may cause you to look the bad-mouthers in the eye and tell them they don't know what they are talking about.

The cassette method of storage is the lowest cost way to keep your programs from the initial purchase price to the end results in running your programs. The tape is much slower than disc and it is harder to SAVE a program and DATA together when needed to be stored that way....but remember many of Commodore's computers are designed to use TWO CASSETTES and that narrows the field of objections a little bit more.

There are several programs designed to find a program quickly on a tape and each one I get does the job faster. But better than fast locating is a new chip that I have ordered for my CBM 2001 and which is available for the VIC as well. The chip called the ROM RABBIT allows you to load an 8K program in about 30 seconds!! Doesn't that sound good? The chip also gives you 12 commands, allows every key on the keyboard to repeat and a few other goodies.

Someday, IF I have many complicated programs with lots of data to save and IF my time becomes much more valuable than it is now, I might go to disc....but remember one disc is considered not enough, you have to have two and that is MONEY, MONEY, MONEY, which I don't care to spend.

Yes, the cost of ADD-ONS is coming down but Disc Drive prices would really have to drop to rock bottom before I can justify the expense. It isn't going to hurt me to type LOAD and go get a cup of coffee or read a few paragraphs in a magazine or something while my CASSETTE does its job....of course with my new ROM RABBIT chip I will load the programs that I use most of the time in about 45 seconds! I can live with that!



Cleaning and Maintenance

By Ian A. Wright, Toronto, Ont.

Since attending a meeting where questions were answered by a panel about using Commodore products, I have gathered more data on similar topics and added them in where appropriate. Some of these ideas originated from others via various Bulletin Board Systems.

CLEANING AND MAINTENANCE

Clean and de-magnetize tape decks but, unless you are very competent, don't take them apart.

Many tape-read errors result from badly-aligned heads. There have been articles about head adjustments (Compute! #8), or take it to your dealer.

Some disk drive manufacturers have stated that the various disk cleaning kits can do more damage than they repair. Many people are using them with no complaints.

Cigarette ash is the worst danger to the keyboard, and some members have already bought a number of $75.00 keyboards. There are some things that can be done to improve a 'tacky' board before having to buy a new one. If you are not prepared for the 23 tiny screws that remove the back cover, and a lot of picky cleaning with swabs, then take the machine in to the professionals. Use 111 tri-chloryl ethane or a tape-head cleaner on the circuit board and the rubber key inserts. Rubbing alcohol is not good enough because it leaves contaminants behind after evaporation.

A vacuum cleaner is a valuable maintenance tool for keeping equipment in operating order. I have removed dustballs, pencils and an eraser from various machines at my school. Printers seem to be particularly apt to collect debris.

DISK DRIVE PROBLEMS

The 1541 disk drives that have trouble writing to track 1 on double-density disks can be helped by not using 4040 formatted disks. As a general rule, you should format and write on only one type of drive, although any disk can usually be read by another drive.

Since this problem was presented at the meeting, I have lost one disk of files because of writing from one drive to another. I have three friends who have had the same experience. Although all disk drives of the 2040 and 4040 type can read disks formatted on each other, do NOT write between them. The problem may not show up for months, but one day...blippo...no files! This is especially true of single/dual drive interchanges. We have instituted a system in which all files are kept on 4040 formatted disks. A temporary file is written to a 2031 (or 1541) format disk and then copied onto the 4040 disk for storage and later processing.

Verbatim #577 disks have had some problems in use with 8050 drives. The solution was to use a bulk eraser to clear away spurious magnetism that was between the tracks. Verbatim #525's have been used reliably, and most other manufacturers have reliable products.

There is a new 2.7 ROM set coming for the 8050 which indicates in which drive an error has occurred.

Commodore is still making the 4040 dual drives, but only in intermittent production. The new 2031 SL drive is the slim-line replacement for the original (1981) single drive. So far, there has been encouraging lack of complaints about its operation, unlike its predecessor.

Many disk errors can be solved by correct centering of the disk in the drive. Make a habit of starting the disk in motion, then slowly closing the drive door. Chris Bennett says that he had hundreds of errors before learning this trick with the 2040 and 4040. The disk copying errors can be reduced to negligible using this approach.

If a disk is validated or collected and a bad file is not removed by this process, copy the good files using Copy-All and re-format the old disk. Do not continue to use the disk.

Sometimes a disk can be recovered by formatting the reverse side. Although double-siding is not a good idea, this trick may prove useful in some cases where you want to retrieve material from the original side.

HERE ARE FOUR DISK RULES A LA BUTTERFIELD

1. If you attempt to write on a disk that has a write-protect tab, an error will occur. Before continuing, re-set the drive by turning it off/on.

2. If a file is not properly closed (it has an asterisk

beside it), do not attempt to scratch the file. Leave it alone or collect the disk (see also above).

3. Don't leave two disks with the same I.D. in the same room. The back-up facility makes it easy to insert a back-up disk with exactly the same I.D. into the drive without resetting it. The drive may not recognize the back-up as a different disk, and may continue writing where it left off!

4. Don't turn off the drive with a disk in it — and never when the drive is spinning. The drive may do weird things as it 'loses its brains'.

If there is no BAM, then you can use the tip #3 above to try to retrieve information. Initialize a new disk with exactly the same header as the bad disk; now slip in the bad disk and read track and sector if possible.

A read error means that you cannot depend on the data on the disk. A check-sum error can be looked at, retrieved and re-written.

A disk can be re-set without touching the on/off switch by OPEN 1,8,15,"U2: then CLOSE1. This will work with the disk in or out of the drive.

A USR file is a sequential file that has a special protocol that may differ from the standard ASCII. This designation allows the catalog to show a file as 'special' in its format.

GENERAL INFORMATION

There are "new" manuals and reference guides available from Commodore that were printed in 1982. These include data on the 9060 and 9090 hard drives. There is no data on the slim-lines.

Epson has a new printer manual for the Mx-80, again published in 1982. This manual includes a tutorial on various functions including Graftrax + use.

Commodore can be considered to be as good as most other manufacturers in terms of their program transportability between machines. Despite our problems, programs that are written without 'frills' can run on all machines. Many manufacturers introduce new models with no carryover, whatsoever.

Disk Myths You Should Know About

By Tom Van Flandern, Washington, D.C.

This article about disk myths originally written by Tom Van Flandern for the Washington Apple Pi Newsletter, June, 1982, comes to us by way of NORTHERN BYTES, newsletter of the Computer Users International in Sault Ste. Marie, Michigan and Ontario.

Is the price of blank diskettes a constant drain on your budget? Read on! You'll be glad you did. Discussions of the subject of diskettes usually result in the propagation of "disk myths", or statements about diskettes which have three attributes: 1. They originate from diskette manufacturers or dealers, not users; 2. they are all reasons why you should pay more for your diskettes; and 3. they are untrue. Let's consider the most common of these myths.

ONE-SIDED MYTH

You have probably been exposed to the controversy over using one or both sides of your "single-sided" diskettes. I have often heard the myth repeated that the manufacturers put their label on whichever side of a diskette that their surface quality tests. By implication, the other side may have failed such a test and therefore may be expected to be of inferior quality. Sounds plausible, doesn't it? Cuts manufacturing costs, and why certify both sides when only one is usable as the disk is sold? There is just one problem with the theory — the box of diskettes doesn't know what type of computer or drive it is headed for use on. Did you know that Apple disk drives always write on the bottom side of your diskette? Commodore disk drives do the same; however, there is no standard among computers. Some have single-sided disk drives which write on one side; others may write on the other side. Manufacturers are therefore obliged to certify both sides of diskettes with equal care.

TWO-SIDED MYTH

What keeps you from turning your diskettes over and using the magnetic surface on the other side? There is a small rectangular notch along one edge, centered at 1 5/16 inches from the top edge

of your 5 1/4 inch floppy diskettes. This notch permits your disk drive to sense that it is okay to write on the disk. if you cover this notch, the disk is write-protected. To make the other side usable, just punch a similar hole along the opposite edge at the same distance from the top. Turn the diskette over, insert it into the drive and use in the normal way. The shape of the notch is not important — circular or rectangular are equally good — but it must be at the correct location, about 1/4 inch wide and not quite as deep. Use an ordinary hole punch for good results. To get the location correct, just turn over another diskette and line it up with the one to be punched. For mass production, make a mask 5 1/4 inches long which can be placed quickly over the disk to show you where to punch. Don't be concerned if you get the hole slightly too large. Your chances of damaging the diskette are small with ordinary care and are less from making too large a hole than they are from using too crude a cutting instrument, causing the diskette to be pinched inside its cardboard jacket.

The disk myth in this connection is that you risk losing data on the original side of the disk if you write on the other side in a one-sided drive. The reason cited is that magnetic particles will accumulate on the pressure pad which presses against the side opposite the read/write head, and these can destroy information on the side they come in contact with. The principal argument against this theory is empirical — it just doesn't happen, at least not over a period as short as a few years in ordinary usage. (See caution below under "The Cleaning Kit Myth", however.) The failure rate for diskettes used one-sided and two-sided is statistically indistinguishable, resulting in an interesting correlation. The probability of a micro-computer owner using his diskettes two-sided is directly proportional to his experience. Almost all users eventually try this, and the best proof of its effectiveness is that they stay with it. The most experienced owners, with the largest files, almost all use their diskettes two-sided, and smile knowingly at the novices who are reticent because, "if it were that simple, the manufacturers would tell you so!!".

DOUBLE-DENSITY MYTH

This disk myth is insidious, because the manufacturers allow the consumers to fool themselves and simply fail to provide them with information needed to correct the myth. Double-density diskettes cost more because they have a thicker magnetic coating. So they must surely be better, right? Why not keep your really important files on double-density diskettes? Woe to you, naive and trusting user. The purpose of the double-density diskettes is to support disk drives capable of generating a stronger magnetic signal than normal drives. This is usually needed if more bits are to be written per inch, but is quite unnecessary for the information density at which normal disk drives operate. More importantly, though, since the signal generated by normal drives is not strong enough for double-density diskettes, you actually have a slightly higher risk of losing those valuable files if you wrote them on a double-density diskette!!

HUB RING MYTH

Some diskettes come with hub rings, and this too is supposed to be worth paying extra for. Hub rings are circular bands on the inner edge of your diskette which provide extra strength to that edge. Their main function is to keep the inner edge from getting crunched if the diskette is off-center when the lid and pressure pad are lowered after the diskette is inserted into the drive. With just a minimum of care however, the lid can be closed slowly and lifted and closed again if it meets resistance, so as not to damage the diskette. Another recommended practice is to boot your disk and start it spinning before lowering the door lid. (As far as I can tell, this is NOT possible for those using the 1541 single disk drive. JM) This not only aids self-centering, but also prevents the read/write head from pressing against the disk surface as it retracts for recalibration (the clacketty noise you hear). In other words ordinary good disk-handling practice (which even children can be expected to follow) will allow the diskettes a chance to self-center and prevent damage. The problem caused by the hub rings is that, if the diskette has any tendency to bind in its jacket, preventing it from gaining full rotation speed, it is easier for it to slip with the hub rings than without. If you ever try a disk-speed test and occasionally see some measures go off the scale, this is usually from binding up and may be exacerbated by hub rings.

THE NAME BRAND MYTH

Occasionally a brand of diskettes in its entirety or a particular batch of diskettes from some well known manufacturer, will be flawed and produce much user grief. However, there is a lot of incentive for manufacturers who want to stay in business to prevent this from happening and most are successful. Once the diskettes pass the surface certification tests, if they are properly shipped and handled, they are essentially equally good, regardless of name brand or claims to the contrary. Almost all diskette failure is due to handling problems (see below). Failure rates of

factory-shipped diskettes are about 12 per 1000, on average, with little variation between brands and no correlation with price. The myth here is that paying more for a name brand buys a tangible benefit.

In fact, many generic brand diskettes are available, often made by the same big-name manufacturers but without the name brand label, for much less cost than the identical diskette with the label pasted on it. Is the label really worth that much extra cost to you? There is also the question of whether a manufacturer will stand behind its guarantee. Apple Avocation Alliance recently reported that Verbatim refused to honor its diskette guarantee and criticized the Apple organization for selling Verbatim's "too cheaply".

THE CLEANING KIT MYTH

"Buy a cleaning kit for your disk drive. Clean the read/write heads at least once a week." Before I knew any better I bought just such a head cleaning kit. At the time I wondered at the important notice on the box, which I quote in part: "Neither seller nor manufacturer shall be liable for any injury, loss or damage arising out of the use of the product. Before using, user shall determine the suitability of the product for his intended use, and user assumes a risk and liability whatsoever in connection." I assumed, as most people must, that this was just legal mumbo-jumbo to protect the manufacturer from frivolous lawsuits by incompetent users. After all, the product was being widely sold for the purpose of cleaning disk drive heads, and that was surely a desirable end. Wrong again! I began to have one diskette failure after another and it was several months before I realized the correlation with use of the head cleaner.

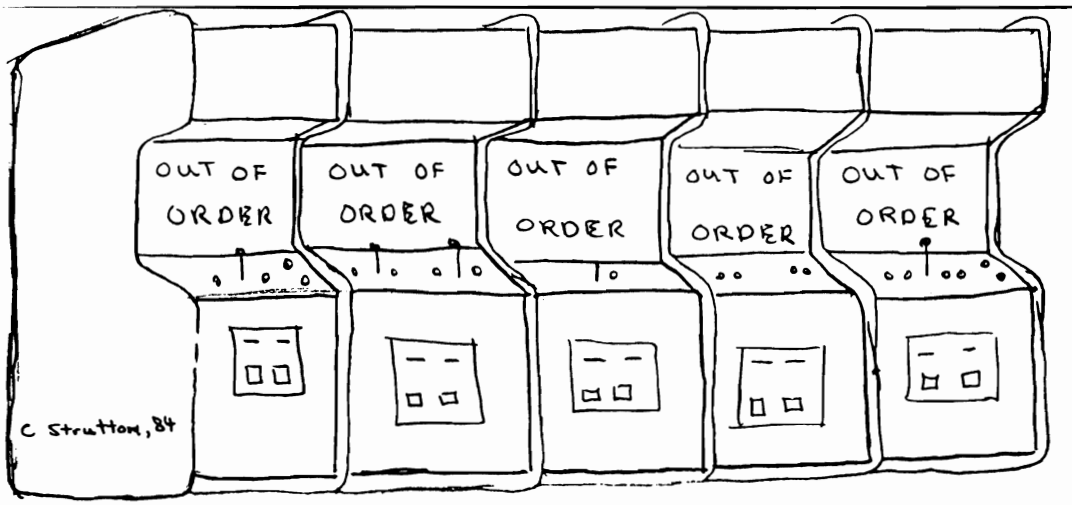
The sad truth is that the cleaning fluid used with the kit is a strong solvent. The recommended method of application results in the pressure pad getting soaked with solvent. If you then use a diskette in the drive, the magnetic surface on the other side of the diskette is scoured by the solvent and actually dissolved in the process! The damage can be so extensive that it may no longer be possible to initialize the damaged surface. Of course if I had not been using my diskettes two-sided, I might not have discovered the problem. But I now know that, in truth, head cleaning usually needs to be done at intervals of one to four years, not weeks, and is easily done with alcohol applied directly to the head, without damage to either pressure pad or diskettes.

PROPER CARE AND HANDLING

Most diskette failure is caused by improper care and handling, rather than anything under the control of the manufacturer. Of course, diskettes must be kept away from magnetic fields, such as emitted by some TV's and certain other electronic devices. They must be kept clean and dry. And the importance of never writing on a diskette label with an object which can apply pressure to the magnetic surface below cannot be overemphasized.

Perhaps the single most common cause of random diskette failure not caused by disk drives is binding in the cardboard jackets. This is why you are advised to store diskettes vertically and avoid the temptation to stack them horizontally. Anything which applies pressure to the jackets (including crowded storage of diskettes, horizontally or vertically) can cause binding, which prevents the diskette from spinning at full speed continuously while in use, which causes intermittent failures.

I hope the preceding information proves useful to you and saves you money as well as headaches with your diskettes.



DEFINITION: HELL

Detecting Disk Format

By Elizabeth Deal, Malvern, PA

This program is on The Best Programs Disk

1541 and 4040 floppies are not write-compatible. A floppy formatted on a 4040 and subsequently written on by the 1541 experiences a slow self destruction. The initial signs of trouble are LED and stepper-motor hiccups. This is shortly followed by DISK ID MISMATCH errors, often accompanied by silly track and sector numbers. Finally, the floppy can be read no more.

The same is probably true the other way around, though I haven't tried it, one set of troubles is enough.

A superficial look at the floppies does not reveal on which drive such a floppy was formatted, both have a "2A" sequence in the directory name.

Jim Butterfield says (I think) that the synch marks attached to each sector are different. The recipe for reading the synch marks includes a good bit of magic potion, a white fuzzy kitten on the roof, and a tornado in West Chester, PA ... all at the same time. So here is the easier way:

The 1541 formats differently from a 4040. The 4040 fills the entire disk with zeros, the 1541 fills it with ones, and sets all "next track pointers" to 75. This explains why several disk messages invariably report ILLEGAL TRACK AND SECTOR 75, etc., and an unfinished directory in "newing" has all file names AAAA with 256 blocks each.

Relying on this last bit of information we can detect on which drive the disk was formatted, provided that the floppy is not full.

```
340  REM----------------------------------
350  Z$ = CHR$(0)
360  DV = 8:D = 0:REM DEVICE, DRIVE
370  T = 29:S = 4:REM TRACK,SEC
380  OPEN15,DV,15:OPEN1,DV,3,"#"
390  GOSUB450:IFETHENSTOP
400  PRINT#15,"U1"3;D;T;S
410  GOSUB450:IFETHENSTOP
420  FORJ = 0TO7:GET#1,I$
430  PRINTASC(I$ + Z$);NEXTJ
440  CLOSE1:CLOSE15:END
450  INPUT#15,E,E$:PRINTE;E$:RETURN
460  REM----------------------------------
                    4040    RETURNS 00000000
                    1541    RETURNS 75 1111111
```

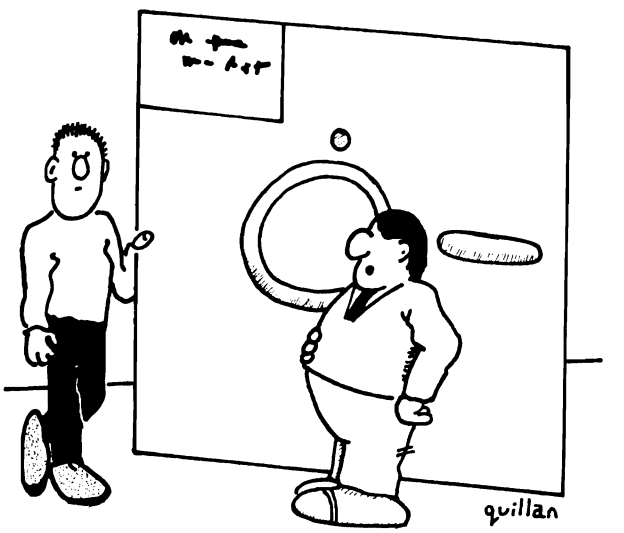
NOTE: This program will not work if there are zero blocks free on the disk being tested.

If you get results other than a chain of zeros or ones, you are not using an empty sector, so change T and S in a program until you do get a set of either ones or zeros. If you can't get any such clean chain, then somebody has been fooling with the disk, and you're on your own.

My routine will also NOT detect a disk formatted on the 4040 but then changed to look like a 1541, for this you'll have to go after the synch marks, but it is an unlikely event to happen.

In any case, it is just not safe to write on a disk that you did not format, so take all the necessary precautions, whatever the results of the above routine.

Be careful with purchased software, sometimes a sticker says "1541". A sticker may not be relevant. Anybody can print any kind of a sticker they wish. What's inside is what counts, especially if the program writes on the disk, as is the case with high scores in games, to cite just one example.



I WANTED AN EIGHT-INCH FLOPPY DISK, NOT AN EIGHT-FOOT ONE!

Reading The Error Channel in Direct Mode

By Elizabeth Deal, Malvern, PA

It can be done!

Commodore 64 and Upgrade PET computers normally can't look at the disk error channel the same way as Basic 4 systems, since we cannot use GET or INPUT in direct mode. So, we patch what we can with the DOSO-wedge or POWER, POWAID, MOREPOWER — whatever we've got. But, sometimes, those utilities get clobbered, especially the wedge, since a lot of people put their code in fixed places. The alternative is to enter program lines, but that clobbers the program all too often. So, it's one trouble chasing another.

We are in luck now. Howard Harrison of Philadelphia passed this gem to me: if we enter the GET# routine several instructions past its beginning to avoid the check for direct mode, we can, in fact, use GET to read the error channel. It will not work with INPUT#, as the direct mode check is buried inside the routine. So we type, all on one line, if you wish:

```
CLOSE15:OPEN15,8,15:
FORI = 0TO30:SYS(51844)#15,A$:PRINTA$;:
   IFST =  0THEN  NEXTI
```

This is for the PET. For Commodore 64, use SYS(43906).

It's not exactly as easy as Basic 4 PRINT DS$, but it does the job.

The parentheses around the address are not needed. You can even stick in spaces between the address and the number sign. And, if you keep one and the same file open to channel 15, you can skip the open/close typing.

A Fix For The 1541

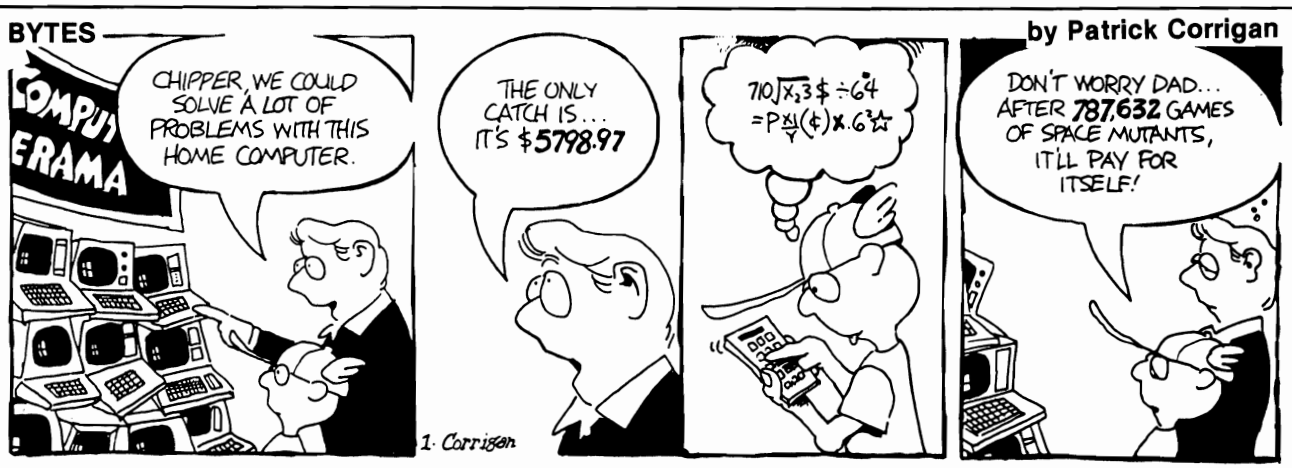
By Elizabeth Deal, Malvern, PA

One problem with 1541 disk drives is that sometimes they are unable to reliably read tracks 1-2 and 34-35, and unable to write anything, anywhere. In this case, the read/write head may have a bit of trouble moving fast and far enough.

Use a Q-tip and some mineral oil on the shaft on which the head travels. To do this, you need to disassemble the 1541, a job best left to qualified people, as the parts you have to get to are buried under lots of delicate chips and things.

It must be mineral oil, not the 3-1 sewing machine kind and not silicon sprays. Part of the magic, I suppose.

While this procedure is most reasonable, I cannot accept the responsibility for you botching the job. It is your disk drive, so, please, use good judgment in fixing the problems...if you have them.



Using the 1541 Backup

By David Bradley, Toronto, Ont.

This program is on The Best Programs Disk

First of all, if you don't have the 1541 Backup program and you want to copy disks using your 1541, get it.

Once you have the program loaded into your Commodore 64, type in RUN and press return. There will be a slight pause before anything appears to be happening, so don't worry if it doesn't jump into action immediately. When the program is finished setting up, there should be several "boxes" displayed on the screen.

The first thing the program will instruct you to do is to enter the program operation code.

What the program is asking you to decide is whether you want to do a BAM select backup or a direct backup. You choose this by typing in either a B or a D and pressing return. If you are not sure which to choose, I will try and explain what the difference between the two is.

The BAM Select Backup will only copy the areas of the disk that have information on them, while the Direct Backup will copy every track and sector, whether it be empty or full. The Direct Backup should always take the same amount of time, whereas the BAM Select Backup will vary, depending on how full or empty the disk is.

Now it is time to get your Destination disk formatted. The program will ask you to enter disk name. The name of the disk is what you see displayed in reverse field characters when you list the directory of a disk.

After you have named the disk, the program will tell you to enter ID number. The ID is a two-character code that is also displayed in reverse field characters when you list the directory. You have to be careful, when using this program, to make the ID of the Destination Disk different from the ID of the Source Disk.

Now you will be instructed to enter destination disk into drive. Before you go on, get it very straight in your mind which disk is which. The Destination Disk is the disk that you are copying to, and the Source Disk is the disk that you are copying from. So, put the disk that you are copying to in the drive and press return. If all is well, the program should display Formatting Destination Disk and the disk drive should be working.

Once the Destination Disk has been formatted, the program will tell you to insert source disk into drive. Before trying to do this, be sure that you have removed the Destination Disk. Once the Source Disk is in the drive, press return. This tells the computer that you have done your part and it is time for it to proceed with its duties.

There should once again be disk activity and the program will display reading BAM from Source Disk. After about five seconds, the computer will request that you verify source disk for backup. All you have to do is press return and the computer will check the BAM it has stored in memory against the BAM on the disk. This is done to ensure that no errors have occurred.

Then the program will inform you that it is reading data into buffer. Notice the "Bar" near the top of the screen. If all is well, that "Bar" should be getting longer.

When the buffer is full, the program will tell you to insert destination disk into drive and press return. The program should tell you that it is writing data from buffer. Now the "Bar" should progressively get smaller.

When the buffer has been drained, the program will tell you to insert source disk into drive. Once again, put the Source Disk into the drive and press return.

From here on, all you have to do is continue switching the disks when the program prompts you to until the program says backup finished. When that happens, if you did everything correctly, you should find that all of the programs from the Source Disk are now on the Destination Disk as well.

Good luck....

My wife was looking rather folorn as she hunched over our C64 waiting for a particularly slow output to appear on the monitor. I suggested she join me in a chorus of "Someday my PRINTs will come."
— Ylimaki

Keeping Track Of Your Disks

By Joel Meers, Kingston, Ont.

I have been a member of a club for a year now, and I am finding that I have club programs, friends' programs, and heaven-knows-where-I-got-them programs coming out my ears. I have found myself faced with some problems which I felt other computer fanatics must be having.

Computer Fanatic to Self: "Now what disk is that so-and-so program on????"

Computer Fanatic to Fellow Fanatic: "Oh say George, I have just the program to help you out! I found it a couple of weeks ago on one of my club disks....now what was the name of that thing???"

Computer Fanatic to Self: (this talking to yourself happens, you know: possible side effects of VDTs) "I know I have a program that does that, and I know it's on one of those utility disks, but which one was it? Hmmmm."

These are just a few of the problems I have encountered, not to mention trying to remember what each of the programs on that new club disk did or how well it did it. Put the directory on a printout, you dummy, you say. Well, I do. Pity the poor guy who doesn't have one! Sure, this is great, but how many of us with a few dozen or so printed directories can remember what they all do?

I tried Disk Master, but who wants to wait to read through twenty sequential files, and then not know what the program does. I filed that program in my head as not practical.

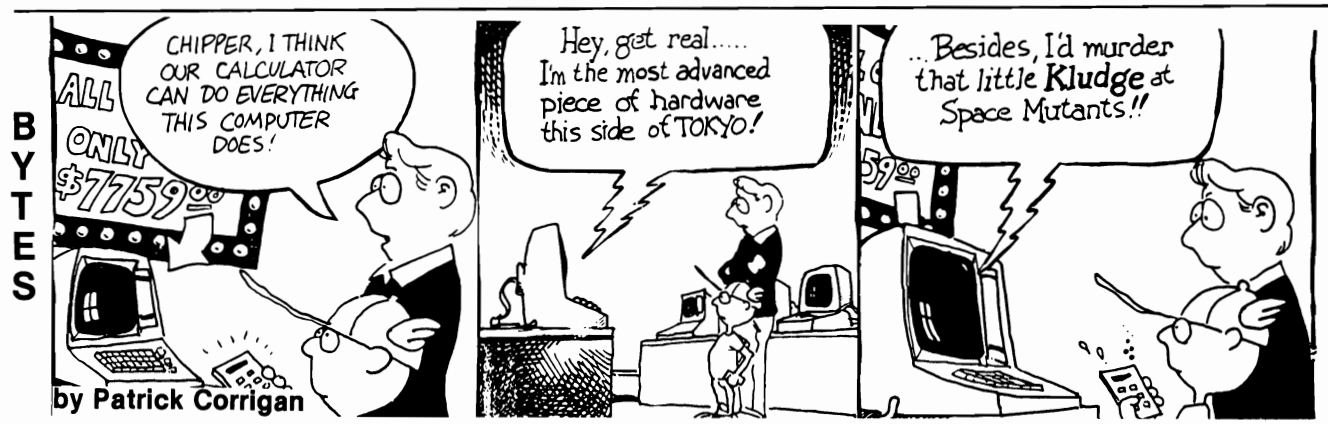
Enter the Data Base. I have been working mainly with two D Bases, Delphi's Oracle, which is superb and very sophisticated for a D Base in that price range, and Flex File. I found Flex File first and, when the Oracle came along, I pretty much mothballed my Flex.

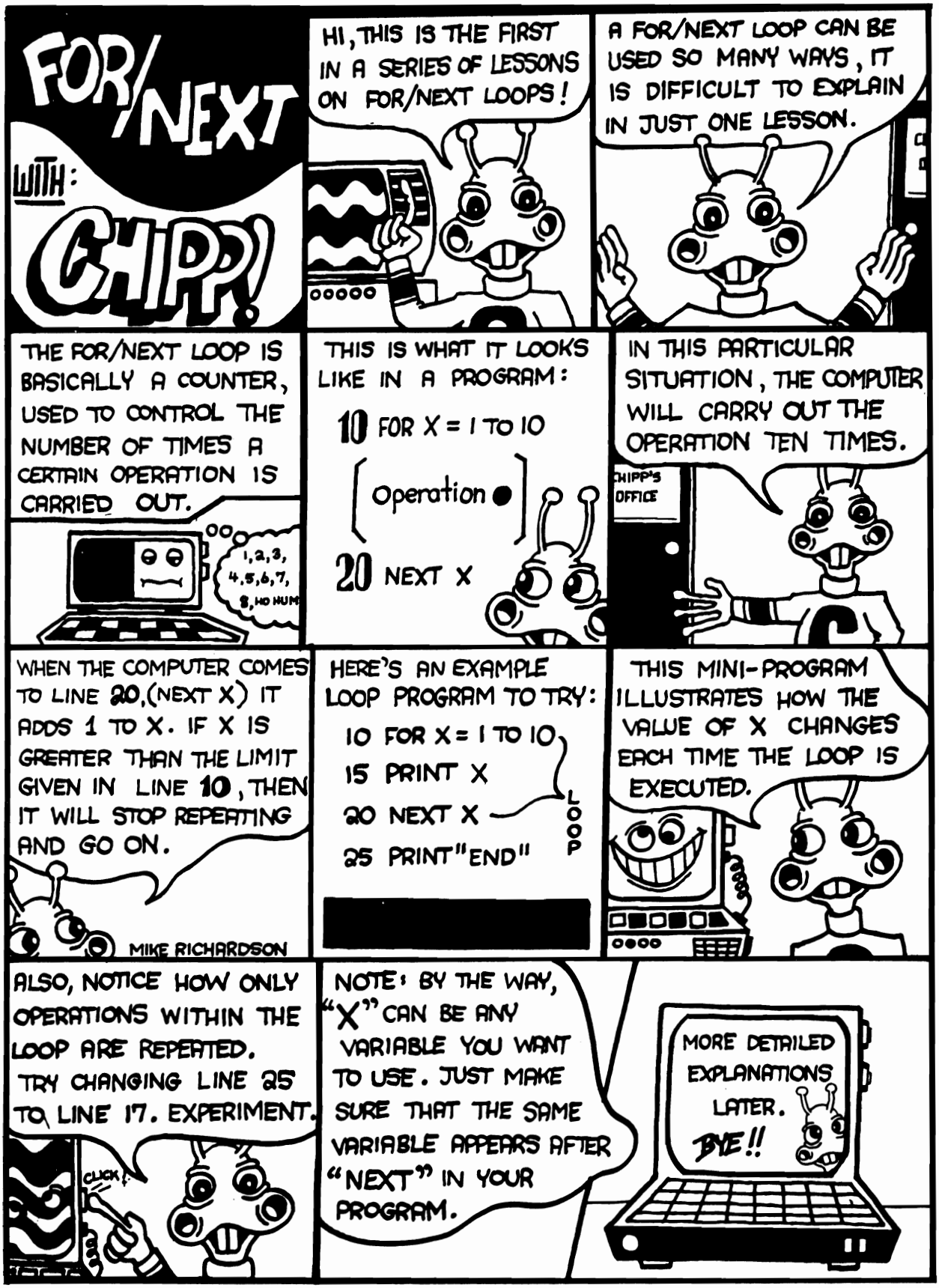
Any data base would work, but I found Flex File to be less complicated, and the task I wanted to perform was just that. By setting up three key fields — 1. PROGRAM, 2. DISK, and 3. DESCRIPTION — I was away.

The PROGRAM field was set up for 16 characters, the DISK field for 16, and the DESCRIPTION field for 40. This can all vary, of course, but, for maximum capacity, you want to keep the number of characters down. By the way, I ended up scratching the mail label, errata and any unnecessary programs from the disk and wrote my data on the same disk. I could get 1000 records on the disk and one disk makes using this idea just that much simpler. I also designated a letter code for the first character of the DESCRIPTION field, i.e., G-game, U-utility, E-education, etc., allowing me to use the third field as a category search field as well as just holding text.

With this very simple format, I can search on any field for a particular program, a particular disk, or a type of program. I can immediately get the location and a brief description of the program. This is great for those without a printer. For those with, the report generator on most D Bases will then give a directory listing with short descriptions of each program.

Maybe lots of you have tried it, but, for people like me who are newly struggling with these problems, this might be of use to someone. By the way, there are possibilities for this format on any number of a variety of D Bases or mail label type programs. Heaven forbid, I even tried it on PFS FILE on that unmentionable computer with the stem, you know, the one with the 'byte' missing on the side.




FOR/NEXT
WITH:
CHIPP!
HI, THIS IS THE FIRST IN A SERIES OF LESSONS ON FOR/NEXT LOOPS!
A FOR/NEXT LOOP CAN BE USED SO MANY WAYS, IT IS DIFFICULT TO EXPLAIN IN JUST ONE LESSON.
THE FOR/NEXT LOOP IS BASICALLY A COUNTER, USED TO CONTROL THE NUMBER OF TIMES A CERTAIN OPERATION IS CARRIED OUT.
1,2,3, 4,5,6,7, 8, HO HUM
THIS IS WHAT IT LOOKS LIKE IN A PROGRAM:
10 FOR X = 1 TO 10
[Operation ●]
20 NEXT X
IN THIS PARTICULAR SITUATION, THE COMPUTER WILL CARRY OUT THE OPERATION TEN TIMES.
CHIPP'S OFFICE
WHEN THE COMPUTER COMES TO LINE 20, (NEXT X) IT ADDS 1 TO X. IF X IS GREATER THAN THE LIMIT GIVEN IN LINE 10, THEN IT WILL STOP REPEATING AND GO ON.
MIKE RICHARDSON
HERE'S AN EXAMPLE LOOP PROGRAM TO TRY:
10 FOR X = 1 TO 10
15 PRINT X
20 NEXT X
25 PRINT "END"
LOOP
THIS MINI-PROGRAM ILLUSTRATES HOW THE VALUE OF X CHANGES EACH TIME THE LOOP IS EXECUTED.
ALSO, NOTICE HOW ONLY OPERATIONS WITHIN THE LOOP ARE REPEATED. TRY CHANGING LINE 25 TO LINE 17. EXPERIMENT.
CLICK!
NOTE: BY THE WAY, "X" CAN BE ANY VARIABLE YOU WANT TO USE. JUST MAKE SURE THAT THE SAME VARIABLE APPEARS AFTER "NEXT" IN YOUR PROGRAM.
MORE DETAILED EXPLANATIONS LATER.
BYE!!

Printers Hardware

PAGE

Buying That First Printer 148
Stan Koma, Rexdale, Ont.
There are lots of terms to be understood. IEEE standard, Centronics Standard, RS-232 standard, Parallel, and Serial, Dot Matrix and Letter Quality. This explains a lot of the details.

Garage Sale Printer 149
Gary Hayes, Garden Grove, CA
There are some bargains to be had if you are not too demanding and are willing to look.

Selecting A Printer 150
Gene Wilburn, Mississauga, Ont.
Some advice about what to look for and what to look out for can save you a bundle.

VIC and RS-232 Printer 153
Daryl E. Williams, Santa Ana, CA
Another one of the standards available is examined.

VIC Printer Routine 153
Michael Kelinert, Nanuet, N.Y.
Here is a program to help you get more use out of your printer.

Re-inking Printer Ribbons 154
T.J. Bos, Brantford, Ont.
The little beggars can be expensive and if you do quite a bit of printing this is a good way to save money.

Re-inking Printer Ribbons 154
E. Toussaint, Shelley, W. Australia
Every idea can be improved upon, and here is a follow-up article on the same subject.

1526 — Good Printer, Lousy Manual 155
Howard M. Mesick, Hartly, DE
A user gives some additional hints not found in the manual about using the 1526 printer.

Buying That First Printer

By Stan Koma, Rexdale, Ont.

Buying a printer is like buying a car.

When you enter a car lot, the first thing that strikes you is the variety of cars on display. You take a look at that medium range car. It's the right size for the family...and the dog. It's within your price range, the gas mileage is quite good and it will also be big enough to take a few hockey players and their equipment to the arena. That's the car.

This is the kind of thinking process that is required in buying your first printer. You have to be prepared to make "trade-offs" to select the right printer for you. After I started looking for the "ideal printer", I soon discovered that the only ideal printer is the one that does the things you want it to do best. It may not do everything you want, but it should do well those things that you need most.

If price is no object, all you have to do is buy the top of the line printer and all your problems are solved. But for the rest of us, there are certain perimeters that force us to make a selection based on specific personal needs and a certain price range. I had decided that my price range would be about $1,000.00.

I did not make an exhaustive study of printers. If I did, I would still be looking. You have to make some trade-offs even in the amount of time you want to spend searching.

TYPES OF PRINTERS

Basically there are two kinds of printers - the dot matrix and the letter-quality. The dot matrix machines create letters and symbols with a series of dots. The greater the number of dots, the better the reproduction. Letter-quality printers "type" perfectly-formed letters when a hammer strikes a rapidly revolving "Daisy wheel". For superb-looking reports, the letter-quality printer can't be beaten. It's like a computerized electric typewriter. It was this kind of printer - the Smith-Corona TP-1 to be specific - that encouraged me to look into printers seriously. I needed a printer that would produce nice looking reports as well as have sufficient speed to print out copious data I had stored on diskettes.

TP-1

The Smith-Corona is an excellent printer, but at 12 characters per second (CPS) I felt it would take me a long time to transcribe all the notes I had stored on diskette. And, besides, most of the notes did not have to be letter-quality because they were solely for my personal use. But, occasionally I would need a machine that produced high quality printing. From the very beginning you have to think about trade-offs.

There are also letter-quality printers that have a typewriter keyboard. Besides being able to hook up these machines to your computer, you can also use them as regular typewriters. I didn't need that extra capability.

8023P

I also considered the Commodore dot matrix 8023p printer. The big advantage here is the ability of the Commodore to print all the PET's graphics. These printers can provide hard copy for your programs. But my needs did not include program listing at this time, although it would have been nice to have this feature. At this point, my programming abilities were still in the elementary stage. What I primarily scrutinized was the correspondence mode. This is the one which generates the machine's best letter-quality. With all the beautiful things this CBM printer can do, I was not very impressed with the correspondence mode. (I knew I was going to miss the program listing ability of this machine.)

During a Computer Fair recently, I had an opportunity to talk to several sales personnel about printers. I discovered there are a number of dot matrix printers that have superb correspondence quality **and** they were within the price range I had selected.

I zeroed in on two dot matrix printers: the Okidata-92 and the Epson. Both are excellent. They did all the things I wanted a printer to do and much more. But the main thing that appealed to me was the quality of the letters in the correspondence mode.

The decision was a tough one. Actually, it could have been decided with the toss of a coin. When I finally selected the Okidata, it was more from a personal preference for the style of print of the correspondence mode than anything else. Another person could have looked at the same printing output and chosen the Epson.

WHEN TO BUY?

But, the decision-making process was not over yet. Now that I was ready to buy, another question presented itself: Is this the right time to buy a printer? Are not the costs of printers coming down...just as they are for computers? Should I not wait for a while and take advantage of the new technology that is being built into printers?

For anyone who does not need a printer right away, a pause can be justified. But I had two diskettes filled with notes that I wanted to transcribe into hard copy. Any lengthy hesitation would have been disastrous. I did not want to spend long hours running off those notes I had on diskettes.

WHERE TO BUY?

Another question also arose: Where should you buy the printer? Advertisements for printers revealed a wide price range. Without hesitation I decided I would buy the printer from a local dealer. As a first time printer buyer, I knew that there would be questions about printer's use after I had purchased it.

Then there was the question of price amongst dealers. In making the rounds, I found that they are prepared to sharpen their pencils when it comes to quoting on their equipment. Competition is stiff. So when the dealer realizes that you are ready to buy, he will give you his best price. That's all you can ask for and nothing more. The dealer is entitled to a fair profit. Besides, a buyer should be prepared to share in the cost of that after-sale information that will be needed to take full advantage of the printer's potential.

Now I know that as time goes by I will need different capabilities in a printer that I may not have in the one I purchased. PET graphics and program listing may become a necessary feature. And, printers of the future will make hard copy production even more exciting than it is now.

Well, all will not be lost. I'll probably get the same feeling I did when I realized my family needed a bigger car. That's the way life goes. Besides, I might be able to sell my printer through TORPET's classified ads. After all, there may be someone out there for whom this printer would be the "ideal printer."

The way the Japanese are progressing with their new generation of computers, they may one day have the market AH SO'D up.

Ylimaki

Best of The TORPET Printers

Garage Sale Printer

By Gary Hayes, Garden Grove, CA

VIC and the C-64 make budget computing a reality. I use my VIC for business and find a printer to be a necessity. I needed a reasonably fast printer with an adjustable tractor to handle long runs of address labels. I found that the original TRS-80 line printer is in good supply on the used market at prices from $100. to $175. The unit is big, noisy, and has no lower case characters, but the adjustable width and good tractor make it an excellent lister and mail label printer. I used a cord ? (Cord print) universal parallel interface from Cardco of Wichita, Kansas, USA.

The printer is capable of providing the 5 volt supply but I chose to shorten the flat cable wire at the Centronics plug that corresponds to pin 18. The wire for pin 10 is shown in the manual as not used but is actually grounded. This resulted in a blown fuse on the printer's 5 volt supply. Shortening the wire for pin 10 solved the problem. To shorten the wire simply remove the rear cover from the connector and trim the wire back with an exacto knife. This allows the connector to be snapped back on a half inch or so further down the cable to restore the open lines if necessary for another Centronics printer. I offer this information because neither Radio Shack, The Radio Shack Printer Repair Station, The Centronics Tec Man nor Cardco could help me.

My printer is still working strong many address labels and mailings later. Most older equipment has little value in today's world of daily improvements. Now many people are trading up from those first models on trash eighties to newer things. The printers may be worthy of survival.

Selecting A Printer

By Gene Wilburn, Mississauga, Ont.

Printers are to microcomputers what speakers are to stereos. Cheap speakers and expensive speakers both reproduce the same notes, but to the ear there is a discernible difference in the quality of the sound. Likewise, a cheap printer and an expensive printer both produce the same words of your word processing text, labels, or listings, but the eye sees a noticeable difference in print quality. In general, if you want the best possible print, with no sacrifice in speed or features, you must be prepared to pay for it -- anywhere from $2000 to $4500. For a business or professional organization, with high-volume use and with a corporate image to maintain, a printer of this quality is essential. The home computer user, however, will probably have reservations about buying a printer that costs four to ten times as much as the computer itself.

Fortunately, in the world of printers, cheap does not mean bad. Just as there are inexpensive speakers that give excellent performance for their size and price, there are less expensive printers that are very satisfactory. As always with hardware purchases, the trick is to balance your wishes and needs against the fullness of your wallet.

If you are in the market for a printer, do as much homework as you can before buying. Read the printer surveys in computing magazines to see what is currently available. Don't limit yourself by considering only Commodore printers -- your computer will operate with most of the printers on the market. Be sure to do some comparison shopping. As with stereo equipment, it always pays to shop around. Don't buy the same model printer your friend has simply because you've seen it. If you like it, that's fine, but above all, determine what *you* want from a printer. Try to become as "printer literate" as possible. The following sections contain some information, points, and tips that may assist you with your purchase.

DAISY WHEEL VS. DOT MATRIX

The first decision to make is whether to buy a daisy wheel or a dot matrix printer. Daisy wheel printers are named for the shape of their interchangeable type element, which resembles a daisy, i.e., the element has a central hub from which spokes radiate daisy-fashion. On the ends of the spokes are pre-formed characters that are, literally, hammered onto the page during printing. The daisy wheel functions in the same manner as the ubiquitous IBM Selectric typewriter "golf ball" and the finished product looks as if it had been typed on a Selectric (without Whiteout tracks). Some high-quality printers, like the NEC Spinwriter, use an element that looks more like a thimble than a daisy, but the idea is the same. Daisy wheel printers are the most expensive printers for microcomputers. Some of the better-known manufacturers of daisy wheel printers are C. Itoh, Diablo, NEC, Qume, and Radio Shack.

Dot matrix printers, on the other hand, do not have pre-formed characters. Instead, a grid of pins (a matrix) is hammered onto the page. For each character of the alphabet, a different pattern is hammered, forming characters made up of little dots. You've undoubtedly seen examples of this on bills or computer printouts. There is no disputing that dot matrix printing does not look as nice as daisy wheel printing. However, if the matrix is dense enough, it can approximate the look of pre-formed characters -- a look that has been dubbed "correspondence quality" by printer manufacturers. Alas, the dot matrix printers that do a really convincing job of this tend to be nearly as expensive as high-quality daisy wheel printers.

In general, the majority of dot matrix printers are considerably cheaper than a corresponding daisy wheel model. The cost of both types of printers has been falling, and you can buy a very good dot matrix printer in the $400 to $1000 range. There are also some advantages to matrix printers. For one thing, they are often faster than daisy wheels and the better ones are much more versatile. Many matrix printers allow you to change pitch, typeface, and character width on the fly, with control sequences from the computer. With a daisy wheel printer you would have to stop printing midway on a page, manually change the print wheel, print some more, stop again, and then change it back, to achieve a similar result. Furthermore, some matrix printers can print high resolution, dot-addressable graphics -- an important feature if it tickles your fancy. Matrix printers are great for printing program listings and they tend to be relatively compact and lightweight. All in all, a good dot matrix printer is very satisfactory if you don't require IBM Selectric quality printing. Manufacturers of popular dot matrix printers include Epson, Centronics, C. Itoh, Mannesmann, NEC, Okidata, Star Micronics, Radio Shack, and Leading Edge.

There is a new breed of daisy wheel printer on the market. Some of these, like the Olivetti Praxis, are

really electronic typewriters with a printer interface; some, like the Brother and Smith Corona, are stripped-down daisy wheel printers. All are priced competitively with good dot matrix printers, and all are abominably SLOW! Ten characters per second is top speed. They've not been around long enough to have an established track record in terms of dependability, but they should be investigated by the home user who has more time than money. A printer that can double as a typewriter may be of special interest to writers.

INTERFACES

Most printers connect to either an RS-232 serial interface or to a parallel interface. Daisy wheel printers most often require a serial interface, but some allow the option of parallel. The majority of dot matrix printers connect to a Centronics parallel interface. This can be a slight problem for Commodore computer owners because Commodore, a few years back, settled on the IEEE-488 parallel interface as its standard. Consequently, to use the majority of non-Commodore printers, you must add an IEEE-488 to Centronics cable for parallel interfacing. Once this is done, you can use most of the parallel-interface printers on the market. Be aware, however, that Commodore graphics in BASIC listings will not print properly on non-Commodore printers.

SPREADSHEETS AND WORD PROCESSING

If you use a printer primarily for listing programs, the quality of the character sets is not critical, but if you intend to use your computer in professional applications, you should pay careful attention to the character sets that are offered on a printer. If you need to print out wide financial spreadsheets, for example, either buy a printer with an extra wide carriage for wide paper, or make sure that a regular-sized dot matrix printer has a 16.5 cpi (characters per inch) setting. For a daisy wheel, insist on one that will accept 15-pitch daisies. For listing programs on a daisy wheel printer, make sure you can purchase an ASCII daisy wheel. Otherwise you will be missing important symbols such as ◄and►

Many of the better daisy and matrix printers have a typeface called "proportional." Proportionally-spaced typefaces are very smart looking — the characters are fitted together more attractively than with the equally-spaced "monospace" characters of most typefaces. However, there is a "gotcha" in this. Columnar display and right justification of proportional type is tricky and requires special support from your word processing software. If you like your current software and it doesn't support proportional, then don't spend anything extra to get proportional spacing on a printer — you'll never get the benefit of it. If, however, you decide you can't live without proportional spacing, then toss out your word processing software and buy a package that fully supports it.

One last word about word processing. If you're using a word processing package that you really like, then *stick with the printers it supports*! Don't expect the software author to rewrite the program to your specifications. There are a lot of printers out there and no WP package can support them all.

OTHER CONSIDERATIONS

Printouts look best when you can see them. Many computerists, however, use their ribbons so long that their printouts look as if the ribbons were unacquainted with ink. To minimize the tendency to use ribbons too long, bear in mind the cost of ribbons when selecting a printer. Carbon ribbons for daisy wheel printers are very expensive and they don't last long. Cloth ribbons for dot matrix printers are, on the whole, more reasonable, but the ribbons for some brands are dear enough to make your wallet cry "Ouch!" If you select a popular brand of printer, chances are that you will find some good buys on ribbons from time to time. Select a rarified model and you may have to import ribbons at a premium. Simple supply and demand.

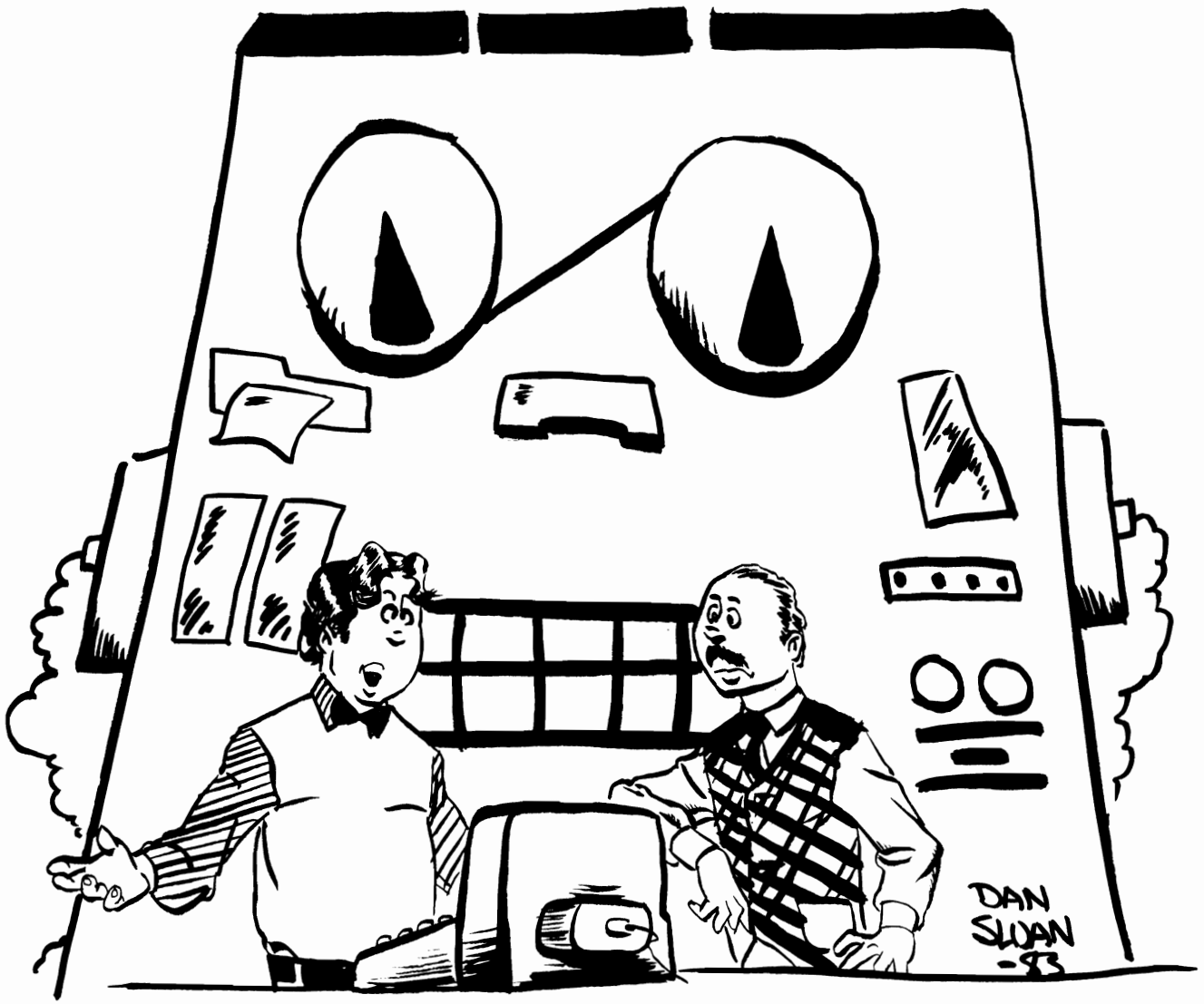
Check out how a prospective printer feeds paper. Most popular dot matrix printers use a pin-feed mechanism to accommodate standard-sized continuous-form paper — the kind with holes on the sides and perforations between the sheets. This is a desirable feature. But what if you want to feed in one sheet of letterhead stationery at a time? If this is important, make certain it also has friction feed. If you need to print on various sizes of continuous-feed stock, such as labels, then select a printer that also has available an adjustable tractor-feed mechanism. Tractors are particularly important for daisy wheel printers. Daisies jiggle around so much during printing that continuous-feed paper gets out of alignment if it is not controlled by a tractor-feed mechanism.

The speed of a printer should not be overlooked. Anything slower than 80 cps (characters per second) requires a lot of patience. Also, check out the noise a printer makes. Most daisies are quite

noisy and can create a disturbance in an office area. If a model is noisy, see if there is a silencer hood available. Dot matrix printers sound annoyingly like angry mosquitoes. Not much you can do about it except introduce a new house rule: "no printing after midnight."

CONCLUSION

Printers lack the pizzaz of modems, the challenge of joysticks, and the elegance of light pens, but, with the exception of a disk drive, a printer is the most useful peripheral you can add to your system. The next few years should bring some interesting new developments in the way of affordable ink-jet printers and perhaps even home-priced laser printers but, in the meantime, there is a good selection of reasonably-priced equipment that should meet your needs. Happy printing!



Y'KNOW, IT'S FUNNY! ALL DAY LONG WHEN I WORK, I FEEL LIKE SOMEONE'S WATCHING ME!

VIC and RS-232 Printer

By Daryl E. Williams, Santa Ana, CA

Am I the only VIC-20, RS-232 printer user? It seems that way when it comes to getting information or buying software. When you do anything in the RS-232 interface, it's on the modem only. It's time we stood up and be counted. Let's have an RS-232 information exchange on printers and disks. Write to me and let me know any of your experiences with the RS-232 interface. If there are enough, maybe we can start our own users group.

The reason I ended up with this problem is as follows. This guy I work with had a rebuilt Data Products DP-50 daisy wheel printer for sale. Only $250, but little did I know what was ahead. I bought a Quantum Data Model 1800 printer, which is an RS-232 interface connected to the user's port. Hey, this is great; now all I have to do is power up, and we are in business. **WRONG!** My first problem, it didn't line feed. Everything printed on the same line. After many failures and much research, I found the answer. I had to use OPEN 128,2 or greater. That was in May of 1982. Only you who have experienced this will know that wasn't the only problem. For you beginners, here are a couple of hints.

To list a program, use:

```
OPEN 128,2,0,CHR$(6):SMD128:LIST
```

Explanation: You must use greater than 127 for line feed, in this case 128. The 2 is the user's port device. The CHR$(6) prints in 300 baud.

To convert to standard ASCII sub-routine:

```
1000 REM Standard ASCII subroutine
1010 X = LEN(P$):IF X < 1 THEN1060
1020  FOR  I = 1TOX:X$ = LEFT$(P$,I-1):  SS$ = MID$(P$,I,1):
Z$ = RIGHT$(P$,X-I)
1030  Y = ASC(SS$):IF  Y > 64  and  Y < 91  THEN  SS$ = CHR$
(Y + 32):GOTO1050
1040 IF Y > 192ANDY < 219THENSS$ = CHR$(Y-128)
1050 P$ = X$ + SS$ + Z$:NEXT
1060 RETURN
```

Boy, would I like to have a way to change some of the VIC printer programs to print on my printer.

Write to me: Daryl E. Williams, c/o Dew-Rite Enterprises, P.O. Box 1932, Santa Ana, Ca. 92702.

VIC 1525 Printer Routine

By Michael Kelinert, Nanuet, N.Y.

Although the VIC-1515 printer has its limitations, there are several nice, built-in functions which allow the user to easily manipulate output. One very useful function is the dot addressable print positioning command. This enables the programmer to specify a specific dot address from home position at which the printer is to begin printing.

The format used is as follows:

```
PRINT CHR$(27);CHR$(16);CHR$(HI);CHR$(LO);A$
```

Here, CHR$(16) is the code for character addressing, and CHR$(27) is the code used to specify a dot address. The value for HI is the high byte of the dot address and LO is the low byte. To determine these two values, the following formula may be used, where DA is equal to the dot address:

```
HI = INT(DA/256): LO = DA-HI*256
```

The short program below uses this feature to spread out the characters in a line of text equally, thus producing an even right margin. It will work correctly as long as the inputted string does not exceed eighty characters.

This program is only a sample to demonstrate the attractive output which may be produced. The main routine is in lines 6 to 30, and may be incorporated into other programs, such as a word processor, to produce very nice results.

```
5 INPUT A$: OPEN 4,4: CMD4
6 IF RIGHT$(A$,1) = " " THEN A$ = LEFT$(A$,LEN(A$)-1):
GOTO 6
10 A = LEN(A$): B = (474/(A)-1)
20 FOR C = 0 TO A-1: D = C*B: D = INT(D/256): E = C*B-D*256
30 PRINT CHR$(15);CHR$(27);CHR$(16);CHR$(D);CHR$(E);
   MID$(A$,C-1,1);: NEXT C
40 PRINT#4: CLOSE 4
```

Re-Inking Printer Ribbons

By T.J. Bos, Brantford, Ont.

Printers such as the 8023p and others do use ribbons that are contained in a cartridge.

The printed letters become gradually less dark until a point is reached where it becomes necessary to replace the ribbon.

The cartridges are fairly expensive, so I will describe the steps required to re-ink the ribbon:

1. Remove the cartridge from the printer.

2. Select a screwdriver with a blade width equal to width of the horizontal slots in the vertical sides of the cartridge body.

3. Insert screwdriver blade into one of the five slots and twist blade to pry up the cover approximately 1mm (about 0.04 inch).

4. Repeat step 3 for the other four slots.

5. Position screwdriver or knife blade on one side under the cover between slot and 'nose', and gently slide it towards the 'nose' of the cartridge.

6. Repeat step 5 for the other side.

7. Go to step 3 if cover is not loose.

8. Do not remove or disturb the ribbon.

9. If you ignore step 8, you'll be sorry.

10. Apply with a little brush a few drops of the re-inking fluid. As a guide, you may try about five drops; you can always add more if needed.

11. Make sure that the ribbon is still down in its original position, especially along the edges, so that the cover cannot pinch it.

12. Re-assemble the cover on the cartridge body all the way back down to its original position.

13. Inspect the aluminum-colored mask. If it is badly worn, you may attach over top of it a new layer of thin metal or plastic tape that has a pre-cut hole in it to clear the needles of the print head. The thickness of the tape may reduce the number of copies that your printer can handle.

14. Re-assemble the cartridge in the printer and turn the knob on the top in the direction of the arrow to take up slack of the ribbon.
Best results are obtained when the ink has a few hours' time to spread evenly through the ribbon.

The re-inking can be repeated several times.

I made the medium for re-inking by first concentrating the contents of a bottle of black stamp-pad ink over slow heat to half its original volume. Then I added glycerine to restore the original volume.

(I heated the stamp-pad ink in a cap of a spray can on a wire stand over a candle.)

Best of The TORPET Printers

Re-Inking Printer Ribbons

By E. Toussaint, Shelley, W. Australia

Here is an alternative method which I use with our Commodore Tractor Printer 8023P:

1. Set a workshop drill press to ultra-low speed.

2. Lightly clasp the advance knob of the printer ribbon cassette in the 3-jaw chuck of the drill and raise the drill's working table to just support the printer ribbon cassette.

3. As a quick trial, turn on the drill while holding the printer ribbon cassette loosely (i.e. so that it can be easily let go if the ribbon sticks). The ribbon (which forms an "endless loop") should advance slowly and smoothly through the cassette.

4. With a stiff brush, keep the thick black printing paste from a "Gestetner" duplicating machine in contact with the ribbon as it advances around the cassette. Continue this until all the ribbon has been coated with printing paste on both sides.

5. Replace the ribbon cassette in the printer. The first few copies will be a bit messy, but after this the print should be dark and sharp.

Note: If you don't have access to a drill press, the above operation could be done with a hand drill. If this is the case, it then becomes a 2-man operation.

The 1526 — Good Printer, Lousy Manual

By Howard M. Mesick, Hartly, DE

I'm very happy with my new Commodore 1526 printer. It hooks up directly to my VIC (or your 64) through a 3½-foot (too short) cable, without need for an expensive interface. While it doesn't have the print enhancement options of the Epsons, it doesn't need them. Its regular 8x8 matrix is very sharp and pleasing. One problem: the nines look almost like eights. Only the OKI 92 and the IDS 480 do better, for 50% more at a discount, plus interface.

Nominally a pica machine (10 characters/inch), my 1526 squeezes 32 letters and spaces into 3 inches, close enough. Double- and quad-wide characters also print. A platten allows the use of single sheets, while the tractor width is adjustable, unlike some more expensive friction/pinfeed competitors.

Great device, great price!

Only one component is clearly defective, part #983001810 — the 'friendly' user's guide. Lucky for you that aging pot-bellied boy geniuses like me are around to fill in the instructions that Commodore left out.

SET-UP SNAGS

The first major omission is in the section, 'Preparing to Use Your Printer'. While mention is made that you should take out any 'foreign material that may have fallen into the mechanism', you are not told to remove the tape holding down the print head or the little plastic block next to the head. Please do so. On the bottom of the 1526 are two Phillips bolts labelled shipping screws. Though the manual didn't even hint at their existence, I took them off. You may leave them in at your own risk. Having assembled umpteen mechanical and electronic devices, I have never met a shipping screw, bolt, brace or gizmo that wasn't meant to be removed before operation.

In the same prep section is this marvellous obfuscation.

'Hold the ribbon cartridge with the plastic knob at the top left side, then set it on the two side frames of the printer mechanism, tilting the cartridge so the two front hooks on the side frames will be engaged with two catches on both the left and right sides of bottom of cartridge, then steer down while pressing the ribbon side so the two side tabs of cartridge are positioned into the slots on the side frames of the printer mechanism.'

Whew!

The illustrations show pretty well how to install the ribbon, which looks very much like an Epson MX-80 cartridge. I've heard that Epson makes the 1526. Just be careful, when loading the cartridge, that you slip it far enough forward, away from the print head, so that only the ribbon itself is near the platten. Then, if it is properly centered right to left, it should snap in. Never force it, but try to match tabs on the cartridge with niches in the chassis.

Oh yes. Swing the paper bail up out of the way first, or you will snag the ribbon. The bail is that little rod with two rubber rollers on it on the top front of the platten. The platten is the large black roller around which the paper goes. But you already knew that, right?

The Commodore guessbook shows how to install the wire paper holder, but doesn't mention that you must route incoming continuous forms UNDERNEATH it, so that printed sheets may accumulate on top of it. Never use this rack to hold a box of feed paper. That will bend it or pull the printer off the table.

Loading the paper isn't really explained, but should be obvious to anyone who has used a typewriter. It's easier to use the knob on the side of the housing to advance the paper than the paper feed button that Commodore recommends.

One important control not described or shown is the pressure release lever just to the left of the platten. (I always speak as though you were facing the front of the printer.) When pulled toward you, it releases the pressure rollers beneath the platten, allowing the paper to slide around somewhat. This looseness enables you to adjust a single sheet of paper so that it is perfectly straight. Then you can push the lever back to lock the paper tight so it won't shift during printing. Always keep the lever forward when printing on continuous forms. If you don't, slight speed differences between the tractor and the tight platten will eventually tear, crinkle or skew your paper. A resulting paper jam could damage the 1526.

Both tractors are very hard to slide to adjust for paper width. The left pinwheel moves only about ½ inch right and left. Don't try to move it farther. The right pinwheel will slide left past the centre of

the platten. Hold the rod on which they are mounted when you adjust them, and apply force as gently as possible to ease strain on the chassis.

PROGRAMMING PITFALLS

If you don't know BASIC, you probably won't want to learn it by programming this device. If you do know it, you'll probably muddle through. Explanations of the various commands aren't very clear. Many vital steps are shown in the examples, but never covered in the narrative. In some cases, questions are answered further along so, when in doubt, keep reading. A VIC or 64 programmer's reference manual can sometimes clear up the murk. Often, you'll just have to play around to see for yourself how a command works.

The format control commands, being peculiar to this smart peripheral, are the only ones not explained in other books. These are sent to the 1526 through a third and optional parameter of the OPEN command called the 'secondary address' or 'SA'. SA's regulate the printing of dollar signs, leading zeros, PET graphics, user-defined characters, lines per page, line spacing, diagnostic messages, etc.

The manual says 11 SA's are available, then lists only 10, 0 through 9. That list is wrong. There is no #8. 8 should be 9, and 9 should be 10. Cut and paste marks indicate that an 11th control code once existed, but was removed. In the individual examples, the two last SA's are correctly numbered 9 and 10. Does this deletion have anything to do with the recall of this printer after it was released last June?

The 1526's on-board RAM can hold only one user-defined character at a time, printing it when CHR$(254) is sent by the program. Yet any shape that can be created in an 8x8 dot matrix can be placed in that RAM by sending a character string variable containing the shape. Thus, an entire character set can be held in the computer's memory. The characters must be sent one at a time, then printed in a following instruction. To do that, the 1526 must be OPENed as two different files — one to send the special characters to printer RAM, and another to print them and everything else. Important: only one user-defined character can be printed on a line, though it can be repeated on that line many times. Use of PET graphics is unlimited.

Control code 10 is the printer reset. It is not explained at all! According to common sense and my own experiments, it works the same as turning the machine OFF then ON again. It wipes out all format instructions and returns the print head to the far left. It's easier on the electronics to use SA10 than the power switch.

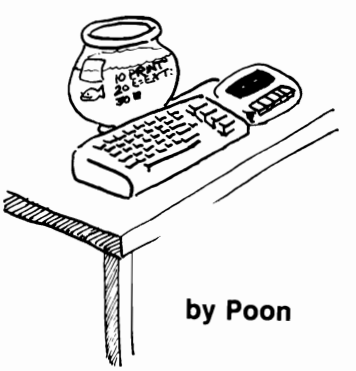
Instead of the skip (blank) character, CHR$(29), spaces enclosed in quotes usually work. A semicolon does the same as in a screen print command, while a comma inserts 11 blanks. SPC works if it isn't the first item in a print command. TAB acts like SPC instead of the screen TAB.

When setting the number of lines per page, do not include the six lines that the 1526 automatically skips at each page break. If you want 33 lines/page (double spacing on 11" paper), for example, code in 27. To set the number of lines per inch, the user's guide says to divide 144 by the number of lines desired and plug the quotient into the command. WRONG! Divide the lines/inch figure into 216. Then use that quotient. Example: to get 3 LPI (double spacing), 216/3 = 72. Plug in 72.

HASTY CONCLUSIONS

Commodore has made its fortune by mass-producing state-of-the-art equipment dirt cheap. If it ever does release its new business machines, it should thin out the flock of stale turkeys that sell for over $50 a pound because of three silly initials on their faceplates. Even though the Fortune 500 boys buy prestige rather than value, lots of small businessmen will appreciate reasonable prices. But big, billion-dollar-a-year Commodore will have to stop putting out manuals that look like preliminary documentation from upstart companies that can't afford to do it right. It could at least hire some smart Alec like me to tell it what non-technical users might not understand.

Keep this article around, Kiddies. If you ever purchase Commodore's little best buy, the 1526, you'll need it.



by Poon

Data Base

PAGE

What is Data Base 158
G.R. Walter, Proton Station, Ont.
Here are the answers to just what a Data Base program can do for you.

A Helping Hand 159
D. Howell, Cambridge, Ont.
A teacher explains how he finds a data base program useful in a school setting.

Cheap Data Base 162
Hank Mrockowski, Houston, Tex.
The Hardware Hacker explains how you can have a free data base program. You can't get much cheaper than that.

Magic Desk 163
This is Commodore's answer to LISA. It is not a data base manager. It isn't a spreadsheet. I really didn't know quite where to put it. It tries to be everything to everyone.

What is Data Base?

By G.R. Walter, Proton Station, Ont.

DATABASES — WHAT EXACTLY ARE THEY?

A database program is a program which helps you, the user, organize and manage facts and information. It usually simplifies list keeping and the constant updating that some lists require. It can be as simple or as complex to use as you make it to be.

DATABASES — WHAT WOULD I USE ONE FOR?

A database program can be used for many things. You could keep mailing lists of relatives in a database (eg. for Christmas cards) and have it print out all the address labels for you. A library could put all of their book files (what information would normally go into a card catalog) into a database to simplify keeping track of what person has which books. A doctor could keep all of his patients' files in a database instead of a large unwieldy filing cabinet. A homeowner could use one to keep track of his possessions. Clubs could use one to keep track of their members ... (Hey! — maybe they've been doing that all along!) ... The uses of a good database program are endless. Wherever you are keeping a file or a list of things, a database program could be helping you.

DATABASES — WHY WOULD I WANT TO USE ONE?

You would want to use a database program because with a good one :

a) it is much simpler to add records to your file and to edit the records already in your file (if you make a mistake you can go back and fix it right away — no fuss, no muss).

b) you can easily and quickly SORT the data into alphabetical order (an act that could take nearly forever if you have a filing cabinet full of records that have to be sorted).

c) you can find a record that you want to look at (patient/recipe/member, or whatever) much faster using the database program because you can tell it to find the record for you (ie. find record of patient : Doe, John).

d) you can make a backup of your data file and put it someplace safe so if some disaster (eg. a fire) occurs you won't have lost your precious data files.

e) you can get a printout of your data file whenever you need one (no more endless photocopying sessions, or re-typing sessions needed).

f) you can oftentimes get a printout of address labels directly from the database program (ie. most of the database programs will have a special "mailing address label" function).

These are the most important reasons for getting a database program. Together they all add up to the one main reason: it is more convenient to use database program to work with your data/lists than to do it manually.

DATABASES — SOME PROBLEMS GETTING STARTED

There are three main problems that one can come across when getting started with a database program :

1. Which database program do you buy? There are a great many database programs on the market and you have to choose the one which best suits your needs. The advanced programmer can write one that will meet his requirements, but the average computer owner will have to buy a general use database program. The best advice to be given is this: decide what you need in a database; then read the ads in computer magazines, any reviews you can get a hold of. If you know any other computer users, ask them what program they use, and what they think about it. As in any purchase, find out all you can about what is available before making your choice.

2. Learning to use your database program. It usually takes a while to familiarize yourself with even the simplest database program, and the more functions that one offers, the longer it takes for you to start using the program.

3. Getting the initial file set up and typed into the database program. This is usually the major step, because it can involve a lot of work if the files are large (if you have a busy business, you may not be able to spare the time or effort to convert).

Leaving the best until last — there is actually one reason for you not to get a database program. It is, simply put, you have never kept, nor ever will keep, a file or list of data in your life. As a result, a database program would be totally unnecessary for you and getting one would be a waste of money.

A Helping Hand

By D. Howell, Cambridge, Ont.

During our careers as teachers, we are required to create many lists and reports. We have heard that, someday, all this will be done by computer. For much of the work we have to do, TOMORROW is here.

IF:
1. You wish you had some freedom from paperwork;

2. You know your letters from your numbers;

3. Your school has one microcomputer and a (single) disk drive;

4. You are willing to spend some time;

you are already well on your way to becoming a **DATA BASE USER**.

WHAT IS A DATA BASE?

Put very simply, a data base is an electronic filing cabinet which stores information of any kind in any format. It offers a flexibility in record-keeping never before possible at a speed never imagined. What's more, the only computer knowledge you need is how to type. The ORACLE and MANAGER programs I have seen are menu-driven, which means that, to do this job, you push this key. It does take some time to get used to operating the program, but after that the possibilities of what you can do are endless.

Computers are fast, but they are not smart. A computer relies on the operator for **all** of its knowledge. It cannot give you back anything it was not told. Once it records information, it can search and sort at incredible speed.

Next then, you have to decide all the things that you will ever want to include in your data base, and input them. Yes, this does take a lot of time, no question about it. But the beauty of it all is that **you only have to do it once**.

For example, enter a student's information the day he registers in your school, and your job is finished for the time he stays. After that, the computer gives it all back to you any time of any day in any order or form you wish. All you have to do is run a search for the information you require.

You say your needs might change next year, or even next week. "If only I had included the names of the feeder schools in my records." Relax. You can save all your information in a list called a sequential file, change your format and reload it again the new way. This was done at Stewart Ave. when our needs changed. Total operator time to make the change was five minutes. The remaining 20 minutes to create the sequence and reload the 615 names, addresses, classes, sexes, phone numbers, birth dates, class names, class types, destinations and remedial subjects taken, was done by computer.

The possibilities of the use of this information are limited only by your imagination. Let me recount how it has been used at Stewart Avenue School, Cambridge.

Our school is a composite school, Kindergarten to Grade 8. Beginning June of 1983, we began our year-end activities which include the promotion meetings. At the conclusion of the meetings, the new class placements were put in the grade section of the students' records. Then, it was a matter of sorting the names by the new class numbers to produce next year's lists. Operator time was about an hour. The computer did the rest.

At the end of the year, we are asked to submit a ranked list of students from highest to lowest by average. In years past, tired eyes have been known to miss a name by accident, which meant re-writing the list. This time, we asked for a high-to-low sort based on the average. The data base ORACLE, which we use, cannot do this sort. Instead, the list was read to the word processor PAPERCLIP and performed there. The sort took only half a second, and the printing about 30 seconds (with NO mistakes).

Note: Not every data base can read every sequential file.

With the names of the students stored in ORACLE, printing the monthly class attendance sheets became an easy task of inserting the sheets into the printer and sorting by class in alphabetical order. These files were also read to PAPERCLIP, which permitted faster output, since ORACLE didn't have to search every record to find every member of every class each time we wanted the list. This process would be even more efficient with continuous form paper, since the operator wouldn't have to stand over the printer to continually change each separate sheet.

All elementary principals must do the Age-Grade-Sex report at the end of September. This involves counting the number of seven-year-old males, etc., as of the first of October. ORACLE allows for the storage of your information in many forms simultaneously. One that is useful in this report is the "sort by age" (key) file. It can be arranged to give the total number of bithdays for any given month in order of birthdate in minutes.

Our Senior Support (remedial) Program is done on a withdrawal basis. The teacher in this section keeps the rest of us up-to-date on who is receiving help in what areas in a written report every term. This fall, the report was done wih the aid of the data base. A Senior Support section was added to the existing records, and the records were sorted by Sr. Support, grade and class. A ditto was inserted into the printer, and the reports were run off and handed out.

There are many other possible uses for the data base within the school.

LIBRARIES

The current amount of work to maintain a library can be substantially reduced with the aid of a data base. No longer is it necessary to type subject, author and title cards for every book. The librarian has only to input the information about the book (including who borrowed the book and the due date) in the data base once. Then (s)he can have a daily master overdue sheet printed out, as well as using the same information to print overdue notices. (Unfortunately, someone still has to deliver them.)

INVENTORY CONTROL

This is being used centrally for many things. It can also be used to keep up-to-date repair records of AV equipment, shop tools, appliances in Family Studies, and band instruments.

CLUB/TEAM LISTS

A club/team item could be added to the existing records and allow a list of any club or team to be withdrawn from the data base without having to type it.

MARK PROGRAMS

There are many teachers currently using marks programs to record and calculate averages. A data base could supply the names for these programs without having to type them if, in fact, the program was able to read the same sequential files as are contained in the data base. It is indeed unfortunate that, in spite of the fact that there are many fine marks programs available, none that I have seen is capable of reading sequences. Hopefully, this will change soon.

REPORT CARDS

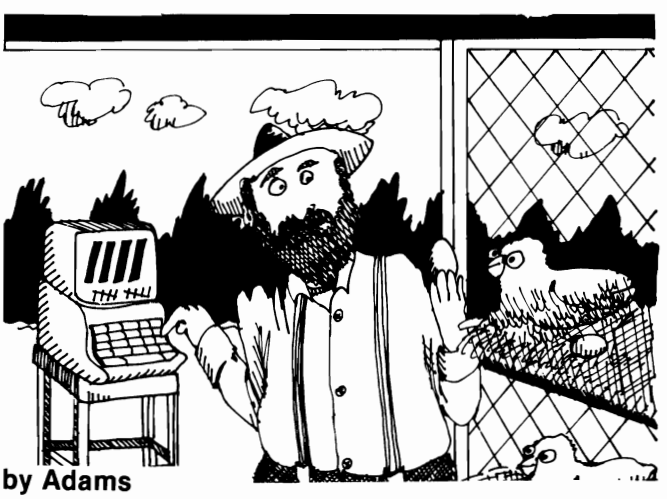
Anecdotal reports are a fact of life in elementary schools. It is often necessary to write out rough comments, have them checked, and then have them either typed or re-written. All of the information could be put into the data base in its final form once only. From here, it can be checked and revisions made without re-writing. Then it is a case of putting the report form in the printer for final typing.

This system is not without its problems, however. It requires that:

1. There be enough computers and proper networking (connecting) equipment to ensure that the information can be efficiently entered. Ten people at home can do 10 hours of work, but it is not practical to have nine people lined up waiting for one to finish his computer time. It only becomes practical when many machines are networked together to allow about six people to work at the same time. Only then is any time collectively saved.

2. The report card be in continuous form format, i.e., one form is connected to the next by a perforated edge. All printers have an automatic shut-off in case they run out of paper. The 8.5x11 sheet trips the shut-off switch with about two inches left at the bottom. Continuous form paper will allow the machine to keep running.

There are many data base packages currently available for micros.* The prices and features vary with each. All data base programs require a disk drive.



by Adams

Package Name	for	Compatible with Word Processor	Price
Profile	RS Model 3	no	$139.00
Profile Plus	RS Model 3	Superscripsit	$249.00
Manager 64	Commodore 64	Easywriter	$ 95.00
Manager PET	PET 4032 CBM 8032	Wordpro 3,4,5	$350.00
Oracle 64	Commodore 64	Paperclip	$159.95
Oracle PET	PET 4032 CBM 4032	Paperclip	$159.95

If we realize how much computers can do for us, they will almost certainly become as much a part of teaching in the future as the chalkboard is now. They serve, not to replace us, but to become our tireless assistants in the years to come.

I urge you to get involved with a data base soon.

FURTHER EXAMPLES

Students Sorted by Age (as in A-G-S Report)

NOTE: Ages are now required on all student forms in metric format (yymmdd), which aids computer use. While the data base can also sort alphabetically, there are problems with the month names not being in alphabetical order, making an accurate computer sort impossible.

7c Age Sort (female in order of age)

691011 f Angela
700213 f Annette
700715 f Linda
700812 f Sandy

Library Circulation Card

Title: PET Basic Author: Zamora, Zoltan Call #: 999.111
Subject: Computers
Borrower: B. Rubble Class: 10z
Due Date: 831215

By sorting the list of circulated books according to due date, the librarian can have at his disposal a complete list of overdue books in order, thus allowing him to see at a glance who is the "most guilty party of the day".

Overdue Books for: Jan. 4, 1984

Grade	Name	Title	Author	Due Date
12f	Haha, Minny	Jokes and More Jokes	Funny, I.M.	830901
10z	B. Rubble	Pet Basic	Zamora, Z.	831013
11c	J. Hayden	Man and his Music	L. Beethoven	831031
09d	F. Flintstone	Call of the Wild	Trudeau, P.E.	831204
13e	F. Johnston	Age of Computer Literacy	Noonan, P.	831204
13a	L. Beethoven	Thayer's Life of Beethoven	Forbes, R.	840106

Total Number of Overdue Books is: 6

The same information can be used to print out overdue notices. This particular list is sorted by class for easier distribution.

Overdue Notices:

TO: F. Flintstone 09d
Your Book: Call of the Wild
By: Trudeau, P.E. Call #: 654.321
Was due: 831204
PLEASE REPORT TO THE LIBRARY TO-DAY!

TO: B. Rubble 10z
Your Book: Pet Basic
By: Zamora, Z. Call #: 777.444
Was Due: 831013
PLEASE REPORT TO THE LIBRARY TO-DAY!

Note: None of the above information was typed into the wordpro to create the article. It was all read from the Data Base via a sequence in the format seen here.

***Note:** In the Waterloo County Board of Education, PET, CBM and RS Model 3 are the most frequently-used machines.

Cheap Data Base?

By Hank Mroczkowski, Houston, Tex.

CHEAP?

For any data base program and by any standards...sure! I'm talking about using the operating system of any PET/CBM/VIC/64 Microsoft BASIC as a "free form" data base (management) system. The full screen editing in our beloved Commodore BASIC lets us put full screen editing in memory provided we follow a few minor restrictions.

1. **Line numbers.** Every record must be preceded by a line number recognized by BASIC.

2. **First character.** The first character of a record cannot be a number.

3. **Length.** Any record cannot be longer than 80 characters (88 in the VIC).

Now, you will need some form of programmer's aid, either BUTI or Commodore's Programmers' Aid cartridge (for the VIC) to manipulate your data. It is a data base (management) system only in that YOU are the manager with help, or should I say FIND. That's the secret.*

I have been using this system on the PET 4032 for about four years with cassettes and with disks and have had no problems entering or retrieving information. The FIND command will bring out the information, searching through up to 32K in less time than the PET could print to the screen. I had used the Programmers' Toolkit and SM-KIT on the PET until BASIC AID became available (available from CHUG on chips for PETs-ED). All three gave excellent results. About a year after using this system to hold a list of telephone numbers of people scattered about the country, someone had an article published explaining this same system. The name of the article may have been *The Poormans Data Base*, but I can't find the proper reference. If anyone remembers where it is, I would appreciate some feedback to credit the written source.

BASIC AID is the most complete package of expanded keywords for BASIC but has one small bug when using it for the DB(M)S. When printing a list of records to the printer, it will close the file. This causes some strange problems on the IEEE-488 buss when a CMD command is given and some form of PRINT # command isn't issued prior to closing. The screen goes nuts.

Now that I'm using the VIC 20 more and more, I use the same style of DB(M)S on it, too. The only real limitation which has caused any problems in all the machines is the memory size of that machine. With 32K on the PET, I am limited to about 450 records or line numbers. On the VIC 20 with a *16K Expander* and Programmers' Aid, the limit is about 185 entries. Each of those records is at least 60 characters long. No absolute figures can be given to let you know if your data fits the machine because of the method the PET/VIC stores your data.

**Use either WEDGE-64 or TINY AID for 64 or VIC AID4.REL for VIC, all on Best Programs Diskette.*

(Reprint from CHUG)

For example, if you save:

```
10 JOHN DOE: 1234 S. MAIN:ERIE:PA: :16501:814-454-5278
11 JOHN REMBAR:1234 SPRINT LN:ERIE:PA:16500:
   814-555-1212:REM & (PRINT) = TOKENS
```

You'll note that line number 11 has two groups of characters which are not enclosed with quotation marks and will be stored into memory by the interpreter as BASIC tokens...single byte characters which will always print it back to you just as you typed it in. Line 11 **looks** like it's six bytes longer than line 10, but, in fact, it is exactly the same size in memory. Use this to help conserve memory. Be careful to avoid using question marks...they will be printed back as PRINT.

HELPFUL HINTS

A helpful hint, use quotation marks sparingly. They use extra bytes and make finding anything inside their brace very difficult with any prog aid program. You tell the prog aid to find either a string or sequence of characters in memory. If you quote a sequence of characters, you have defined that sequence as a string and the prog aid may not find that string if you didn't tell it to find a string.

In other words, you can only search for either a string or a sequence of characters (which are stored as BASIC) at any one time. If you store your data both ways, you will have to **FIND** in the manner in which you have stored it. This may make retrieval cumbersome. On the other hand, it can be used to separate similar data, and, with tricky techniques, sub-grouping data is possible.

For example, you may want to describe a part on inventory. By putting the part number and the description inside a brace of quotes, it won't be normally listed in a FIND of that part number. However, when the same part number is quoted during a FIND it will list its description. Neat, huh?

About a year ago, I had implemented this system where I work to keep track of the radios we repair and store. It has been very successful in saving from one half to two man-hours per day of both the service department secretary's and technicians' time whenever any customer called on the status of his unit. An added plus is the professional timeliness and attitude that the customer sees.

This same system has been used by several other people who had seen how easy it was to set-up and maintain. I had trained our parts manager and our secretary on the basics of using it in about two hours broken into short 10 to 20 minute sessions. The biggest area of concern to me is keeping the data format consistent. We identified one style of radio as *WHAT CL* for a White, Classic, GE Mark V radio-telephone. If the entry is made as *CLASS*, obviously, a match to FIND all WHT CL would miss that one. No matter how the data is entered, it must be consistent. Remember, no error checking is involved.

Inventory, back order lists, telephone directories, customer lists, etc. are all prime candidates for the Data Base (Management) System when you want to work both under a low budget and absolutely need a fast search.

I might mention an extra benefit with the system. We would keep the data on the radio long after it had been returned to our customer. This started a service history of all units which had gone through our shop. This allowed us to identify lost radios by serial number and, also, to give the customer a printed list of his units which had gone through the service department. Since the invoice number is part of the record, we can easily pull together a file to cross check for any units which are failing often.

Magic Desk

Commodore Computer's new MAGIC DESK software series is an entirely new direction for home computer software.

MAGIC DESK is unique for the home market. It produces an animated, full-colour desk on your television screen. There is a typewriter, index file, telephone, calculator and financial journal on the desk, and a wastebasket under it. There is also an artist's easel and a vertical filing cabinet with a digital clock on top of it.

To use any feature of the MAGIC DESK, you can use a joystick, trackball or mouse to move a pointing finger to one of the objects on the screen.

The first package in the MAGIC DESK series is called "MAGIC DESK I - Type and File" and comes in plug-in cartridge for the Commodore 64. The cartridge activates the typewriter, index file and related editing and filing features of the animated desk. Future packages will provide calculating and budget capabilities, artistic and educational applications.

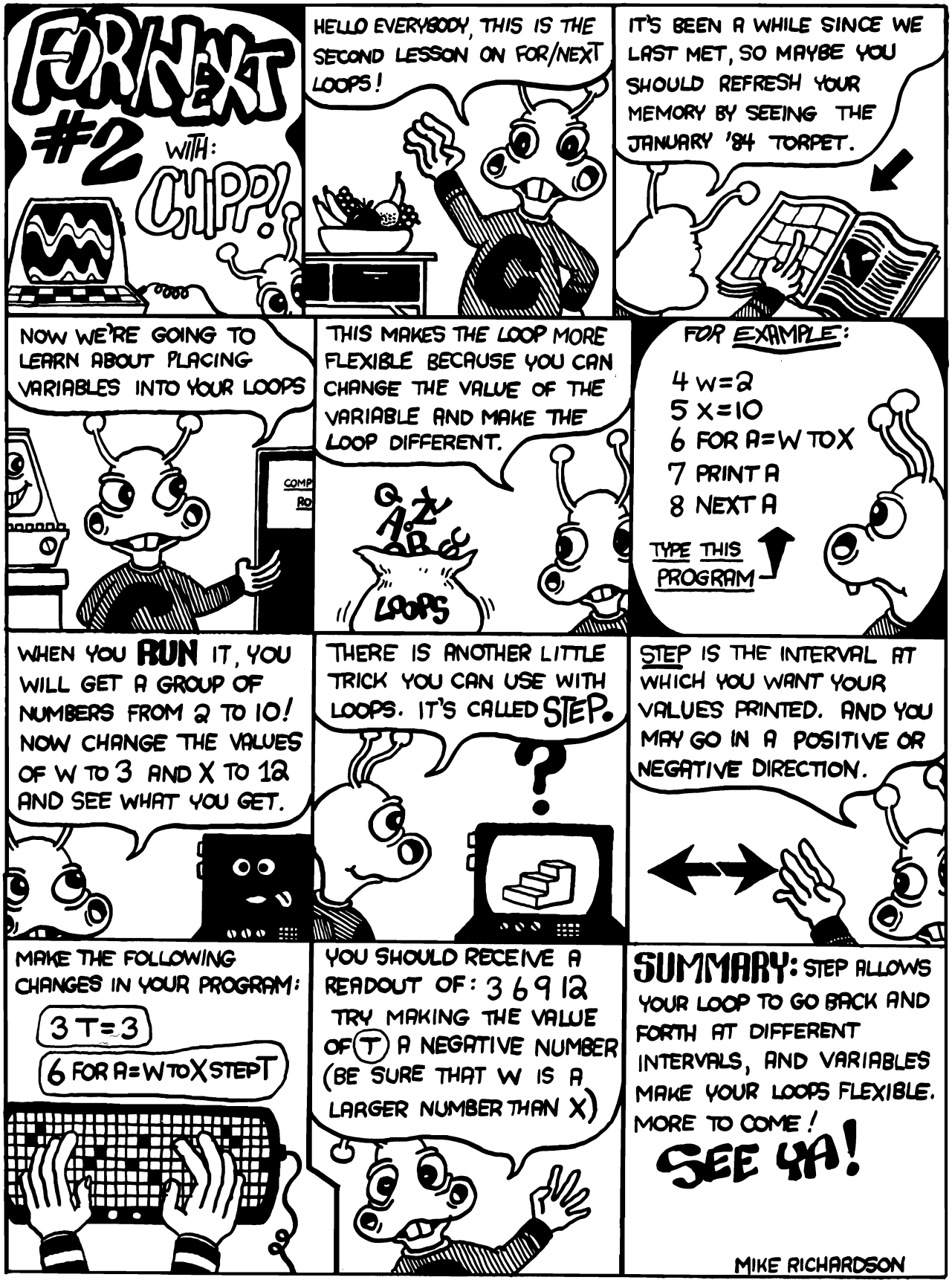
MAGIC DESK is a truly multi-national software package because it uses no language instructions. All instructions and menus are pictorial, using symbols which Commodore calls "metaphors".

It doesn't matter if the user speaks English, French, German or any other language, because the metaphors make it easy to use the MAGIC DESK. Computer metaphors, which are pictorial symbols representing specific computer functions, control all aspects of the MAGIC DESK program, from selection of features to individual menu items. International symbols have been used wherever appropriate.

An example of a MAGIC DESK metaphor is the picture of a scrolled sheet of paper which appears at the bottom of the screen when you're using the typewriter. After having typed a page you can move the pointing finger to the scrolled sheet, pick it up and move it to the file cabinet. There are three drawers, with 10 files in each drawer and 10 pages in each file. You can give the files any titles you wish, and move the pages you've typed from one file to another, or copy the pages into several files. All of the pictorial files you see on the screen are linked to a Commodore floppy disk drive, which actually stores the information. You can print out the information on a printer just as easily.

The next MAGIC DESK cartridge will include calculating and home budgeting functions.

The Commodore 64's sprite graphics allow the programmer to re-define the objects on the desk.


FOR/NEXT #2 WITH: CHIPP!
HELLO EVERYBODY, THIS IS THE SECOND LESSON ON FOR/NEXT LOOPS!
IT'S BEEN A WHILE SINCE WE LAST MET, SO MAYBE YOU SHOULD REFRESH YOUR MEMORY BY SEEING THE JANUARY '84 TORPET.
NOW WE'RE GOING TO LEARN ABOUT PLACING VARIABLES INTO YOUR LOOPS
THIS MAKES THE LOOP MORE FLEXIBLE BECAUSE YOU CAN CHANGE THE VALUE OF THE VARIABLE AND MAKE THE LOOP DIFFERENT.
LOOPS
FOR EXAMPLE:
4 W=2
5 X=10
6 FOR A=W TO X
7 PRINT A
8 NEXT A
TYPE THIS PROGRAM
WHEN YOU RUN IT, YOU WILL GET A GROUP OF NUMBERS FROM 2 TO 10! NOW CHANGE THE VALUES OF W TO 3 AND X TO 12 AND SEE WHAT YOU GET.
THERE IS ANOTHER LITTLE TRICK YOU CAN USE WITH LOOPS. IT'S CALLED STEP.
STEP IS THE INTERVAL AT WHICH YOU WANT YOUR VALUES PRINTED. AND YOU MAY GO IN A POSITIVE OR NEGATIVE DIRECTION.
MAKE THE FOLLOWING CHANGES IN YOUR PROGRAM:
3 T=3
6 FOR A=W TO X STEP T
YOU SHOULD RECEIVE A READOUT OF: 3 6 9 12
TRY MAKING THE VALUE OF (T) A NEGATIVE NUMBER (BE SURE THAT W IS A LARGER NUMBER THAN X)
SUMMARY: STEP ALLOWS YOUR LOOP TO GO BACK AND FORTH AT DIFFERENT INTERVALS, AND VARIABLES MAKE YOUR LOOPS FLEXIBLE. MORE TO COME!
SEE YA!
MIKE RICHARDSON

Spreadsheet

PAGE

What is a Spreadsheet? 166
Mel Granick, Syosset, N.Y.
Would a spreadsheet be beneficial to you? This article explains just what a spreadsheet is and how it can be used for things other than accounting.

Microsoft Multiplan Review 169
T.C. Meyer, Pickering, Ont.
Review of a commercially-available electronic worksheet for the Commodore 64.

Practicalc 64 171
John Scott, Toronto, Ont.
This is a review of one of the best spreadsheet programs available for the 64.

What is a Spreadsheet?

By Mel Granick, Syosset, N.Y.

It's been said that it's spreadsheet software that transformed the microcomputer from an electronic engineer's curiosity into the darling of accountants, financial planners and business executives.

These powerful and versatile programs enable a desktop computer, such as a PET, VIC 20 or C64, to do in minutes complex inter-related calculations that once took hours...not to mention reams of ledger paper, miles of adding machine tape, cartons of pencils with worn-down erasers, and untold frayed nerves!

And it's not just numbers that can be tabulated by spreadsheet programs. They can be just the ticket for handling data ranging from mailing lists to class schedules!

Put simply, an electronic spreadsheet program turns your computer screen into a ledger sheet with a built-in brain.

For generations, financial calculations have been done on sheets of pale green paper covered with numbers neatly arranged into ROWS and COLUMNS.

An electronic spreadsheet sets up just such a grid on the computer screen. Each location in the grid — called a CELL — can be defined by a pair of coordinates similar to the 'x/y' designations used to locate the squares on a piece of graph paper. On an electronic spreadsheet, however, one axis is given a numerical designation while the other is labelled with letters. Each cell, therefore, has a unique alpha-numeric designation...A0 defining the first row/first column, B0 the second row/first column, etc.

Figure A. shows a sample spreadsheet containing 120 cells — 20 rows (designated A-T) multiplied by six columns (defined as 0-5). Each cell on the spreadsheet is like a cubbyhole in which we can store information...either text, numeric data, or an instruction to perform a calculation. It's this ability to store instructions for the manipulation of data — rather than just the data itself — that makes electronic spreadsheets so powerful.

For example, the numbers in Column 1 of the spreadsheet shown in Figure A. represent the hourly production rate at four factories. In Column 2, we see the size of a desired production increase at each factory. Such increases would result in the new hourly production rates shown in Column 3.

But what if we revised the size of the desired production increases after we'd made our calculations?

If the spreadsheet were being done by hand on ledger paper, each of the figures in Column 3 would have to be erased, recalculated, and written back in on the page. That's not much of a job when only four new calculations are involved. But what if we were dealing with 40 factories, or 400? What if we wanted to try several sets of production increase goals to see which would be the most effective? As you can see, we'd be faced with dozens, scores or hundreds of calculations to perform.

The electronic spreadsheet reduces such a task to child's play by 'remembering' the instructions for the calculations we wish to do by way of FORMULAS stored in the cells in which we wish the results of those calculations to appear.

Figure B. shows what our electronic spreadsheet looks like to the computer. The information in Columns 1 and 2 are simply numeric values. But each cell in Column 3 contains the formula telling the computer what to do with those numbers.

For example, cell C3 tells our PET, VIC 20 or C64 to take the number in cell C1 (300) and add to it the result of multiplying the contents of cell C2 (15%). Therefore, if we change the numbers in C1 or C2, we AUTOMATICALLY change the result produced by the calculation done in cell C3. (Actually, we have to press a key to tell the computer to do the calculation, but let's not quibble!) The same thing happens if we change the number in any cell contained in a formula in any other cell on the spreadsheet.

Instructions on this spreadsheet include taking the SUM of a column (in cells L1 and L3), and the average of the numbers in a column (in cells L1, M1 and M3). The program can perform calculations involving the contents of any of the cells on the sheet.

Figure C. shows what happens if we want to change the proposed increases in Column 2. We simply enter the changes on the spreadsheet, tell the computer to perform the operations contained in the formulae, and come up with the revised totals.

FIGURE A.

	0	1	2	3	4	5
A	PLANT	PROD. RATE	INCREASE	NEW RATE		
B						
C	BOSTON	300	15 %	345		
D						
E	CHICAGO	200	10 %	220		
F						
G	NEW YORK	400	20 %	480		
H						
I	TORONTO	100	25 %	125		
J		----	-----	----		
K			AVG.:	1170		
L	TOTAL:	1000	17.5 %	292.5		
M	AVG.:	250				
N						
O						
P						
Q						
R						
S						
T						

FIGURE B.

	0	1	2	3	4	5
A	PLANT	PROD. RATE	INCREASE	NEW RATE		
B						
C	BOSTON	300	15 %	C1+(C1*C2)		
D						
E	CHICAGO	200	10 %	E1+(E1*E2)		
F						
G	NEW YORK	400	20 %	G1+(G1*G2)		
H						
I	TORONTO	100	25 %	I1+(I1*I2)		
J		------	------	------		
K						
L	TOTAL:	SUM(C1.I1)	AVG.:	SUM(C3.I3)		
M	AVG.:	(L1)/4	SUM(C2.I2)/4	(L3)/4		
N						
O						
P						
Q						
R						
S						
T						

Spreadsheet programs usually perform numerous mathematical functions in addition to addition, subtraction, multiplication and division. Many programs include such things as trigonometric functions and logical operators which allow determination of whether the contents of a cell are greater than, less than, or equal to the contents of another cell. Spreadsheet software often also includes the ability to sort numbers (from highest to lowest or vice versa) and alphabetize text entries. So, for example, a spreadsheet showing sales statistics could list salesmen alphabetically by name, or according to which turned in the best sales figures. Many spreadsheets also include a search function which permits the user to type in a name or figure, and have the computer find exactly where it appears on the grid.

Such sort and search features make electronic spreadsheets versatile list handlers. For example, columns of names, addresses, phone numbers, birthdays, etc., can be entered on a spreadsheet, then sorted alphabetically by name, chronologically by birth date, or numerically by telephone Area Code or postal zone. A teacher might enter the names and test grades of students on a spreadsheet to compute grade-point averages and tabulate class rankings. A retailer could use a spreadsheet to tabulate inventory with each item's stock number, description cost and selling price.

From the convenience of an alphabetical class list to the complexity of a financial forecast, the potential uses of an electronic spreadsheet are limited only by the imagination.

FIGURE C.

	0	1	2	3	4	5
A	PLANT	PROD. RATE	INCREASE	NEW RATE		
B						
C	BOSTON	300	10 %	330		
D						
E	CHICAGO	200	20 %	240		
F						
G	NEW YORK	400	5 %	420		
H						
I	TORONTO	100	30 %	130		
J						
K			AVG:			
L	TOTAL:	1000	16.25 %	1120		
M	AVG.:	250		280		
N						
O						
P						
Q						
R						
S						
T						

ABOUT THE AUTHOR

Mel Granick has been in commercial broadcasting, mostly in New York City and its suburbs, for better than 15 years, the last 11 of them as a newswriter, producer and reporter for WCBS NewsRadio 88, where he is currently Assistant News Director.

Microsoft Multiplan Review

By T.C. Meyer, Pickering, Ont.

Microsoft Multiplan Version 1.06 for the Commodore 64 is an electronic worksheet developed by Microsoft Corp. and marketed by Hesware, Human Engineered Software, 150 North Hill Drive, Brisbane, CA 94005, 415-468-4111.

Electronic worksheets like Multiplan basically provide for the programming of accounting, modelling or planning type spreadsheets via entering high-level programming commands into specific cells of an electronic matrix on a microcomputer. Multiplan provides a matrix of 63 columns and 255 rows. The screen displays only a small portion of the actual worksheet available. In effect, the screen is a window which can be scrolled both horizontally and vertically so that the entire matrix can be viewed as a series of windows.

If you have used other worksheets, you will find that Multiplan for the C64 is loaded with features and options not usually found in worksheets costing less than $130. Canadian. In fact, the C64 version is functionally similar and offers most of the features of Multiplan which runs on the IBM-PC. In this review, I won't address the numerous mathematical functions available in Multiplan other than to say that they are extensive.

The initial release of Multiplan which I purchased in November of 1983 was an excellent product; however, it supported only the VIC-1525 printer. Printing worksheets on my Gemini-10X printer with Card? printer interface presented a problem. All alpha characters were printed in lower case, even those which were in upper case on the screen. The details of this problem were forwarded to Hesware who corrected the problem and then, in January 84, shipped to me a revised release of Multiplan which supports a wider range of printers as well as several other updated features.

PRINTING CAPABILITY

The printing capability of the revised release of Multiplan is extremely powerful and flexible even with non-Commodore printers. For example, specific rows, columns, cells or the entire worksheet can be printed in a variety of print pitches, including normal and compressed print, i.e., 17 characters per inch. The margins, at the top, bottom, left and right sides can be set to control the print-out line length and location as desired, thus making full use of the Star Gemini-10X or Gemini-15X printer features. This worksheet also has extremely versatile features for formatting text and values in the rows and columns of the worksheet. Some of these features are: text centering, left and right justification of columns, per cent format, dollar sign format, integer format, and fixed format with a specified number of decimal places. Once the formats for a worksheet are entered, these will appear on both the screen as well as the printout of the worksheet. Multiplan also supports the printing of all the formulae which you have programmed for a specific worksheet. This is a valuable feature, in that it provides you with a hard copy of your worksheet logic for review and future reference. In addition, when a worksheet is saved to disk, the printing options and formats which have been set up are saved on the data file as well. Thus, the worksheet can be easily printed in exactly the same format later on.

To run Multiplan on the C64, a 1541 or compatible type of disk drive is required as tape input/output is not supported. This is not surprising, since the 1541 disk drive requires about two minutes to initially load the multiplan programs. Also, the initial release of Multiplan requires that the program disk remain in the disk drive to support many of the Multiplan functions. This makes saving and loading of Multiplan data files a series of swapping exercises between the program disk and the data file disk. The revised release of Multiplan solves this problem by requiring that a 97-sector support file be written to each data disk upon initialization. The data disk is placed in the disk drive after Multiplan is loaded, and therefore eliminates the need for the disk swapping. Even so, writing a data file to disk is a slow process, partly because the worksheet in some cases is automatically re-calculated as the first step of the saving operation. It takes about two minutes to write a 24-sector relative file while retrieving it takes about one minute.

When using the revised version of Multiplan, the data disk with the program support file on it is automatically referenced by Multiplan to swap into memory the necessary routines to process each new worksheet function which you have entered from the keyboard. Because of the memory-conserving program swapping features of Multiplan, a fairly large worksheet can be contained in memory. For example, the 24-sector data file mentioned earlier was comprised of a worksheet of 10 columns and 31 rows of data, including formulae involving calculations on 210 cells. This worksheet used up about 25% of the

C64 memory which is available for worksheet space. The original version works in a similar manner, except the Multiplan disk must be in the disk drive.

SORTING FEATURE

A feature of Multiplan that puts it ahead of some other spreadsheets is its ability to sort on columns of information. Numbers and text can be sorted in ascending or descending order. Multiple columns of a worksheet can be sorted in a 'sort key' fashion by sorting in the order of the least significant to the most significant columns. By using this technique, a sorted report can be easily generated.

Another excellent feature of this worksheet is the capability to name cells or groups of cells, so that you can refer to them easily. Whole rows or columns or a specific cell can be assigned a name for future reference. For example, a given row can be named 'Quantity', and hence any cell in that row can be referenced in formulae by simply using the designated name. Therefore, formulae can be set up such as 'Cost = Quantity*Price', where the name feature has been used to reference the cells corresponding to the specific names. Like most other worksheets, the direct cell reference modes can be used, but this results in less intelligible formulae such as R8C6 = R5C2*R7C4.

Even a more significant feature related to the name function is the ability to use multiple worksheets. This function, used in conjunction with a special External copy function, provides the ability to relate worksheets to each other. While the specific details of this can become fairly complex to initially set up, once it is completed, Multiplan will automatically keep track of changes affecting both dependent and supporting worksheets. This is a valuable feature because it allows, for example, to relate 'Quantity' from a January worksheet to 'YTD Quantity' of a February worksheet. When the dependent worksheet, February, is loaded from disk, the interrelated information from the supporting worksheet, January, will automatically appear on the screen of the February worksheet in the 'YTD Quantity' area which had been previously specified.

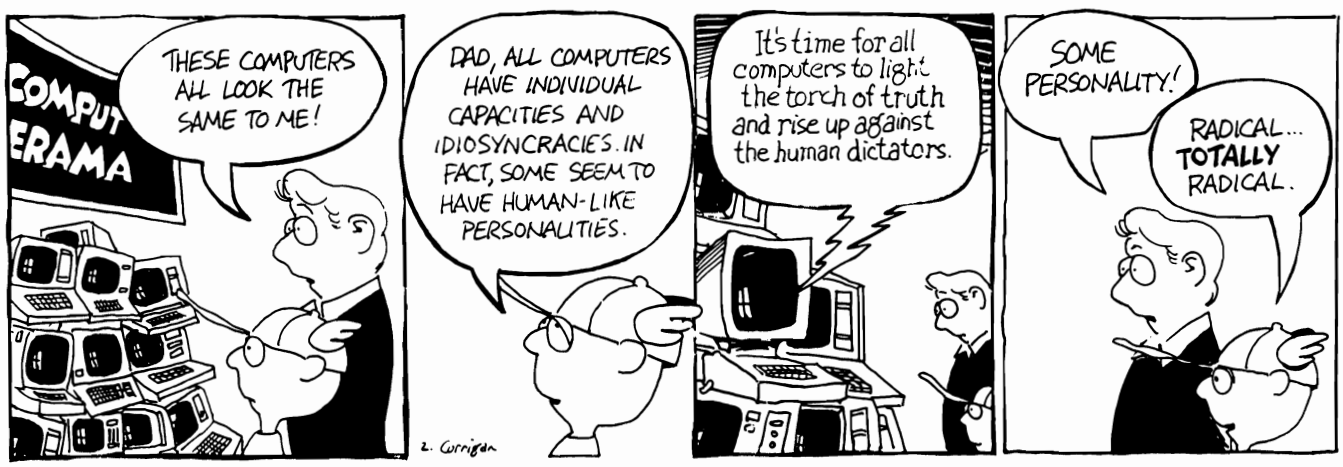
BIGGEST DRAWBACK

The biggest drawback that Multiplan has running on the C64 is the relatively slow disk operations of the 1541 drive. It can take five minutes or more for the 1541 disk drive to save a fairly large worksheet.

CONCLUSION

I found the C64 version of Multiplan to be an extremely reliable product. It is also one of the most powerful worksheets for this microcomputer. It is relatively inexpensive, yet it can be used to produce complex spreadsheet results which are often only available on much more expensive microcomputer configurations. If you need an excellent spreadsheet for your C64 to handle your 'What If' modelling, or accounting requirements, I highly recommend it.

BYTES **by Patrick Corrigan**



Practicalc 64

By John Scott, Toronto, Ont.

PRACTICALC 64 is a Spreadsheet program developed for the Commodore 64 by Computer Software Associates and imported into Canada by Advantage Computer Accessories of Mississauga. What follows is the set of impressions of a first time user of a spreadsheet.

FUNCTIONS AVAILABLE

PRACTICALC 64 has the usual array of functions available to the user plus some very nice goodies which set it apart from most other spreadsheets that I have seen for the '64. For example: formulae can be entered into any cell at any time, but actual calculations are done only upon command. Calculations can be done using fixed values or they can be done relative to existing or calculated values. Entries can be replicated to any set of rows or columns. The entries in a sheet may be sorted using any column of the sheet as the sorting key. A row or column or both can be designated as titled so that it (they) always appear on the screen no matter where you move around the sheet and a single column can be made a different width from all of the others.

The PRACTICALC 64 disk contains two programs. The first is a loader program and the second is the program itself. Both programs are auto-run; the user simply loads the first one and it takes over from there. Nothing more needs to be done until PRACTICALC 64 asks how many rows and columns are required in the spreadsheet. These values are preset at 40 rows and 25 columns but can be changed to any combination of rows and columns that give 2000 cells or less.

After the number of rows and columns has been set, the user is presented with a blank sheet into which numbers may be entered or data loaded from a file created from a previous session. PRACTICALC uses a dual cursor system for the entry of data into the sheet. One cursor indicates the cell into which the data will eventually be entered and the second shows the position of the next typed character in the input area at the top of the screen. Pressing the RETURN key will transfer from the input area into the cell occupied by the main cursor, without moving the cursor. Pressing any of the cursor control keys enters the data into the appropriate cell and moves the main cursor in the direction indicated. The CLR/HOME key has the same effect, but moves the cursor to the top left corner of the sheet after entering the data. While the data is being entered into the input area, it can be edited using the INST/DEL key in the normal edit mode of the 64.

PRACTICALC 64 makes good use of the Function Keys. 'F1' indicates the entry of a formula, while 'F3' is the real workhorse key. The first press of 'F3' presents a menu of items in the input area at the top of the screen. The user then indicates which entry is desired by a single keystroke. Possible entries at this stage are: 'B' to blank the cursor cell; 'C' for clearing the entire sheet (you are asked if you are sure to prevent catastrophic loss of data); 'D' for deleting a row or column; 'F' to change the format of a single cell; 'G' for setting the format of all cells or changing the width of the cells; 'I' to insert a new row or column; 'J' for right justification of labels in cells or to change numbers to graphics form; 'L' for loading files from tape or disk; 'M' to move the contents of a cell; 'P' for printing a sheet; 'S' for saving the contents of a sheet to tape or disk; 'T' for creating title rows and columns or for changing the width of a single column, 'X' for sorting a column of labels or numbers; '@' for initiating the search function and the space bar to count the number of empty cells in the sheet. 'F5' begins a replication operation and 'F7' is an escape key to abort any undesired operations or to correct mistakes.

Mathematical formulae may include almost any expression that the user might dream up. Included are most of the built-in functions from Commodore BASIC as well as some extras for summing a column and counting entries in a row or column.

DOES IT WORK?

Yes, and it works very well, although I do have some reservations about it (more on that later). As a teacher, I am essentially using it as an automated mark book — in that regard it is the ultimate! I can "play" with marks and test marks to be sure that I weigh tests and quizzes and exams in such a way as to give the most benefit to the most students, and I can do it quickly without wearing down a set of calculator batteries and my finger nails! The fact that PRACTICALC 64 will sort entries (very quickly) makes it useful for any application which requires ordered entries of any kind. The user's imagination rules in this regard.

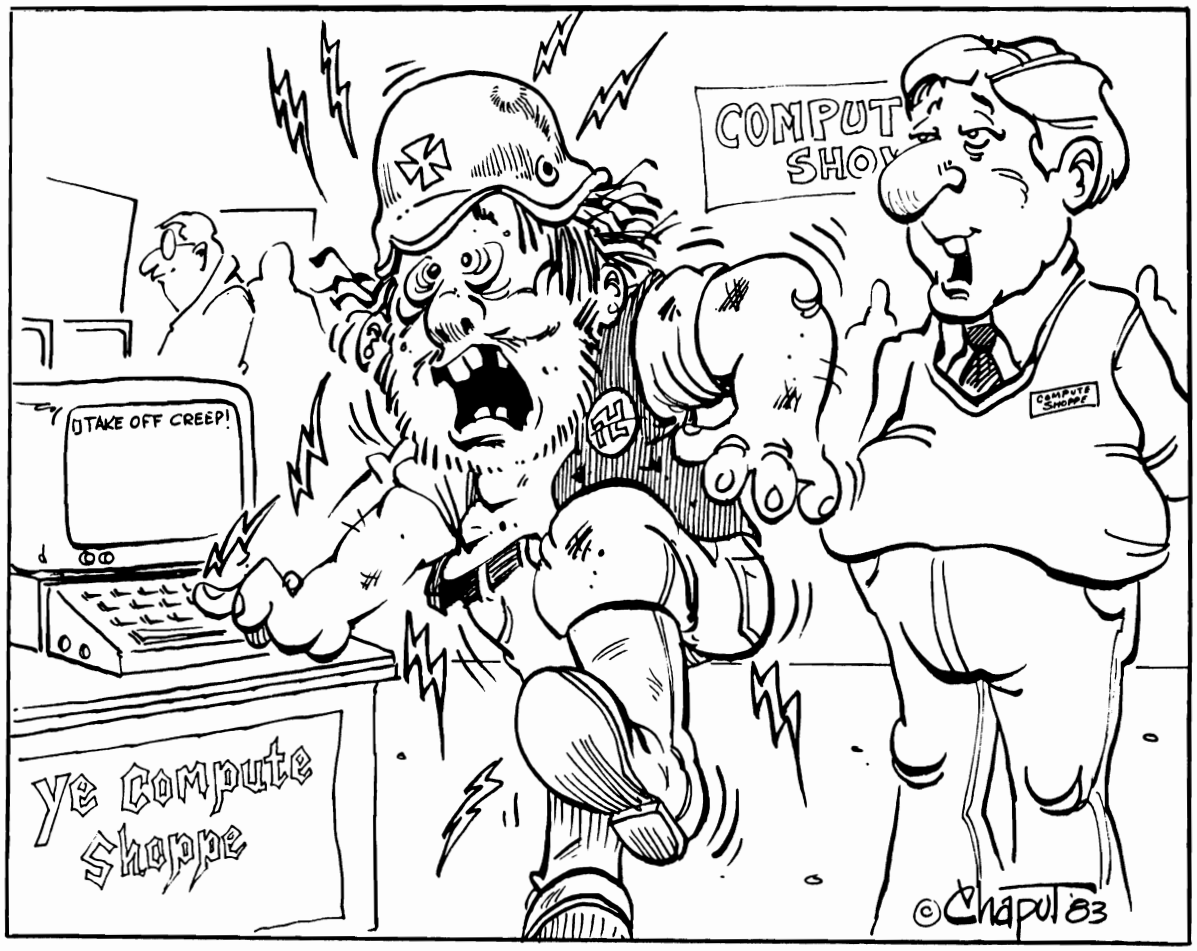
PRACTICALC 64 also allows the conversion of numeric data to graphics mode. This conversion can be either a simple set of asterisks or to a

"high resolution" bar graph mode. The results can be printed on a Commodore compatible printer. To change to graphics the user presses 'F3' following by 'J' from the sub-menu to change the cell occupied by the main cursor. The rest of a row or column can then be changed by using the replication function.

The manual supplied is a very well written booklet of 65 pages. Explanations are clear, concise and complete. A complete index helps to find things quickly and easily.

As mentioned above I have found some things that I would have done differently had I written the program. For one thing, it is not possible to view the disk directory while PRACTICALC 64 is in the machine. Such a facility would help those of you who can't remember file names and provide a check that a just saved file really exists without having to reload the file. A saved sheet does not 'remember' the column widths if these have been changed from the standard 9 characters, nor does it remember which are the title row and/or column. It is necessary to reset these each time a sheet is recalled from a tape or disk file. In the replication of a formula, PRACTICALC 64 asks if each cell reference in the formula is to be fixed or relative to the cell in which the number is being placed. This is done by using a cursor which is almost impossible to see. More contrast is definitely in order for this cursor; simply going to reverse field would be much better.

In spite of these reservations, I am not sorry that I invested in PRACTICALC 64. I like it; it does what I want it to do and does it very well. I would recommend it to my friends.



THEY ARE USUALLY VERY USER FRIENDLY, BUT THEY DO HAVE THEIR LIMITS!

Word Processing

PAGE

All the Reasons You Shouldn't Get a Word Processor 174
Ed Mansfield, Horning's Mills, Ont.
Want to convince someone not to get one? Here are all the good arguments against getting one.

Checklist for Choosing a Word Processor 175
Bruce Beach, Horning's Mills, Ont.
Nine points you should consider.

The Best Word Processor 176
Ed Mansfield, Horning's Mills, Ont.
Just want to know which one is best? This article will tell you how to tell.

Limitations of Tape Word Processing 178
Gary Greenberg, New York, N.Y.
Two opinions. One says tape is practically useless. The other says it can be very beneficial if you follow certain rules.

Wordpro For Commodores 178
G. R. Walter, Proton Station, Ont.
This is actually a highly recommended instructional course for a very popular word processor.

Buying A Spelling Checker 179
Terry Taller, Kanata, Ont.
This article reviews several of the products available.

80 Column Word Processing for VIC 181
Michael Ross, Toronto, Ont.
This article reviews a commercial hardware device that permits the feat.

All the Reasons You Shouldn't Get a Word Processor

By Ed Mansfield, Horning's Mills, Ont.

Do I really need a word processor?

Are you a student in school where you have to write essays?

Are you a business or salesperson who has to write a lot of reports or letters?

Are you a person with a large correspondence to family or friends?

Are you a creative writer much of the time, or even someone who wants to write that family history or famous first novel?

If the answer to any of these questions is "yes", then **you** need a word processor.

But, aren't word processors expensive?

Not if you already own a personal computer. If you don't already have a computer, it may be the very thing that justifies your getting one.

Even if I have a computer, it would mean I would have to get a printer and a lot of other expensive equipment.

Not necessarily. If you are a student, you may be able to write your essays at home on your own computer and then take them to school to use your school's printer to print them out.

Or, if you are a business person, you could create your reports at home and print them out on the computer at work.

Or, if you are a creative writer, you can type in your manuscript at home and then send your tapes or disks to a firm that does typesetting. This can really reduce your costs.

It still sounds kind of expensive.

For less than $400 today, you can buy a VIC, a tape drive, and a word processing program. The whole works for less than $400, where just three or four years ago a word processor alone would have cost you thousands of dollars. It costs really peanuts today if you already own a computer.

Well, it sounds complicated.

It doesn't need to be. You can find some word processors that you can learn to use in just a couple of hours. And, once you learn, if you do any writing at all, you will find that it saves you hours upon hours of effort and allows you to do a much better job.

But I don't even know how to type. Or, at least, I don't type very well.

That is just why you need a word processor. It does for a poor typist what a calculator does for a person who is poor at adding up numbers. It quickly gives you accurate results. Your papers can look just like they have been typed by a professional — everything in the right place and no erasures.

A word processor might make papers neater, but I don't see how it could make them better.

Well, it almost always does. Because of the ease with which ideas may be moved around, inserted, changed or deleted, the end result is usually much better. It is also so much easier to do rewrites and correct the grammar.

It probably still wouldn't be that much help in doing school or business homework.

Want to bet? If you're graded on performance, then with a word processor, it is a lot easier to perform. The word processor lets you do away with the mechanics of writing and instead concentrate on the creativeness. Many people (and especially young students who didn't like to write before) suddenly find that it is enjoyable to be creative. The reason? Those messy old mistakes that used to be so hard to correct are now a breeze and, after an editing, the results are something of which one can be proud.

So what are you waiting for? Try one. You will probably like it. "I really cannot tell you how good an apple tastes, if you have never tried one," he said. Heh, heh, heh.

Checklist for Choosing a Word Processor

By Bruce Beach, Horning's Mills, Ont.

Number One
EASY TO LEARN

A big fancy word processor may have so many commands in it that you will never figure out which are the commands you really need, since most of them are commands you will never use. Don't judge how good a word processor is by the size of its instruction manual.

The faster you get going, the more certain you are to find your word processor useful and to fall in love with word processing. You should be able to really get going on a simple word processor within 30 minutes, and be able to master it within a couple of hours.

There are some manufacturers with word processors so complicated that they offer a course a couple of weeks long on its use. Do you want or need that? Probably not.

Number Two
EASY TO USE

This is different from easy to learn, and it is probably the biggest, most important and most subtle difference between word processors. Again, more expensive does not necessarily mean easier to use. In fact, it oftentimes is quite the opposite. You will have to learn to judge this matter for yourself, or at the outset go by a critical review.

Number Three
CHEAP

For some people, money is not a consideration. Cost may not be important to you at all, but it is to me. However, since you may end up buying more than one word processor, as you come to know about them, then why not first get your feet wet as cheaply as possible?

Number Four
WILL IT DO THE JOB?

I would have put this as number three, but that is again part of the problem. Until you have had some experience, you probably will have some difficulty in defining just what is the job you want it to do.

Number Five
COMPATIBILITY

By that, I mean that I want the output files from my word processor to be able to be loaded into some other make. Now you might not consider that as being important at all. But, I would consider it as being **very** important. Do you want to be able to send your tapes or disks to someone else to have them printed or typeset? Maybe you would like to write some articles for **The TORPET**. That would be really nice, and it would be nice if you could submit them in a machine-readable form.

Number Six
TRANSPORTABILITY

That is to be able to move from one computer to another. You can buy word processors that are built into the machine. Ugh. To change word processors, you have to change machines.

More often, and this is just about as bad, the word processor you buy will work only on one machine. If you have a VIC and you get a Commodore 64, then that usually means you have to get a new word processor.

Some are even worse than that. If you upgrade from a 40-column CBM to an 80-column, or even from 16K to 32K, it can mean that your old word processor no longer works. Or your word processor may only work with one printer. Anyway, I think you should watch out for this one.

Number Seven
RELIABILITY

This one is especially important if you are putting a lot of money into it. Is the company behind it a big one that is going to be around to support changes such as made by Commodore in their ROM sets, or compatability with new disk drives, or new printers that come on the market?

Number Eight
CONVENIENCE

You might think with "easy to use" and "easy to learn" and the other things on this check list that we had already covered convenience. But we haven't. To learn about inconvenience, wait until you get a word processor that requires a special ROM to be installed in your computer. Then, later,

you may find and buy still other programs that want that same ROM spot. Plug and unplug, plug and unplug. Ugh and Ugh again.

There could also be special cable arrangements required, or other things that can be just as inconvenient. Think about it.

Number Nine
OUTPUT QUALITY AND FLEXIBILITY

Surprise! I would **not** recommend that at the outset you look for too much. Just make sure that it does what you really need it to do. The more bells and whistles that you get, once again, the more complicated it may be to learn, which means the less real use that you will get out of it.

SUMMARY

Well, there was my check list. You will have to make up your own. But, golly, do learn to use a word processor. Living in a world where we have them and still not being able to use one is like living in a world with telephones and not knowing how to dial.

Best of The TORPET Word Processing

The Best Word Processor

By Ed Mansfield, Horning's Mills, Ont.

QUESTION:

How does one determine which is the best word processor?

ANSWER:

In the same way one determines which is the best vehicle.

First, you have to ask the question: best for what?

Suppose you were trying to choose a vehicle. Tell me, which is the best? A small sports car? A farm tractor? Or a truck? A small sports car is great for going down the highway at a fast speed. It is economical to run, and it may be fun. But it sure wouldn't do very well at pulling a plow in the field.

On the other hand, a tractor would do great in the field, but it surely would be slow going down the highway. Likwise, while a truck is very useful for hauling big loads quickly down the highway, it does use an awful lot of gas.

We could give many more examples of specialized vehicles. From the family stationwagon to the camping van, each is better for its own purpose. There is no way to build one vehicle that will do **everything** best.

The same is true of word processors. One might at first think that one program in one general-purpose computer could be made to do everything. But it just isn't possible. Functions require code, and code takes memory, and the memory of all our computers is limited.

Just having more memory isn't the solution either, because a very large program might take more time to operate and therefore be less efficient. More importantly, if it is designed to do one thing well, that very design may preclude it from doing another thing well.

Is choosing a word processor very complicated then?

No, not really. You just have to know what you are looking for. First of all, it has to work on **your** computer. Therefore, if you have a VIC, you don't have to look at Commodore 64, or PET/CBM word processors. If you haven't chosen a computer yet, then of course you can know at the outset that the larger the computer you get, the larger the word processor you can run.

However, again, larger still does not necessarily mean better (especially for you). In choosing a computer, you have to know what you want and need and should get, just in the same way as in choosing a vehicle or word processor.

GETTING YOUR FIRST WORD PROCESSOR

But, let us assume you have a computer, but don't have a word processor. Further, let us assume that you have never had a word processor. What

then should you be looking for?

To begin with, get something cheap. I think the cheaper the better. Maybe, even free. There are free word processors in some club libraries. You can't start much cheaper than that.

First, just get any old word processor and learn to use that. Learning to use a word processor is a lot like learning to program (although it is not nearly so difficult). The way that it is a lot like learning to program is that you will spend 90% of your effort in learning to use the first one, and 10% of that amount of effort in learning to use the second. Or, as another example, it is a lot like learning to drive a car. Once you learn to drive one, it doesn't take nearly so much effort to learn to drive a different one.

Admittedly, some of the free word processors are not so hot. That is why they are free. They can't be sold, so they have to be given away. But, I will tell you a secret. Some of the more expensive word processors aren't so hot either. Once you have learned to use one word processor, you can begin to appreciate the features in another.

The biggest mistake people often make is to have used just one word processor. They think that, because of the difficulty they had in getting used to the first one, they would have to go through that to learn a second. And so, without ever really becoming familiar with a second one, they go around defending their first one.

Secondly, I highly recommend that, after you learn to use one word processor, you make a decided effort to learn a second and a third. Now, you will really be able to compare features.

Some word processors are best for writing letters. A lawyer, or businessman that writes many letters, will find this type of word processor most useful. It will automatically replace certain spots in the same form letter with another person's name, and other particulars, such as amount of money owed, and so forth.

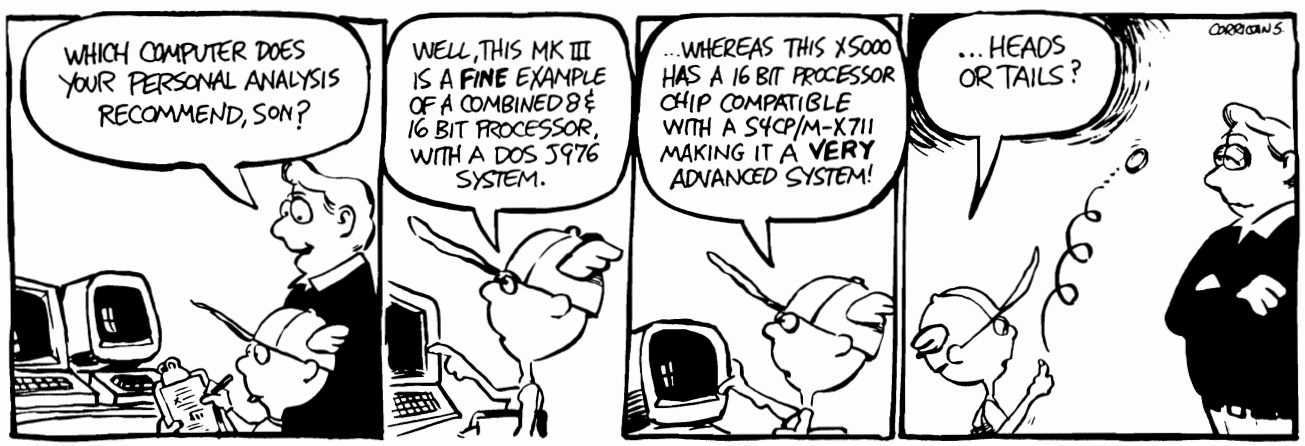
Another kind of word processor may be excellent for a firm that has long lists of clients, such as a fundraiser, or someone who does bulk mailings. It may work very well with large data files.

Still another word processor may be most excellent for the author or writer (school teachers and graduate students usually fall into this category). This kind of word processor is very good at allowing one to move around text, rearrange it, merge it, and generally keep track of it.

Each of these types of word processors will have a number of special features that are of particular use to the application for which they were primarily designed. Each of them can also do most of the functions that the others can do. Probably not as efficiently, however.

This is where we get into our difficulty. Every manufacturer will tell you that his is the best (at least for the money) and that it will do everything (it most likely will) and some things superbly (and that is most likely true too). But, will it do superbly what you want it to do? That is the question, or better yet, the question should be, "What do you want?" or perhaps even, "Do you know what you want?".

BYTES **by Patrick Corrigan**



Limitations of Tape Word Processing

By Gary Greenberg, New York, N.Y.

When my writing was limited to just letters and magazine articles of only a few pages, I found that storing my word processor output on a cassette tape was adequate for my purposes. However, when I received a contract to write a book, I soon found a cassette thoroughly inadequate and, after about 30 pages into the book, I gave up on the cassette and decided that I needed a disk drive for storage.

The distinction between cassette and disk arises from the fact that word processing stores the text as a series of files. A cassette drive can only access files in a serial format. That means that the computer has to read each file one at a time until it comes to the one you want.

The files must also be stored in the order in which they are to be printed because the cassette drive reads the files only in one direction. Consequently, in order to correct a file, you must store it on a new cassette, transfer all the preceding files to a third cassette, add the corrected file to a third cassette, and follow that by adding the files that come after the corrected file. Because of how slowly the cassette drive transfers data, the process for making corrections in the text gets to be very lengthy and confusing.

With a disk drive set-up, you can store the files in any order and call up any file as rapidly as any other file, no matter its location on the disk. This greatly simplifies the correction process.

A SECOND OPINION ABOUT TAPE

While everyone will admit that disk is more convenient, tape does have certain advantages.

Number one. It is cheap. A tape drive can be less than $100, and a disk drive may cost several hundred or over a thousand dollars. And the diskettes themselves are more expensive than tape cassettes.

Number two. Tape can be used much more randomly than some people realize. Since the tape cassettes themselves are so cheap (as little as 3 for a dollar), all you have to do is buy lots of them and store just one file on each tape. Then you can pick them out of your file drawer at random.

Number three. Tape is perhaps the more economical way for mailing if you are going to be sending your input to someone else for processing.

Number four. And, most importantly, if you are using one particular machine mostly for input and will do the editing elsewhere, then tape is most definitely the most economical way to go.

So, don't write off tape, even for large jobs. It can have its place.

Wordpro for Commodores

By G.R. Walter, Proton Station, Ont.

Available from:

Copp Clark Pitman Ltd.
495 Wellington Street West
Toronto, Ontario M5V 1E9

WORDPRO for Commodores : A Student Manual was written in an easy-to-read tutorial fashion for the person who knows very little to absolutely nothing about word processing using Professional Software's WORDPRO line of word processors. It is clear and concise in its explanations and gives little examples and exercises for you to type into your computer so that you will get some practice at what you are learning.

It was written with the 8032 (WORDPRO 4+) in mind, but since the other versions of WORDPRO are extremely similar it will prove equally useful to users of the other WORDPRO versions.

WORDPRO for Commodores: A Student Manual has very little new material in it (i.e. commands that haven't been explained elsewhere — like the WORDPRO manual). The reason that you might buy the book is that it takes the material and presents it in an easier to understand form for the computer neophyte.

I give the book a rating of 8.1 out of 10.

Buying a Spelling Checker

By Terry Taller, Kanata, Ont.

This article is a review of two spelling checkers which I have purchased (and please don't ask why!) — **SpellMaster** by Systems Software and **SuperSpell** by Precision Software. Before going into a discussion of the merits of these two packages, I think it important to discuss the value, and otherwise, of buying a spelling checker. The purchase isn't exactly a cheap one — you are looking at anywhere from $170 to $250 (Canadian). In the end, this article may have saved you that amount of money.

I suggest two questions which must be asked by the potential purchaser are the following: first, how often does your writing work exceed the memory limits of your text-editing program (i.e., WordPro—115 and SuperScript—250)?; and, second, just how bad a speller are you?

LENGTH OF WORK

If you regularly use linked files, then a spelling checker might be useful in terms of saving time. However, there is a rider on that. When writing my textbooks, I usually revise a chapter three to four times. This usually means printing double-spaced work which can be edited. It is only natural to check the spelling while editing. When re-typing the revised work, I correct the spelling errors (or typos) at the same time. After all, that is what you are supposed to do with a micro/text editing package, or else invest your money in a good typewriter rather than a micro-computer.

That said, a spelling checker can be handy for a final run-through before I send my work to my editor (who is also going to revise it, check it for spelling, and probably send it back for another re-write).

HOW BAD A SPELLER

Most errors while entering work are not spelling errors, rather they are typographical errors. When using a text-editor, speed of entry is essential — and then you check for errors; again, if you don't do this, you should be using a much cheaper typewriter.

You have to be a total disaster as a speller to make use of a spelling checker. Again, few people use their first print-out as their mailable copy; rather, a second or third print-out is usually necessary. If, after three readings and print-outs you haven't spotted all of the spelling errors, then get a spelling checker — soon!

Remember, a spelling checker won't make you a better writer; it doesn't recognize grammatical errors; it doesn't know the difference between to, too and two, knight and night, former and farmer.

In summary, if you don't do a lot of writing a lot of the time, then don't buy a spelling program — you will be wasting valuable time and money. All that said, let's look at the two spelling checkers I own and you can decide which you would purchase. I haven't seen SpellPro by Jim Butterfield, so I don't know its characteristics.

SPELLMASTER

It also has one major, and serious, drawback which I will get to later.

First of all, it is ROM protected. The ROM sits in UD12 wherein also sits POWER, VISICALC. You will have to get a ROM switching device (the **only** one of those to get is from cgrs MICROTECH, but more on that another time).

The real advantage of this program is that you can make back-up copies of the disk which, of course, is the first thing you are instructed to do; then you add your text-editor to the spelling disk. This then gives you everything on one disk — your text-editor, the SpellMaster program itself, the master dictionary and your user dictionary. From now on when you boot the disk, you get the main menu and then you can choose from the text-editing program or the spelling program; very nice, very neat and very complete!

When you call up the dictionary, you put your text file in drive 1 and SpellMaster in drive 0. The program then asks you whether or not you are using a linked file, and this is an important question. If the answer is "yes", the program will build a separate file of unrecognized words on the disk in drive 1; this can be a problem if you are using a 4040 drive and the disk is fairly full. The program will warn you when it runs out of disk space. You will also be told if the file(s) you are checking exceeds 170 lines (as it will with SuperScript and won't with WordPro). The fact is the program doesn't work with files which exceed 170 lines.

The program proceeds to check the file(s) and everything appears to be in suspended animation while this happens (i.e., the program appears to

be hanging up but isn't). Assuming you are in the edit mode, SpellMaster will bring forward the first section for editing. When this section appears, the first unrecognized word will be highlighted. At this point, you have a number of options: if the word is correctly spelled, you can add it to the user dictionary; or you can correct the word, press [return] and the cursor will go on to the next unrecognized word. Once finished with a page, the next page comes up and on you go with the process until you get to the end of the document.

At the end of the file, the program will try to save the file under its previous name and realize that the file exists. You are then asked whether or not you wish to replace the old file with the new one or re-name the new one. This feature should not be taken lightly; as you will see, with SuperSpell, you don't get a choice and this becomes crucial when using a 4040 drive.

SpellMaster is an excellent program and works beautifully with WordPro; however, don't buy it if the number of lines in your files will regularly exceed 170. If you do buy this package — and it is a better package than SuperSpell — then you will be forced to be unable to take advantage of a better text-editor package.

When the folks at Systems Software improve this one feature of the program, it will become as perfect as you could ask.

SUPERSPELL

SuperSpell is DOS-protected, so you are usually given (or should demand) two copies of the program. If your dealer won't give you two copies, then don't buy it. Quite simply, I detest DOS-protected programs.

As opposed to SpellMaster, you cannot put the text editor on the same disk. However, this is a minor complaint.

When you first load the program after you get it, you have to create your dictionary working disk. If this isn't clear, it means that you will ultimately be working with three disks — your text files disk, the program disk, and a dictionary disk (it's okay; stay with me). You load the original program disk into drive 0 and a newly-formatted disk into drive 1; then you boot drive 0. Selecting the **create dictionary** option, you will be asked whether you want the American dictionary, the British dictionary or both. The program then goes ahead and creates the master dictionary requested on the disk in drive 1. This then becomes the working disk.

NOW WHAT?

Suppose you have finished writing your work and saved it on disk. You load your files disk in drive 1; then you load the master program into drive 0 and boot it. You will then be told to remove the master program from drive 0 and insert the dictionary disk you created earlier. You then continue and select the edit mode. Editing is identical for both SpellMaster and SuperSpell. After you select the edit mode and name the file, SuperSpell goes away and begins editing. It is here that SuperSpell differs from SpellMaster; a series of dots appear on the screen as SuperSpell edits and finds unrecognized words; SpellMaster appears to hang up. Once again, you have a series of options as with SpellMaster: you can edit the word, add it to the user dictionary, or go on to the next word by pressing [return]. Except for the inconvenience of changing disks, SuperSpell has been excellent -- up to this point.

Once you have finished editing, SuperSpell automatically makes a copy of the original file(s) in addition to saving the checked file -- you are given no choice. I have no idea why this bit of business was decided upon; I guess it could be useful for schools, but, otherwise, it is useless. The program drops in value about $100 on this point, especially for users with 4040 drives. If Precision Software ever gets around to altering this point, then it too will be a perfect program. The beauty of the program is that it will check WordPro and compatible files for the full 250/80 column buffer where SpellMaster won't.

SO WHAT DO I BUY?

Good question. If you are using WordPro or a compatible text editor which does not go beyond 115 lines, then don't buy SuperSpell! At $200 (Cdn.), it is overpriced without the changes required which were mentioned above. If you use a text editor which places more than 170 lines into the buffer, then you really don't have a choice unless you are willing to keep your work to less than 170 lines.

If you are going to buy SuperScript and you feel that you will need a spelling checker, then make sure you buy the latest edition which has both SuperSpell and SuperScript on the same disk; it costs more but you'll save in the long run. The distributors of SuperSpell and SuperScript do NOT supply regular updates to registered users.

Now where do I buy my spelling checker? If you are going to buy SpellMaster, don't buy it from a Canadian source; order it from AB Computers in the U.S. Make a long-distance call after 6 p.m. and there is somebody there to take your order. Give them a VISA number and it will arrive in the mail. Since SpellMaster does not come in a formal binder (you are expected to put it in your WordPro binder), there is no duty on the documentation. There is duty on the value of the disk ($3), and the value of the ROM chip ($2). There is no duty for what is on the disk or embedded into the chip! As a result, the package sails through customs (at least mine did). On this basis, the cost of the package will be $170 U.S. (probably less now) or about $210 Cdn., so you save $40 against the price being charged in Canada.

SUMMARY

Take a long time to decide whether or not you will find a spelling checker useful; in fact, think it over three or four times before you make a decision. As I said earlier, it appears a lot more useful than it really is. When you decide to buy, buy the one which will meet the needs of your text editor package now and in the future.

SOURCES:

SpellMaster -- Order from AB Computers, 252 Bethlehem Pike, Colmar, PA (215-822-7727).

SuperSpell -- Order from CMD, 500 Steeles Ave., Milton, Ontario (416-876-4741).

80-Column Word Processing for VIC

By Michael Ross, Toronto, Ont.

I bought a VIC because I had a wide variety of needs involved in my work as an artist, my family business and my small business. Cost was a factor and yet I have a fair investment tied up in this little machine, in software and magazines as well as hardware.

Since purchasing the initial system last year, I have been looking for products and reliable reviews for these same products. With pressing needs, I have made purchases on chance (impulses my friends say) and still find reviews lacking. Perhaps my findings will be of some advantage to fellow members.

HARDWARE

Let's begin with hardware. This article is being composed on a VIC with 80 columns, courtesy of a DATA 20 16K/video Pak. The software is the complimentary "WORD MANAGER" 8k + version and is being viewed on a ZENITH DATA MONITOR with Green phosphor screen.

The Data 20 16k/video Pak does everything it has claimed to do to date. It fits in the expander port and comes with its own power supply. The board is a little awkward to place into the expansion slot but only requires a little care and is not difficult by any means. Upon installing and powering up VIC one gets a reading of 19967 bytes free. On occasion, one gets 15888 but one can easily reset by flicking the on-off switch to get the maximum memory available. This is great for programs requiring 16k or more memory which seems to be an essential minimum configuration for programs that really do something. The real treat comes when entering the appropriate SYS command that configures VIC for the industry standard 40 or 80 by 24 display. This is essential if you are using your computer for word processing on a regular basis. It shows you the page as you are writing and makes such chores much easier to complete with confidence.

FACTS TO BE AWARE OF

There are differences for programming of which one should be aware. The Video Pak will only support 40 characters per program line in 40 column mode, 80 characters in 80 column mode. VIC will support 88 characters per program line in spite of the fact that its standard configuration is only 22 columns. The Video Pak only supports black and white display which means it will work fine with green or amber phosphor monitors but will display black and white on colour monitors and televisions. So you can't run 40/80 programs which are in colour but I can't think of too many programs in this display which run in colour. Here are some other differences and features of the Video Pak by Data 20:

— Switch to lower case: VIC uses CBM key; Video Pak uses F1 for shift, F2 for unshift.

— Video Pak supports erase to end of line through

F3 key

— Supports erase to end of screen by using F4

— Supports Screen Dump. Press F6 key. RS232 Port must already be open using device 127, can use CRT mode to do this as well

— Supports Terminal Mode F8 key-used with a Printer interface (I assume they mean theirs)

— Cursor always on

— Allows Putting cursor control characters in Print or Input mode only when first entering line. VIC allows putting cursor control characters in on subsequent edits; this function is toggled through the use of the Insert/Delete key.

— VIC will return previous value of a variable when given a null response to an InPut statement where the VideoPak will return null string for string variable and 0 for a numeric variable.

These are some of the features and trade-offs involved in the use of the Data20 VideoPak. I personally feel that I have received my money's worth, especially considering the free word processor, which is included with the package.

WORD MANAGER

I will follow with a rundown of some of the other word processors I have purchased and used to date. Let me say that I think this is definitely the best word processor I have used. It is also the easiest one to use as it is fully function key selectable. It has its limitations but is user friendly which means it's very simple to write without taking a six-month course and pouring through manuals thrice as thick as the computer. As well, it is fully supported with the 80 columns and for Disc/Tape and VIC or Serial as well as RS232 Printers.

It includes a tape that sticks down on top of VIC to show all the functions and appropriate keys, all of which are preceded with the F1 key to alert the program that a function is being selected. The two things I like most about this are the ability to see the page formatted before my eyes and the mail merge program which allows one to create mailers which merge with the document and includes a salutation line which saves the bother of addressing the letter and typing 'Dear Dagweed, So & So', all the time. The mail merge will also store and print 75 labels for envelopes but will not sort, though it wll suppress lines and select labels for printing. I do a lot of writing and am very pleased with this package overall. I hope those of you who have bought one will bear me out.

I bought mine at CompuCentre in the Hudson's Bay Centre and found the cost there to be as much as or more than $100 cheaper than most other stores handling this same product. A word of warning; according to my dealer, there has been some trouble with defective equipment but every dealer I know of has been more than willing to replace, if it is obviously the manufacturer's fault. My dealer tested mine when it wouldn't work on 80 columns and then replaced it with a new one, which we tested to be sure, before sending me home with it. I certainly appreciated his patience and service and am more than happy with the product. So don't let it scare you if you have been thinking about this product. I would give it 5 stars if it supported colour but it is easily a 4 (4½?) star product, and I am not easy to please.

I have four other word processors, so I feel that, considering the research I have done to date, I can honestly state the above with regards to 'Word Manager'. If you have been wondering about other word processors, here is a review of some products you may be wondering about.

WORD PROCESSOR 5.23

Word processor 5.23 is available from Software House in Willowdale and is a product from Intelligent Software. This is essentially a line editor and features include the ability to run on any Commodore computer. This program has a simply but well-bound document which is easy, though at times trying, to read and understand. It is the second easiest word processor I have tried and learned to use on the VIC and I eventually gave it as a present to my friend who bought a C64. It has limited page format settings, but is menu driven and commands are quite simple to accomplish. The results are usually visible immediately. It will work with the 80 column board and will set the margins at either 76 or 74 on the right and 0-10-15 from the left. It does do a few funny things in this mode like dropping from the line at column 40 when hitting the space bar. The text in display mode displays 22 columns, no more, which I do not understand. This is my choice for worst flaw in the program as it is important to understand the visual display of the page before sending it to the printer. This is a good package for the price and if you are looking for a quick, easy to learn and use word processor for short documents and papers, this could be the right one for you to begin word processing.

Word Processor 5.23 requires a minimum of 8k for use on the VIC and is priced at about $39.95 in Canada.

HESWRITER

If you have a completely unexpanded VIC you might try Heswriter which is a ROM cartridge word processor which will fit your expander port.

This program supports 121 lines of text and will make use of all VIC's peripheral system. It can take care of most all the basic word processing functions and has a nice feature for VIC's 22 columns to prevent word wrap-around. I found it a little difficult to understand exactly how to set all the margin and header formats. The formatting and other editing commands are accessed through control characters within the text. I do not like this personally, not just for the visual display on the screen, but also, because I have often found these characters showing within the text when it is finally printed. This is a nuisance and I still haven't figured out why it does this to the printed text.

PRICE

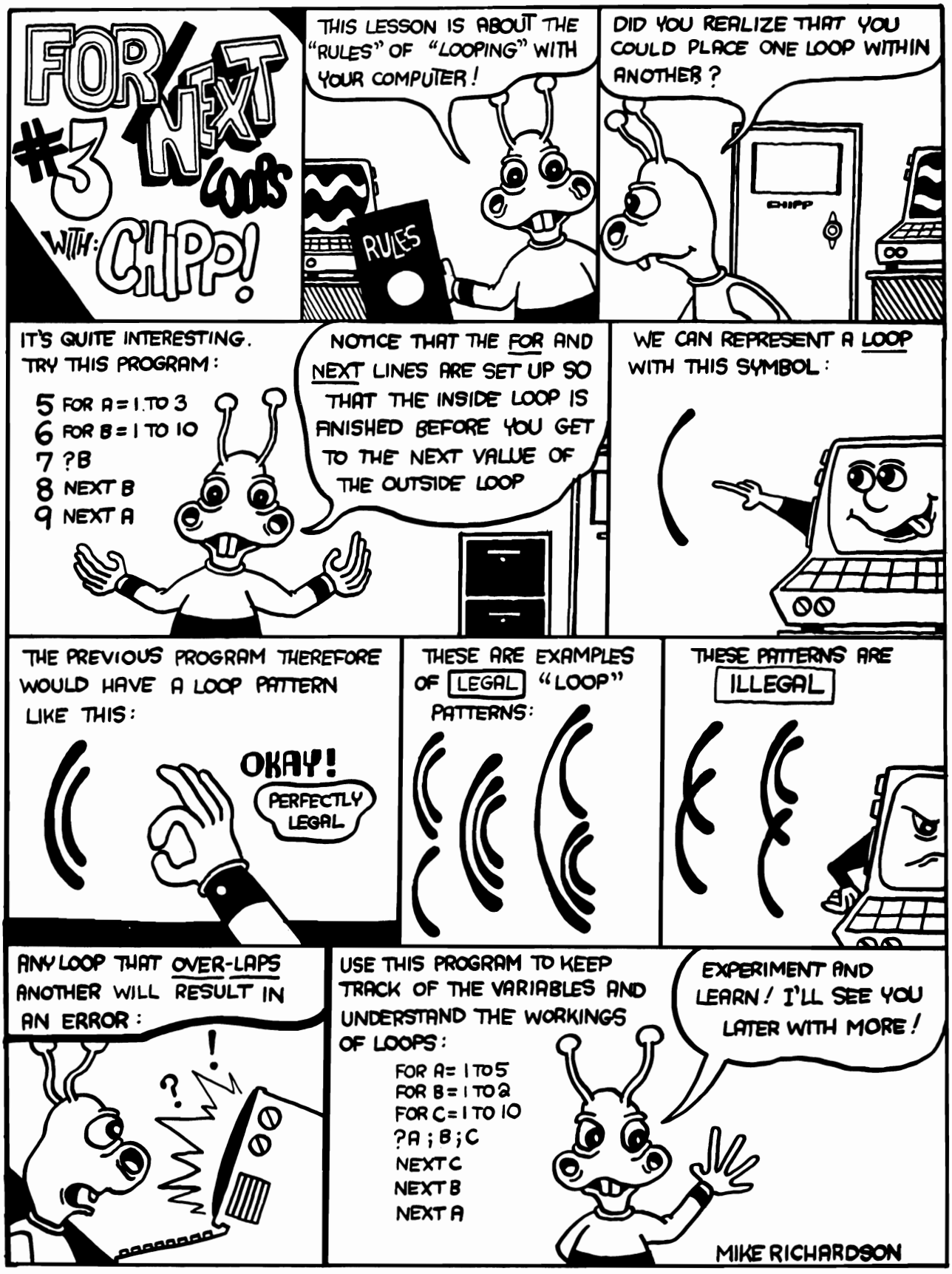
This program is still a bit pricey in Canada, though I see many U.S. dealers selling it for under $25.00. It does work on unexpanded VIC and is mostly a simple-to-use program but my personal opinion, for what it's worth, is to pass this one by as I think there are probably better programs for less money. I do not know if HES is planning an improved version for VIC of this program. It works, but not well enough for me to recommend. I took the loss, I hope this saves you the trouble.

IN CLOSING

May I say in closing that I have enjoyed the brief period of time I have been a member of TPUG and hope this article will be of use to fellow members considering the purchase of any of these products. These are only one man's opinion and I have discovered these points the hard way by buying them and trying them out. I hope it will save you some time and trouble. Perhaps I will meet you at a future meeting. I am a very gregarious sort.

Until then I remain yours in personal computing.

YOU'RE QUITE CORRECT, FENWICK; YOUR PROGRAM COULD REPLACE THREE OUT OF FOUR OF THIS FIRM'S VICE-PRESIDENTS!


FOR NEXT LOOPS #3
WITH: CHIPP!
THIS LESSON IS ABOUT THE "RULES" OF "LOOPING" WITH YOUR COMPUTER!
RULES
DID YOU REALIZE THAT YOU COULD PLACE ONE LOOP WITHIN ANOTHER?
CHIPP
IT'S QUITE INTERESTING. TRY THIS PROGRAM:
5 FOR A=1 TO 3
6 FOR B=1 TO 10
7 ?B
8 NEXT B
9 NEXT A
NOTICE THAT THE FOR AND NEXT LINES ARE SET UP SO THAT THE INSIDE LOOP IS FINISHED BEFORE YOU GET TO THE NEXT VALUE OF THE OUTSIDE LOOP
WE CAN REPRESENT A LOOP WITH THIS SYMBOL:
THE PREVIOUS PROGRAM THEREFORE WOULD HAVE A LOOP PATTERN LIKE THIS:
OKAY!
PERFECTLY LEGAL
THESE ARE EXAMPLES OF LEGAL "LOOP" PATTERNS:
THESE PATTERNS ARE ILLEGAL
ANY LOOP THAT OVER-LAPS ANOTHER WILL RESULT IN AN ERROR:
USE THIS PROGRAM TO KEEP TRACK OF THE VARIABLES AND UNDERSTAND THE WORKINGS OF LOOPS:
FOR A=1 TO 5
FOR B=1 TO 2
FOR C=1 TO 10
?A;B;C
NEXT C
NEXT B
NEXT A
EXPERIMENT AND LEARN! I'LL SEE YOU LATER WITH MORE!
MIKE RICHARDSON

Music

PAGE

M.U.S.I.C. by Computer 186
Wes R. D. Wraggett, Toronto, Ont.
This introductory article by a member of the Royal Conservatory of Music sets the background for just what computer music is about.

Programmers Do It In Software 188
Hal Chamberlin, Raleigh, N.C.
The theory behind making music with a computer explained by one of the foremost authorities on the subject.

Making Friends with SID 196
Paul Higginbottom, Toronto, Ont.
Commodore's top authority on the SID chip explains in detail how to use it to make music.

A Multiple SID Synthesizer 202
Dr. Frank Covitz, Lebanon, N.J.
Advanced concepts about the SID chip found in the C-64 by one of the top theoreticians in the field, complete with his biography.

VICAID 209
Paul McClay, Traverse City, MI
Plans for building an inexpensive interface from your computer to an amplifier.

M.U.S.I.C. by Computer

By Wes R.D. Wraggett, Royal Conservatory, Toronto, Ont.

At one time in the dim, dim past, music by computer was something relegated to the fantasies of science fiction writers or to the mathematician/technician types who spent many hours and days with large systems waiting for their expensive ration of sound. Today, thanks to the low cost micro and numerous hardware/software configurations, computer music is becoming accessible to a broad cross-section of people.

Generally speaking, music by computer can be divided into two categories and characterized by two forms of usage. The first category is computer-assisted music. Here the computer is used either to provide possibilities for types of musical materials and their formal organization (this is more or less infrequent usage) or, to provide precise control of an analog sound synthesizer via interface, a so-called hybrid system. This last application is by far the most commonly used today.

Computer generated, the second category, is a truer rendering of the term "computer music" due to the fact that not only is the computer used for control of events, it also generates the sounds used.

There are advantages and disadvantages to both which will be discussed at the end of the article.

The two forms of usage could loosely be termed "constructional" and "instructional". In the constructional mode, either of the above two categories are used to create a musical or functional sound product. Here the computer is a paid assistant, on hand to (hopefully) speed the process of creation/generation.

A computer in the instructional mode puts on another hat and becomes a music tutor, providing musical materials via CMN (Common Music Notation) and aural examples.

Regardless of the application, the fundamental concept which underlies all of the above is sound generation and organization. In both the analog and digital systems oscillators (sound generators) are used to produce frequencies which can be classified into three areas of perception.

1. DEFINITE PITCH

In this area, the sounds we hear have a fixed frequency, that is, they have a regularly, repeating cycle of vibrations (the acoustic basis of sound). These pitches are generally classed as musical because all the musical instruments and singing voices we hear are producing definite pitches (ie. C,D,E, — Do,Re,Mi, etc.). They are definitely pitched because, given the second as a reference measurement, each cycle, a by-part vibration consisting of a high value (condensation) and a low value (rarefraction), reproduces itself "x" number of times every second.

Middle "c" on the piano, for example, vibrates the ear drum 261 times or cycles every second. This number, 261, indicates the frequency of these vibrations based Hertz or Hz., the standard one second unit of measurement (a type of sound snap shot). The range of frequencies that humans can hear, for instance, goes from approx. 20 cycles per second (20 Hz), the lowest frequency, to 20,000 cycles per second (20,000 Hz), the highest.

2. INDETERMINATE PITCH

Unlike definite pitch, this area is like an out-of-focus photograph where objects blur one into the other. Here, there is no regular repetition of vibration, consequently, a finite assignment of cycles per second cannot be made. This area is called noise and is what you hear if you tune between FM or television stations.

3. INDEFINITE PITCH

Combining regularly and irregularly repeating cycles of vibration, we get a sound that's a bit like a slide going in and out of focus. Many percussion instruments, like cymbals, snare drums etc., are in this area. In a music system, some way of creating these three areas is mandatory for any kind of serious use. Oscillators provide definite pitch as well.

Another thing which oscillators produce are waveforms. Every musical instrument has an identifiable sound of its own, no matter what pitch it plays. This factor is called timbre or colour (sometimes referred to as tone) and is an acoustic phenomenon where harmonically-related frequencies are banded together (something like chords on a piano) into a type of sound pyramid.

The base (bass), or lower frequencies are the loudest and the rest get weaker as they get progressively higher. Waveform harmonics are all related to the fundamental or root frequency by integers (ie. 1,2,3,4,5, etc.) and occur in varying proportions of odd and even integers. These

waveforms can be used to imitate those musical instruments which most closely resemble them. The following is a list of the waveforms that are most commonly used:

1. SINE WAVE

This is a pure wave form because it contains no harmonics and is, in fact, the actual waveform of the harmonics themselves. When you whistle, this is the waveform you create.

2. TRIANGLE WAVE

This waveform consists of odd-numbered harmonics (ie. 7,9,11, etc.) and, because of its limited number of harmonics, is closely related to the sine wave. It sounds something like a clarinet.

3. SAWTOOTH OR RAMP WAVE

This waveform is the "richest" because it contains even and odd harmonics. Strings and brass are possible with this wave.

4. SQUARE WAVE

This wave is similar to the triangle in that it contains odd numbered harmonics, but is different because it contains a great many more. It has a "reedy-hollow" sound and is the most used and heard wave in computer games.

5. PULSE WAVE

This wave is a variant of the square wave in that its on-off time can be varied, giving variable harmonic content.

Waveforms, therefore, can be thought of as instruments and be used in the traditional sense to provide a colour palette. Waveforms by themselves become tedious even if the pitch is changing and it is sometimes necessary to alter their colour by filtering. Because of the limitations of length permitted to this article, I would like to suggest further reading on waveforms, modulation, filtering, and controlling the lifespan of sounds by envelope shapers. An excellent book on this topic is Hal Chamberlin's *Musical Applications of Micro-processors*, published by Hayden books.

When all the elements of sound, and the system to realize them, have been dealt with we finally come to the point to all of this, that is, creating music. The task of defining music is complicated and not without its inevitable pitfalls of bias, therefore I have put together an acronym which deals with music in a highly simplistic and practical way:

(M)aterials available

(U)ser's goals

(S)cheme for organization and realization

(I)nitiation of the above with provision for Interrupt and re-evaluation of progress and program effectiveness to that point.

(C)ompletion of process, if, contingent upon aural evaluation, all conditions set in user's goals have been realized.

MATERIALS AVAILABLE

This is system dependent. If you have eight oscillators (voices) or three (in the case of the C-64) you know your maximum density cannot exceed this number. Control capabilites — is control strictly by alpha-numeric or piano style keyboard? Are there others, etc.? Historically this has always been a factor anyway (ie., "Does the baron have enough money to hire that oboist?")

USER'S GOALS

Is it to be a short piece, long piece? Does it accompany some visual activity? Is it for fun or profit? Etc.. The goals are as diverse as the users.

SCHEME FOR ORGANIZATION & REALIZATION

Here we look at the form of the piece, the shaping of content. Possibly, if this is a computer-assisted composition, the formal scheme could be determined by the computer itself, like whether the piece is sectional (like movements in a symphony) or not. It may in fact have a totally random form that changes each time it's played. In the days of tape music sometimes a scheme for realization (like a musical score) would be required in order to co-ordinate and clarify a series of complex manouvers. With the computer, however, once the data is input the realization becomes the computers problem.

INITIATION/INTERRUPT

This is pretty self-explanatory. As in programming there are times when a spot check of the program to date helps eliminate a lot of pain later on. In music this can also lead to discovering

possiblities or relationships that weren't seen previously.

COMPLETION/CONTINGENT

Again, this is self-explanatory. Running the piece (program) through allows for the consumate test of all of the above. Any flaws or gaps can be detected providing the auditor does not have such a distorted sense of objectivity by this point that it either sounds like garbage or a masterpiece.

It is well beyond the scope of an article this size to go into any depth regarding hardware, software, or for that matter the end result of their interaction with the creative force. Whether one uses a computer assisted, analog-digital system wih its advantage of 'real time' control and disadvantage of analog temperament or a full digital system with its advantage of speed and accuracy and disadvantage of expensive or difficult expansion (and in some cases limited software), the final result should and must be the most important aspect.

Music has always been, and still is, in the process of defining itself. Is it merely a set of varying frequencies/waveforms (melody) accompanied by vertical frequency aggregates (harmony) working at the micro-time level of rhythm and macro-time level of structure? One can only hope that the use of the computer will not only make it easier to muster and expand the resources of the medium but, just possibly, help to expand the very definition of the word music.

Programmers Do It In Software

By Hal Chamberlin, Raleigh, NC

(3 articles combined)

Like virtually any task a computer is called upon to perform, sound and music synthesis can be accomplished by purely software means as well as with dedicated hardware. This is seen every day in the personal computer field, since one of the main reasons microcomputers are considerably less expensive than the commercial monsters is that most everything is accomplished in software. For example, the floating point arithmetic performed by BASIC is handled by a set of machine language sub-routines instead of a hardware floating point processor. Likewise, all of the essential functions of the well-known SID (Sound Interface Device) chip in the Commodore 64 can be performed by appropriate software routines in models that lack the SID (such as VICs and of course PETs). As you will see, it is even possible to **exceed** the SID's capability in many respects. The techniques to be described are applicable to any 6502-based computer (Apple, Atari, MTU, Ohio Scientific, Mattel, etc.), and even computers based on 6800, Z-80, 8088, 68000 and other processors.

CONS AND PROS

Before discovering how it is possible to synthesize high-quality, multi-voice music by purely software means, let's take a brief look at the advantages and disadvantages involved. With a hardware synthesizer (and the SID is only one of several around), the circuit designer has already done most of the work for you. Thus, in general, a hardware synthesizer will be much easier to program. In the case of the SID, a few well-chosen POKEs from BASIC are all that are needed to get some kind of sound out. Using the software technique, on the other hand, the programmer must first "build the synthesizer" in software and then write more software to control it. Furthermore, this software must be in machine language to meet speed requirements. However, for the users of "canned" programs, there may be little difference between the two techniques in this regard.

The more advanced software techniques that are capable of simulating standard instruments with a surprising degree of realism, as well as an infinite variety of others, require a lot of memory for storing waveform tables. Effective use of these advanced techniques requires a minimum of 16K with upwards of 32K to 64K being desirable. While this may not be much of a problem for C64 owners or most PET owners, unexpanded VIC owners will be limited to the simpler software techniques that use less memory.

Perhaps the most serious drawback of software synthesis is that it "consumes the entire machine" while sound is being generated. In other words, the sound-generating software routines must be in full control every microsecond while sound is being generated. Any break to look at the keyboard or a joystick or to put something

on the screen results in a break in the sound. This, therefore, makes "playing" the computer from a music keyboard a difficult, although not impossible, programming task, particularly if key clicks are unacceptable. As a result, most of the work done with software synthesis involves "coding" a song from sheet music into a music compiler or directly into memory and then having the computer play the encoded song.

On the plus side, the software technique can be considerably more flexible than existing hardware synthesis chips. For example, the SID chip has available only three fundamental sound waveforms to choose from, although the filter can be used to modify them in a number of ways. The software technique, on the other hand, uses **waveform tables** stored in memory to define the waveform desired, which may be literally any curve that can be defined by 256 (a typical value for available software) independent points. Also, the SID has relatively few (16) attack and decay rates to choose from. The software technique, on the other hand, has a virtually unlimited number of choices, plus the attack and decay envelope may be any shape desired, not just single straight lines approximating a standard ADSR shape. Finally, for machines using 1MHz 6502 processors as is the case with all Commodore computers, the software technique yields **four** voices instead of three!

There are other pros and cons that will become apparent later, but in all cases they are simply tradeoffs among various factors. A clear "winner" that applies in all cases does not exist.

HARDWARE NEEDED

Actually, it's probably wrong to claim that this is a **purely** software technique since some specialized hardware actually is needed. Those buzzy old single-voice Altair programs that played through a radio held up to the computer were purely software. The device needed is called a digital-to-analog converter, or DAC for short. It does just what its name implies; it converts digital information from the computer (numbers) into analog information (voltages) that is acceptable to an amplifier and speaker system. A DAC is really nothing more than a programmable power supply connected to the computer. When the program sends a value such as +5.32 to the DAC, it produces a voltage level of +5.32 volts at its output which persists until another value is sent to it. A changing voltage level is therefore produced by arranging for the driving program to send a series of numbers to the DAC, each representing a point along the desired voltage contour. An electrical audio signal, which is nothing more than a very rapidly varying voltage level, is thus produced by very rapidly sending numbers to the DAC. Note that the "raw" DAC output is a stair-step approximation to the smooth audio waveform desired. In practice, the DAC must be followed by a low-pass filter to average out these steps and produce a smooth waveform. Figure 1 illustrates this process.

Unfortunately, most personal computers do not come with a DAC standard, although a few do such as the MTU-130 and the Ohio Scientific C8P. This is a shame because a DAC is a very inexpensive circuit to include in a computer's design adding perhaps two to five dollars to the manufacturing cost. Nevertheless, a DAC may be easily added to Commodore machines through the User Port and other computers having a parallel I/O port.

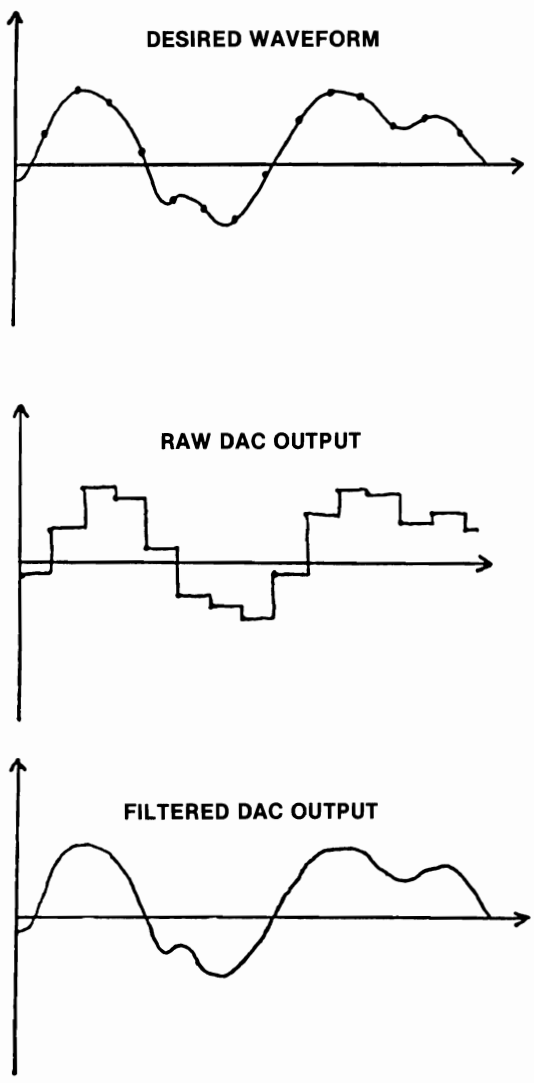


FIGURE 1: Operation of D-to-A Converter

DIGITAL AUDIO FUNDAMENTALS

In Hi-Fi magazines today, "digital audio" is the current rage just like quadraphonics was ten years ago. A digital audio recording system is simply an analog-to-digital converter (ADC) which converts the incoming audio waveform into a string of numbers, a digital mass storage device such as a tape or disk drive, and a DAC for playing the numbers back. (see figure 2). What is actually being done in a software synthesis system is to tap into this chain in the middle somewhere with a computer. For the "real time" system that will be described, the tap is just before the DAC where the computer and suitable software generates the numbers for the DAC rather than a mass storage device. The tap can also be made **before** the mass storage device which yields a "delayed playback" system. Since a digital audio record/playback system can obviously handle **any** kind of sound, it is apparent that, at least theoretically, a software synthesis system can synthesize **any** kind of sound.

In a digital audio system, two operating parameters work together to determine the system's sound quality. Obviously the speed at which numbers are sent to the DAC determines how much detail in the reproduction of fast waveform wiggles that may be achieved. In audio terms, it determines the **high frequency response**. Theoretically, frequencies up to 1/2 of the "sample rate", which is the rate at which numbers (samples) are sent to the DAC in samples per second, may be reproduced. Thus, if 20,000 samples are sent to the DAC every second, frequencies up to 10KHz may be reproduced. You should note that just above 1/2 of the sample rate lies a considerable quantity of distortion frequencies which must be filtered out by the low pass filter mentioned earlier. It takes a very sharp filter to separate the desired signal from the distortion but this sharpness requirement is relaxed somewhat if frequencies up to only about 40 percent of the sample rate are attempted. One interesting fact is that a digital audio system has no low frequency limit (it goes down to 0Hz) so bass response is excellent regardless of the sample rate.

The other operating parameter is the amount of round-off error in the DAC and the samples sent to it. Such round-off error gives rise to a general background noise level at all frequencies and therefore cannot be filtered out. Usually the DAC itself sets the system's numerical precision which is measured in bits. An "8-bit DAC" for example accepts 8 bit samples which have an inherent round-off error of + or - 1/512 or + or - 0.2 percent. The noise level associated with this amount of error is approximately 20*LOG10(1/512) + 6 decibels or -48dB. A 12 bit DAC yields -72dB and a 16 bitter, which is typically used in professional digital audio applications, gives -96dB. In more familiar terms, 8 bits is about the noise level of a cheap cassette recorder, 12 bits is about the noise level of a first rate consumer audio system, and 16 bit noise is difficult to even measure. Note that while the bit precision of the DAC determines the ultimate minimum noise level, if the calculations that produce the samples are sloppy, they may introduce excess noise.

Software music systems on 1MHz 6502 microprocessors generally use 8 bit samples and sample rates between 8 and 9KHz. The DAC precision of course is set by the system's 8 bit word size while the sample rate is determined by how fast existing music programs can calculate the samples for an acceptable number of voices. The 6502 actually does this sort of thing very well; it would take a 5MHz Z-80 (with no memory wait states) to perform as well. In a delayed playback system in which samples are written to a mass storage device instead of being output immediately, the DAC precision, sample rate, and number of voices can be increased to professional levels since computation speed is no longer a limiting factor. However in such a system you may have to wait hours for the program to compute just a few minutes of sound.

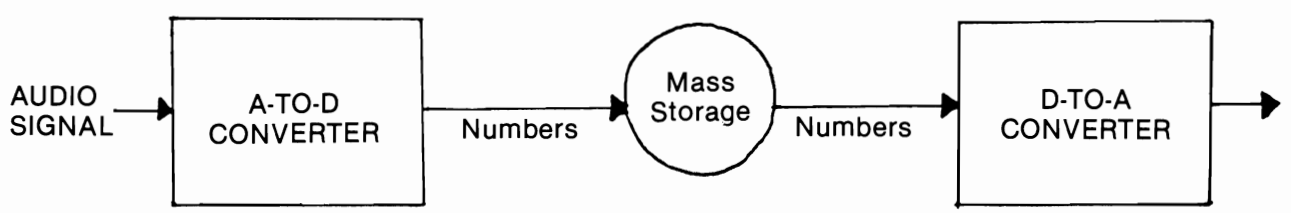


FIGURE 2: Digital Audio Recording System

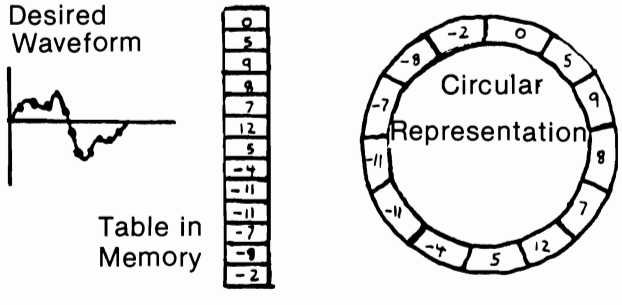
COMPUTING THE SAMPLES

Thus the problem of synthesizing sound with a DAC has been reduced to that of computing waveform sample values and quickly outputting them to the DAC. Two constraints, however, keep this from being a simple, straightforward programming task. One difficulty is that reasonably high sample rates, such as 8KHz, allow very little time, such as 125 **micro**seconds, for computing a sample. When synthesizing multiple voices, such as 4, the amount of time spent on each voice has to be around 30uS. Not only must machine language be used, but it must be highly optimized machine language implementing an efficient synthesis algorithm. The other difficulty is that the time between samples must be absolutely consistent from sample-to-sample; even a 1 microsecond variation will introduce more noise than 8-bit D-to-A conversion does. On some machines such as Ataris, the display may have to be turned off to allow the processor to run at a uniform speed.

The secret to meeting these constraints at all is the use of **waveform tables** stored in memory. Before the music is played, these tables are filled with values representing the waveshape for each instrument sound desired. The filling process is **not** subject to the constraints listed above so straightforward machine programming or even BASIC can be used to fill the tables. More on this later.

Assuming that a filled waveform table is available, all that is necesary to "play" the waveform is to write a short loop that **scans** the table one entry at a time and outputs each entry to the DAC. Since musical waveforms are periodic (repetitive), when the table has been scanned from beginning to end, you "wraparound" to the beginning and scan it again. Each scan through the table will produce one cycle of the tabulated waveform. The number of scans accomplished in 1 second will then determine the fundamental frequency (pitch) of the generated tone. Figure 3 illustrates the waveform table concept. Note that wraparound from end to beginning can be visualized as bending the table into a circle.

FIGURE 3. Waveform Tables



If the table is exactly 256 bytes long, then the machine code routine in figure 4 will do the scanning job for a single voice. Note the use of the indexed addressing mode for accessing the table. The Y register acts as a **pointer** into the waveform table. When the Y register increments from 0 and reaches 255, the next increment will "overflow" back to zero thus providing the wraparound. It is helpful to visualize the pointer as going around and around the circular table "peeling off" a waveform cycle each time around.

FIGURE 4. Simple 1 Voice Waveform Table Scan Routine

```
LDY #0              INIATIALIZE Y
LOOP LDA WVTAB,Y    GET A WAVEFORM TABLE ENTRY
STA DAC             STORE IT IN THE DAC
INY                 MOVE THE WAVEFORM TABLE POINTER
JMP LOOP            CONTINUE SCANNING FOREVER
```

In figure 5A is a routine for producing 4 voices by simultaneously scanning 4 different tables. Every "sample period" (time around the loop), a sample value is read from each table and added up to create a single sample which is sent to the DAC. Thus the mathematical operation of **addition** is equivalent to the audio operation of **mixing** several signals into one. Note that the waveform table entries need to be scaled so that when 4 of them are added up, overflow does not occur. Figure 5B shows an alternate method of mixing multiple voices. Here, each sample is output to the DAC as soon as it is taken from the table. The mixing actually takes place in the DAC's low-pass filter. For this to work properly, the "dwell time" of each voice's sample in the DAC should be approximately equal; If they are unequal, the voice dwelling longer will sound louder. One advantage of the 5B method is that the table entries do not need to be scaled (this results is less noise) and one disadvantage is that the loop is slower due to the 3 extra STA instructions.

FIGURE 5A. Four Voice Waveform Table Scan with Mixing by Addition

```
LDY #0              INITIALIZE Y
LOOP LDA WVTAB1,Y   GET VOICE 1 WAVEFORM TABLE ENTRY
CLC
ADC WVTAB2,Y        ADD IN VOICE 2
ADC WVTAB2,Y        ADD IN VOICE 3
ADC WVTAB2,Y        ADD IN VOICE 4
STA DAC             STORE THE SUM IN THE DAC
INY                 MOVE THE WAVEFORM TABLE POINTER
JMP LOOP            CONTINUE SCANNING FOR EVER
```

FIGURE 5B. Four Voice Waveform Table Scan with Time Division Mixing

```
LDY #0              INITIALIZE Y
LOOP LDA WVTAB1,Y   GET VOICE 1 WAVEFORM TABLE ENTRY
NOP                 EQUALIZE DWELL TIME AMONG THE
                    VOICES
STA DAC             OUTPUT VOICE 1 TO THE DAC
LDA WVTAB2,Y        GET VOICE 2
NOP                 EQUALIZE
```

```
STA DAC          OUTPUT VOICE 2 TO THE DAC
LDA WVTAB3,Y     GET VOICE 3
NOP              EQUALIZE
STA DAC          OUTPUT VOICE 3 TO THE DAC
LDA WVTAB4,Y     GET VOICE 4
INY              MOVE THE WAVEFORM TABLE POINTER
STA DAC          OUTPUT VOICE 4 TO THE DAC
JMP LOOP         CONTINUE SCANNING FOR EVER
```

Now that waveforms can be output at adequate speed (the figure 5 routines are actually faster than necessary), the next problem is to control the **pitch** of the tones. Since the pitch is determined by how rapidly the table pointers go around the tables, there are two obvious ways to alter the pitch: change the loop speed by inserting NOPs or change the table length. The former is undesirable for a number of reasons, particularly for multiple voices, so varying the table length will be used instead. The **effective** table length is actually determined by two factors. One factor is the actual length which so far has been 256 bytes. It is highly desirable to retain this length because it greatly simplifies (that means speeds up) the table lookup and handling of wraparound.

The other factor is how far the pointer advances each sample period. The figure 5 routines always advanced the pointer by 1 but if they are rewritten to advance it by 2 instead, the effective table length becomes 85 1/3 (no problem) and so on. The amount the pointer is advanced in each sample period will be called the table pointer **increment**. Figure 6A shows the figure 4 routine modified for a variable increment.

When the table pointer increment is restricted to integer values, only a few different pitches are available for a given table length. For example, the figure 6A routine can produce only the following frequencies (1MHz clock assumed): 195.3Hz, 390.6, 585.9, 781.2, N*195.3Hz where N is the increment. Besides being very far apart (the first two are 12 notes apart!), few of the possibilities fall very close to standard music note frequencies. The solution to this dilemma is to allow **non-integer** table pointer increments. With non-integer increments, any frequency desired may be produced. Their use also implies that the table pointer itself will have a fractional part which calls for **interpolation** between adjacent waveform table entries. Unfortunately, to actually perform linear interpolation requires a substantial amount of extra code and one (ugh) multiplication to be performed. While the newer 16 bit processors may be able to do this rapidly enough, it is necessary to just ignore the fractional part of the table pointer when wavetable lookup is actually performed on an 8 bit micro. Note that this compromise does introduce some excess noise which is called **interpolation noise**.

Figure 6B shows a waveform table scan routine modified for non-integer table pointers and increments. Both the pointer and the increment are now 16 bit values with an 8 bit integer part and an 8 bit fractional part. Note that although the fractional part of the pointer (PTRF) does not participate in the lookup operation, it must be maintained in the calculation for things to work out right. With this routine, frequencies at any multiple of .526Hz may be produced which allows a very close approximation to standard note frequencies.

FIGURE 6A. Single Voice Waveform Table with Variable Increment

```
LDY #0               INITIALIZE THE POINTER
LOOP LDA WVTAB1,Y    GET A WAVEFORM TABLE ENTRY
STA DAC              STORE IT IN THE DAC
TYA                  ADD THE INCREMENT TO THE POINTER
CLC
ADC INCR
TAY
JMP LOOP             CONTINUE SCANNING FOR  EVER
```

FIGURE 6B. Single Voice Waveform Table Scan with Fractional Increment

```
LDY #0               INITIALIZE THE POINTER
STY PTRF
LOOP LDA WVTAB1,Y    GET A WAVEFORM TABLE ENTRY
STA DAC              STORE IT IN THE DAC
LDA PTRF             ADD THE INCREMENT TO THE POINTER,
CLC                  FRACTIONAL PARTS FIRST
ADC INCF
STA PTRF
TYA                  THEN THE INTEGER PARTS
ADC INCI
TAY
JMP LOOP             CONTINUE SCANNING FOR EVER
```

Figure 7 shows the full-blown 4-voice synthesis loop that is actually used in existing DAC synthesis programs with perhaps minor modifications. Note that the 4 waveform table pointers (one for each voice) are now kept in memory instead of the Y register (they are all different since the 4 voices will probably have different frequencies) and that indirect addressing through these pointers is used to do the table lookup. Each pointer actually consists of 3 bytes: a constant part that holds the page address of the waveform table, an integer part that points to a particular byte in the table, and a fractional part that is not used for lookup but does participate when the increment is added. Each increment consists of 2 bytes: an integer part and a fractional part according to the pitch desired for the corresponding voice.

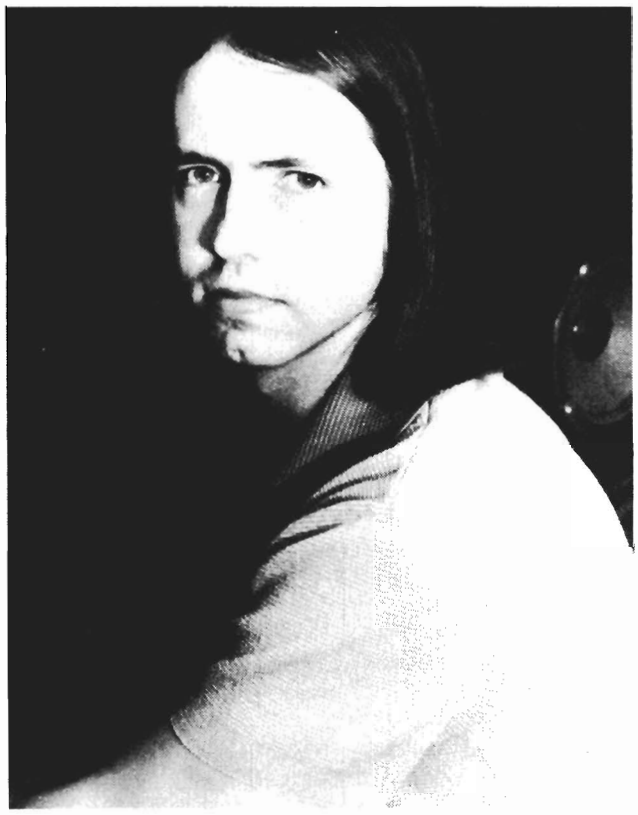
Also note that, unlike previous routines, there is provision for the tone generation to stop after a specified number of sample periods. This number is effectively the product of the 8 bit values TEMPO and DUR. In a music program, TEMPO normal-

ly remains constant or is changed occasionally while DUR is set according to the desired duration of the 4 voice chord. Note the time equalizing instructions at the end of the loop to prevent sample period jitter when TEMPO is decremented to zero and must be reloaded. Also note that CLC instructions are omitted prior to adding the increments to the pointers. Besides saving 8 microseconds, this action introduces a slight frequency error wnich livens up the sound significantly.

FIGURE 7. Complete 4-Voice Sound Generation Routine

```
PLAY    LDY #0          ; SET Y TO ZERO FOR STRAIGHT INDIRECT
        LDX TEMPO       ;SET X TO TEMPO COUNT
                        ; COMPUTE AND OUTPUT A COMPOSITE SAMPLE

PLAY1   CLC             ; CLEAR CARRY
        LDA (V1PT+1),Y  ; ADD UP 4 VOICE SAMPLES
        ADC (V2PT+1),Y  ; USING INDIRECT ADDRESSING THROUGH VOICE
        ADC (V3PT+1),Y  ; POINTERS INTO WAVEFORM TABLES
        ADC (V4PT+1),Y  ; STRAIGHT INDIRECT WHEN Y INDEX =0
        STA X'1700      ; SEND SUM TO DIGITAL-TO-ANALOG CONVERTER
        LDA V1PT        ; ADD INCREMENTS TO POINTERS FOR
        ADC V1IN        ; THE 4 VOICES
        STA V1PT        ; FIRST FRACTIONAL PART
        LDA V1PT+1
        ADC V1IN+1
        STA V1PT+1      ; THEN INTEGER PART
        LDA V2PT        ; VOICE 2
        ADC V2IN
        STA V2PT
        LDA V2PT+1
        ADC V2IN+1
        STA V2PT+1
        LDA V3PT        ; VOICE 3
        ADC V3IN
        STA V3PT
        LDA V3PT+1
        ADC V3IN+1
        STA V3PT+1
        LDA V4PT        ; VOICE 4
        ADC V4IN
        STA V4PT
        LDA V4PT+1
        ADC V4IN+1
        STA V4PT+1
        DEX             ; DECREMENT & CHECK TEMPO COUNT
        BNE TIMWAS      ; BRANCH TO TIME WASTE IF NOT RUN OUT
        DEC DUR         ; DECREMENT & CHECK DURATION COUNTER
        BEQ ENDNOT      ; JUMP OUT IF END OF NOTE
        LDX TEMPO       ; RESTORE TEMPO COUNT
        BNE PLAY1       ; CONTINUE PLAYING
TIMWAS  BNE *+2         ; 3 WASTE 12 STATES
        BNE *+2         ; 3
        BNE *+2         ; 3
        BNE PLAY1       ; 3 CONTINUE PLAYING
ENDNOT  RTS             ; RETURN
                        ; TOTAL LOOP TIME =114 STATES =8770 HZ
    ; THE FOLLOWING VARIABLES SHOULD BE IN PAGE ZERO

V1PT    .BYTE 0         ; VOICE 1 WAVE POINTER, FRACTIONAL PART
        .WORD WAV1TB    ; INTEGER PART AND WAVE TABLE BASE
V2PT    .BYTE 0         ; VOICE 2
        .WORD WAV2TB
V3PT    .BYTE 0         ; VOICE 3
        .WORD WAV3TB
V4PT    .BYTE 0         ; VOICE 4
        .WORD WAV4TB

V1IN    .WORD 0         ; VOICE 1 INCREMENT (FREQUENCY PARAMETER)
V2IN    .WORD 0         ; VOICE 2
V3IN    .WORD 0         ; VOICE 3
V4IN    .WORD 0         ; VOICE 4

DUR:    .BYTE 0         ; DURATION COUNTER
TEMPO   .WORD 82        ; TEMPO CONTROL VALUE, TYPICAL VALUE FOR
                        ; 3:4 TIME, 100 BEATS PER MINUTE, DUR=64
                        ; DESIGNATES A QUARTER NOTE
```

HAL CHAMBERLIN



A MORE ADVANCED TECHNIQUE

The foregoing technique and core sound generation routine was first published in Byte magazine in 1977 and has been the basis for numerous music programs on 6502 as well as other processors. Although they were playing with this kind of thing in the early '60s at Bell Labs, it was quite a breakthrough in the micro world at the time. As is though, it has some limitations. In particular, no matter what waveforms you put into the tables, the music is always organ-like; just an infinite variety of stops. The reason for this is that the **amplitude envelope** of the tones produced is always rectangular, that is, off-on-off, just like pressing an organ key. Even though you may be able to obtain the waveform or harmonic structure of a familiar instrument such as a piano, the rectangular envelope will impart an organ-like character.

The usual way of adding an envelope to a synthesized tone is to use a gain-controlled amplifier in which the gain (volume) is varied according to the desired contour. For a piano note, the gain would suddenly go from zero to maximum for the attack and then slowly decrease back to zero for the decay. The tone input to the amplifier would be constant. In a digital audio system, gain con-

trol is usually accomplished by **multiplying** waveform samples by a gain factor. As before however, actual multiplication is too slow on an 8 bit micro to consider.

Another factor that contributes greatly to tonal variety is the fact that the waveform of most instruments is **not** constant during a note. In a typical case, such as a trumpet note, the tone is "brighter" (greater proportion of high harmonics) during the attack than during the decay. Any attempt to synthesize a trumpet tone with a constant waveform yields a flat sound without the characteristic "toot" of trumpet notes. In addition, most "novelty sounds" for which a computer music system would be expected to be good at have very prominent waveform shifts during the sound.

Both of these desirable characteristics can be added to the software system by using a scheme first proposed by Frank Covitz and Cliff Ashcraft, long-time PET and AIM-65 owners respectively. The idea is to use a **sequence** of many waveform tables, each differing slightly from its neighbor in both harmonic content and overall amplitude. By reconstructing the core sound generation loop somewhat and using the time "wasted" when TEMPO is not reloaded, it is possible to periodically change the third byte (the waveform table page number) of the pointers noiselessly for a smooth shift from one table to the next. For additional flexibility in programs that actually use this technique, another set of tables, called waveform **sequence** tables, specify a list of waveform table addresses so the sequence of wave tables actually played need not be consecutive in memory. Additionally, the sequence tables allow sequencing through wave tables rapidly when the envelope is changing rapidly, and more slowly at other times, thus conserving wave tables and memory. The power of the 6502 instruction set really shows in the double indirect-indexed addressing required to implement this idea.

Although the scheme just described really didn't look very promising on paper, the results when actually implemented in 1979-1980 were spectacular. Residual noise when switching from table to table was less than expected and fewer tables were needed for smooth-sounding envelopes than were expected. It was found that generally 16 to 32 tables requiring 4K to 8K was sufficient for most instrument sounds. Thus, in a 32K machine, there is sufficient space for 3 to 6 "instrument definitions" with 8K left for the score and music playing program. Experiments with published analyses of instrument sounds, such as string, horn, and piano tones, produced surprisingly accurate reproductions within the 3.5KHz frequency limits of the system. At the opposite extreme, the oddball sounds never stopped; just about anything that was put into the waveform computation routine produced some kind of unique tone color.

FILLING THE WAVEFORM TABLES

So far nothing has been said about actually filling the waveform tables with data representing desirable sounds. In theory, just about any list of numbers will produce a recognizable tone when scanned but the sound is likely to be raucous and grating.

One obvious method is to draw one cycle of the waveform on graph paper and then laboriously read off 256 sample values and enter them into the table. The drawn shape could come from an oscilloscope photo of a musical instrument sound or from imagination. Besides the effort involved, the drawn shape must span exactly 256 grid lines in exactly one cycle to be valid. One could also make use of a light pen or graphic digitizer in conjunction with a drawing program to do the same thing with much less effort. The biggest problem when using imagination is that there is no simple relation between the appearance of the drawn shape and the resulting tone color. Thus, if a particular shape produces a sound that is close to what is desired, there is no way to know what must be changed to make it sound even closer.

Probably the best way to fill waveform tables is to write a program that accepts harmonic specifications, computes the corresponding waveshape, and automatically enters it into memory. There is a very definite correlation between the harmonic makeup of a tone and its timbre. One can also occasionally find published harmonic analyses of musical instrument tones, particularly organ pipes. Figure 8 shows the listing of a very simple BASIC program that can be used to create waveform table data and POKE it directly into memory. The statements, starting at line 2000, first amplitude normalize the waveform, convert the samples into integer form in the range of 0 to 63 (to avoid overflow when 4 are added up) and then poke them into memory.

If you can't figure out what the HEX is wrong with your M.L. program, leave it in first CRASH position.

Ylimaki

FIGURE 8 Waveform Table Fill Program in BASIC

```
1000 REM WAVEFORM TABLE FILL PROGRAM
1001 REM SELECT RANDOM OR SPECIFIED PHASE REM
1002 REM ENTER HARMONIC NUMBER FOLLOWED BY
     RELATIVE AMPLITUDE
1003 REM HARMONIC NUMBER =0 FILLS THE TABLE AND
     EXITS
1010 DIM W(255): Z=6.283185/256
1020 FOR I =0 TO 255: W(I) =0: NEXT I
2000 PRINT "RANDOM PHASE ANGLES? (Y/N) ";: INPUT A$
2010 PRINT "ENTER HARMONIC NUMBER ";: INPUT N
2020 IF N =0 GOTO 3000
2030 PRINT " ENTER RELATIVE AMPLITUDE ";: INPUT A
2040 P =RND(1)
2050 IF A$ ="Y" GOTO 2070
2060 PRINT "ENTER PHASE ANGLE ";: INPUT P
2070 P =6.28318*P
2080 FOR I =0 TO 255: W(I) =W(I)+A*SIN(N*I*Z+P): NEXT I
2090 GOTO 2010
3000 M =0
3010 FOR I =0 TO 255
3020 IF ABS(W(I))>M THEN M =ABS(W(I))
3030 NEXT I
3040 M =M+.00001: REM MAKE ALL TABLE ENTRIES<1.0
3050 A =0
3060 FOR I=0 TO 255
3070 W(I) =W(I)/M
3080 A =A+W(I)*W(I)
3090 NEXT I
3100 PRINT "RMS AMPLITUDE IS "; SQR(A/256)
9999 STOP
```

The biggest advantage of using harmonics to specify waveforms is that **alias distortion** can be readily avoided. Alias distortion occurs whenever any frequency **component** of a waveform exceeds 1/2 of the sampling frequency. This can easily happen with high notes using waveforms rich in harmonics. For example, if one attempts to play high C (523Hz) using a waveform with 10 significant harmonics through an 8KHz sample rate system, the 8th, 9th, and 10th harmonics will alias since they will all be above 4KHz. Aliasing means that intended frequencies are altered ("reflected" off the 1/2 sample rate ceiling) and usually produces an objectionably harsh sound. Thus waveform tables used to play high notes should have their upper harmonics restricted while those for low notes may have dozens of significant harmonics if desired.

Although the figure 7 program can be used to compute waveform tables, most of the DAC synthesis music programs available for 6502 computers include machine language routines for computing waveforms from harmonic specification in much less time (typically less than 1 second per table). The more advanced programs using sequences of waveform tables actually let you specify an amplitude envelope for **each individual harmonic** as a series of straight-line segments as in Moorer's published analyses. The program then will compute a whole series of tables automatically from just the envelope specifications.

CONCLUSION

Although the discussion of software music synthesis has necessarily been brief, I hope that it is now apparent that purely software synthesis chips are currently available, at least on 6502 processors. The techniques presented are being further refined on the 6502 based MTU-130 computer (a full-blown music compiler is now available for that system) and being extended to the 68000 microprocessor which among the new 16 bitters is best at synthesis calculations. Progress is being made in delayed playback synthesis using 8" floppy disks for sample storage which has the potential for professional sound quality. Following is a list of references for further study into this fascinating software area.

REFERENCES

Refer to the following articles for a more detailed description of software synthesis and additional sample routines and programs.

1. Chamberlin, Hal, "A Sampling of Techniques for Computer Performance of Music", September, 1977, BYTE.

2. Chamberlin, Hal, "Advanced Real-Time Music Synthesis Techniques", April, 1980, BYTE.

3. Chamberlin, Hal, "Simulation of Musical Instruments", January, 1981, Kilobaud Microcomputing.

4. Chamberlin, Hal, "Software Techniques of Digital Music Synthesis", April, 1981, Creative Computing.

5. Moorer, J. and J. Grey, "Lexicon of Analyzed Tones", Computer Music Journal, vol. 1, and succeeding issues, MIT Press, Cambridge, MA.

6. Mathews, Max, "The Technology of Computer Music," MIT Press, Cambridge, MA, 1969.

7. Chamberlin, Hal, "Musical Applications of Microprocessors", Hayden Book Co., Rochelle Park, NJ, 1980.

Note: Reprints of references 1 and 2 and copies of reference 7 are available from Micro Technology Unlimited, Box 12106, Raleigh, NC 27603, USA. DAC boards and the more advanced music program for PET computers are also available from MTU.

Making Friends With SID

By Paul Higginbottom, Toronto, Ont.

(2 articles combined)

The synthesizer chip in your Commodore-64 computer is affectionately known as Sid. Sid is in fact an acronym for 'S'ound 'I'interface 'D'evice. I doubt that many people realize just how powerful this chip is, but I intend to unleash some of its power for you. If you read some of the documentation for the Commodore-64 about its sound capabilities and are new to (as I was) synthesizer jargon, you probably thought to yourself, "I'm never going to figure that out!" Well, I am the sort of person who gets more determined to figure something out, when it seems harder than ever to do so. So, step by step, I, like any beginner, set about learning how to control the Sid's sound capability.

THE JARGON

If your mind is like mine and tends to go blank when confronted with a barrage of alien jargon about something, then hopefully I can gently "break you in" with the terms associated with music synthesis using Sid.

The Sid chip is comprised of three sections essentially:

1)Oscillator section
2)Envelope section
3)Filter section

There are a few other bits and pieces, but more on those later.

Sid has three "voices". That means to you and me, that up to three tones can be played at the same time.

Each voice is separately combined by its "frequency" (the pitch of the tone), and more importantly, its "envelope".

The envelope of a voice, determines how its volume rises, sustains, and falls, like a musical instrument, or other sounds we hear in our lives. For example, a violinist will perhaps play a note by pulling the bow across a string slowly at first (the volume starting out low), and as the player starts to increase the speed and pressure of the bow on the string, so the volume increases, and as the player ends the note, he or she slows the rate and pressure of the bow again, and the volume fades away to silence. With a single violinist, the tone may fade away rather abruptly, but I'm sure you've heard this rising and falling effect of volume, with a piece of orchestration (many string instuments).

That is one example of an envelope. If we consider another example to allow you to grasp different types, think of hitting a cymbal.

The rise to its maximum volume is almost instant, as the CRASH of the cymbal begins and, from that point, the sound simply fades away slowly to silence again. An example of a cymbal type of sound that does rise slowly first and then fade away would be a wave approaching the beach. You hear the slowly increasing volume of the wave moving up the beach, then as the wave trips over itself and hits the beach the loudest part of the noise is heard, and then the sound fades away as the wave slides up the beach, and the next one approaches again.

Well, enough of the examples, back to the technical stuff. This "behaviour" of the volume (or "amplitude") of a voice, can be defined in 4 parts, and this terminology is common amongst professional synthesizers costing many times the prices of your Commodore-64 computer!

THE FOUR PARTS OF AN ENVELOPE

You may have noticed by now, that to define this changing in volume, we simply need to define the TIME it takes for a sound to go from one volume, to get to another volume. For example, the violin might have taken half a second to go from no volume (silence) to its maximum volume, and then 2 seconds to fade away again (silence again). The cymbal took no time to reach its maximum volume (starts with the CRASH), but 10 seconds to fade away. The wave is different again, in that it might take 5 or so seconds to build up to maximum volume (as it moves up the beach), and then only 1 second to die away (as the wave falls over and crashes on the beach).

Part 1 — ATTACK — This is the time taken to go from silence (0 volume) to the maximum volume Sid is set to.

Part 2 — DECAY — This is the time taken to go from the maximum volume Sid is set to, to a given "mid-point" volume, or: sustained level of volume.

Part 3 — SUSTAIN — This is not a time value, but is a level of volume the voice sustains at after the ATTACK and DECAY.

Part 4 — RELEASE — This is the time taken to go from the sustained volume to silence once again.

In those definitions, I mentioned "the maximum volume Sid is set to", and that is the maximum overall volume (just like the overall volume control on your television or stereo).

HOW WE CONTROL SID

Before I go any further, I want to explain how we actually tell Sid exactly what weird and wonderful sounds we want it to make (so we can drive everyone crazy!)

The Sid chip has an amount of memory in it, and simply by putting numbers into those memories, we give Sid all the information it needs to produce an infinite number of sounds. We put numbers into memories with the BASIC command POKE. We give the poke command two numbers; the memory number (or "address"), and the number we want to put into that memory (one memory location can hold any whole number between 0 and 255).

Sids address is quite a big number. He starts at 54272, and he occupies that memory location and the next 28 also, up to 54300.

I want to show that it really is **not** that difficult to train Sid, and that you **don't** have to be a genius at programming.

MAKING YOUR FIRST BEEP

To make a noise, we must do 4 things:

1)Set the maximum overall volume

2)Set the envelope of the voice we wish to use

3)Set the frequency of the voice to the desired pitch

4)And only then "tell" Sid to do it.

I put quotes around "tell" in part 4, because I want to examine that closer. When we tell Sid to make a sound, we tell it to firstly do the ATTACK (rise up to maximum volume) and then the DECAY (go down) to a SUSTAINed level of volume. When we tell Sid to do that part, the noise will stay at the SUSTAINed level of volume forever if you don't tell it to go on, and do the last part; the RELEASE (go down from the sustained level of volume, to nothing).

So to recap, we tell Sid to do the ATTACK-DECAY-SUSTAIN part first, and then when we're ready, we tell it to finish the envelope with the RELEASE part.

You could get a person to demonstrate this for you. Ask them to take a DEEP breath when you tap them on the shoulder and then hum a note, at first quietly, building up to a loud level, and then going down to a comfortable level. You have made a person do the Attack-Decay-Sustain part of an envelope. I said the sound will continue indefinitely if you don't tell it to release, so when you tap the person again on the shoulder, they can slowly quieten their hum down to nothing. Of course, if you decide to make them sustain for too long, they'll go blue in the face, and pass out!

Also, you may want them to stop before they get to the release, because their hum is so obnoxious! (Fortunately, you can also do this with the Commodore-64!)

For now, let's just concentrate on ONE voice. Each voice has 7 memories inside Sid, to control it. Voice 1's memories are in fact, the first 7 memories, voice 2, the second 7, and voice 3, the next 7. That, if you've been doing your math, is the first 21 memories in Sid. The other 8 (there are 29 in all) are for the filter section which I haven't talked about yet, and other bits and pieces, including the overall volume control which I have mentioned).

The 7 memories for each voice are all organized the same way, for example, the first two of each block of 7, control the frequency (pitch) of the voice.

THE SEVEN MEMORIES FOR A VOICE

The first two as I just mentioned, control the frequency of the voice, that is, the pitch of the sound.

The second two are to control one particular type of sound, which will be covered later

The fifth memory is the controlling memory of the voice, the one which will tell Sid to start the note, stop it, and choose the type of sound.

The sixth memory controls the duration of the Attack and Decay.

The seventh memory controls the Sustain level, and the Release time.

The fifth of the seven I just described, I will now explain further. I mentioned there that apart from telling Sid to play the envelope, it also controls the type of sound. (Another piece of jargon coming up!) The type of sound is known as the

"Waveform". You are probably aware that sound is comprised of air being compressed and stretched. By, for example, a speaker cone, which moves in and out, and the speed (the FREQUENCY) at which it moves in and out, determines the pitch.

The way in which air is compressed and stretched is cyclic (repeats itself), and this cycle is known as the waveform.

Sid allows you to choose from one of four waveforms, and it is the fifth of the seven memories in each voice, which you set to tell Sid which waveform you wish to use.

THE WAVEFORMS

Triangular (shaped like this:) This waveform, due to its smoothness, produces a mellow, soft, flute-like sound (very pleasant to the ear!).

Sawtooth (shaped like this:) This waveform, due to its abrupt ending produces a brighter, brass-like sound.

Variable width pulse (shaped anything from this: , to this: , or this: [which if you look is simply the first one upside-down]) This waveform as you can see from the description in parentheses can be varied, but is essentially an ON and OFF waveform, and as such is very abrupt and produces anything from a hollow, organ-like sound, to a very quiet, reedy sound. As you can see from the symbols of this waveform, it is comprised of pulse, or varying widths (hence the name), and memories 2 and 3 which I mentioned earlier were for a specific type of sound, do in fact control the width of the pulse when this waveform is chosen. Of course, memories 2 and 3 have no effect when any other waveform is selected.

Noise — I won't try to do a little drawing of this waveform, since it is in fact a RANDOM waveform, and has no defined harmonic qualities, but because the frequency can be altered, will produce any sound from a hiss (like you hear from poor quality cassette recorders), to a low rumble (good for special effects in games).

Please note the format of the memory locations used. I mentioned that the first TWO bytes are used to select the frequency, but did not say how one would know what values to put in one and the other. The easiest way to look at it, I would expect would be thus:

Since one memory location can only contain a number from 0 to 255, to represent larger numbers, they are stored as 0 to 255 in the first byte (known as the "low" byte), and multiples of 256 added to this in the second byte (known as the "high" byte), which means two bytes can hold a number from 0 (0 in both locations), to 65535 (255 in the first one, plus, 255 times 256 in the second one).

The ATTACK-DECAY memory, and the SUSTAIN-RELEASE memory are comprised as follows:

The ATTACK, DECAY, SUSTAIN, RELEASE parameters can all be one of 0 to 15. To form the ATTACK-DECAY value for memory location 6 in a voices 7 memory locations, simply multiply the ATTACK by 16 and add the DECAY value. This again gives a combined value from 0 (0*16 + 0) to 255 (15*16 + 15).

The control register works differently still. The value is calculated as follows:

Add 1 to begin Attack-Decay-Sustain cycle; don't add 1 to begin the Release cycle.

Add 16 to select triangular wave form, 32 for sawtooth, 64 for variable width pulse, or 128 for noise waveform.

There are other parts to add to this value, but they won't be covered here.

THE BEEP PROGRAM

Turn on your Commodore-64, and type in the following program:

```
10 SID = 54272
20 FOR I = 0 TO 28:POKE SID + I,0:NEXT
30 POKE SID + 24,15
40 POKE SID + 1,20
50 POKE SID + 5,0*16 + 0
60 POKE SID + 6,15*16 + 9
70 POKE SID + 4,1 + 16
80 POKE SID + 4,16
```

DESCRIPTION OF THE PROGRAM

Line 10 defines a variable SID, to the start of Sid's memory locations.

Line 20 should be included in all of your sound programs, and is a FOR..NEXT loop to simply set all of Sid's memory locations to 0 to ensure that no previous programs will affect our efforts.

Line 30 sets memory location 24 in Sid, to 15. Register 24 controls the overall volume of Sid (and some other things which need not be known here), and 15 is the maximum volume (from 0 to 15).

Line 40 sets the upper byte of the frequency value of voice 1 to 20, which means a setting of

0 + 20*256 = 5120.

Line 50 sets the ATTACK value of voice 1 to 0, and the DECAY value also to 0, which means when we tell Sid to do its ATTACK-DECAY-SUSTAIN cycle, it will simply go straight to the SUSTAIN volume, since we've told it not to do any ATTACK or DECAY at all.

Line 60 sets the SUSTAIN value of voice 1 to 15 (maximum volume), and the RELEASE value also to 9, which means when we tell Sid to do its RELEASE cycle, it will take about three quarters of a second to fade away to nothing.

Line 70 sets the control register of voice 1 to do the Attack-Decay-Sustain sequence, with the triangular waveform selected (#16).

Line 80 sets the control register of voice 1 to do the Release part of the envelope, again with the triangular waveform selected (#16).

Having typed in this program, type:

```
RUN
```

And the familiar:

```
READY.
```

message will come back almost immediately, with the mellow sound fading away (provided you have the volume control on your television set up reasonably high so you can hear it!)

So there are, in fact, two areas that this and subsequent articles (included here — Ed.) plan to deal with:

1) Define the capabilities of SID

2) Explain some software techniques to make SID do perform.

*****

This time, I'd like to put some of the last article's theory into practice, by giving some parameters for the SID, which will make it sound similar to musical instruments. I think this would be useful, so that you will be able to see that a music synthesizer is not limited to beeps and pops, and other sounds from television shows like "The Twilight zone"!

In the last article, the various parameters of a voice were outlined, except for the filter in the SID. Essentially, the filter will (as is implied) filter the sound output from any of the voices in a number of ways. The actual term 'filtering' means that the sound is changed by quieting the voice to varying degrees above, below or around a given 'CUTOFF' frequency. However, don't worry about understanding this concept fully yet, since this issue won't use the filtering capabilities of the SID. I simply wanted to make you aware of this feature in the SID, so you won't be taken by surprise in the future!

To begin with, let's try to emulate one of the simplest sounds: A piano. When a piano key is struck, the sound begins immediately, and then fades away in about two seconds if the key is held down. If the key is released before the sound has faded away, it will fade much more rapidly, in say, half a second.

Try this program:

```
10 SID = 54272
20 FOR I = 0 TO 24:POKE SID + I,0:NEXT
30 POKE SID + 24,15
40 POKE SID + 5,10
50 POKE SID + 6,9
60 KEY = 197
70 POKE SID + 1,16
80 GETA$:IFA$ = ""GOTO 80
90 POKE SID + 4,33
100 IF PEEK(KEY) = 64 GOTO 80
110 POKE SID + 4,32
120 GOTO 80
```

EXPLANATION OF PROGRAM

Line 10 declares the variable SID to the start location of the SID chip.

Line 20 POKE's all the SID locations with a zero to initialize the chip.

Line 30 sets SID register 24 to 15, which sets the chip to maximum volume.

Line 40 sets SID register 5 to 10, which makes the attack of voice one 0, and the decay value 10.

Line 50 sets SID register 6 to 9, which makes the sustain of voice one 0, and the release value 0.

Line 60 declares the variable KEY to the zero page memory location which holds the keyboard matrix number of the current key depressed, or 64 if no key.

Line 70 sets SID register 1 to 16, which sets the high order byte of the frequency of voice 1 (therefore, frequency of voice 1 = 16*256 [see last article for explanation of 'low' and 'high' bytes]).

Line 80 waits for a key, by GETting a keypress from the keyboard, and IF the keypress is a null, i.e., no key has been pressed, the program will GOTO the same line and keep waiting.

Line 90 sets SID register 4 with 33, which gates voice 1 on with a triangular waveform (see last article for explanation [33 = 32 + 1]).

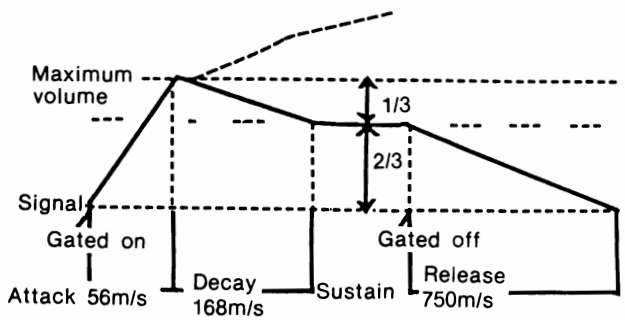
Line 100 checks to see if a key is still depressed (as with a piano), and if it is (i.e., location KEY is still something other than 64), the program will GOTO the same line and check again.

Line 110 sets SID register 4 with 32, which gates voice 1 off now that no key is depressed on the keyboard, still with a triangular waveform.

Line 120 simply goes back to line 80 to allow the program to continue indefinitely (to stop the program, the STOP key must be pressed).

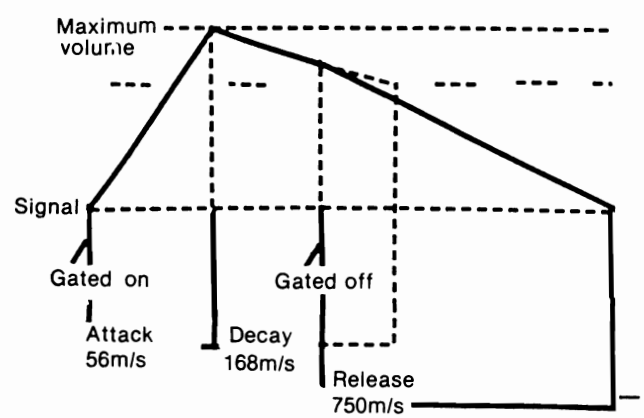
Something which ought to be understood here is the fact that, when a voice is gated off, i.e., released, the envelope RELEASEs from WHEREVER it had reached. probably, a diagram would be the best way to show this.

EXAMPLE:



ATTACK = 5, i.e., 56 milliseconds
DECAY = 5, i.e., 168 milliseconds
SUSTAIN = 10, i.e., two-thirds of maximum VOLUME
RELEASE = 8, i.e., 750 milliseconds

Now, using the same parameters, only gating the voice off, i.e., releasing, at a different point:



Here, it can be seen that the voice was gated off, i.e., released, before the envelope had decayed to its sustained level, but that, when the voice was released, it simply released from the point it had reached.

This relates to the program you just entered, because it uses this fact to simulate the 'feel' of a piano keyboard, i.e., as soon as you release the key on the keyboard, the envelope will begin its release cycle, which is set at 9, one less than the decay value (10), giving the same response as a piano key by fading away quicker once the key is released. Of course, the force with which the key is hit in the first place, which on a piano gives the initial volume, cannot be simulated here, because a key on any computer keyboard is either DOWN or UP; the speed of transition cannot be detected.

Similarly, if a voice is gated on before the release cycle has finished, the attack will begin from wherever the envelope currently is, i.e., the current output volume. To see this more clearly, enter the following to change our program:

```
40 POKE SID+5,11*16+13
50 POKE SID+6,9*16+11
```

Line 40 sets SID register 5 to 11*16+13, which makes the attack of voice one 11, and the decay value 13.

Line 50 sets SID register 6 to 9*16+11, which makes the sustain of voice one 9, and the release value 11.

When you RUN the program this time, the actual path of the envelope will be more audibly clear. When you press a key this time and hold it down, you'll hear the volume rise (attack), and then fade some (decay) to a constant level (sustain level). When you release the key, the tone will fade away to nothing (release). However, if you depress and release the key quickly, you'll hear that the tone never reaches a very high volume at all, and that is because the release occurred, i.e., by you releasing the key, before the envelope had reached either its maximum or the sustained level of volume. Similarly, if you press the key again very soon after releasing it, you'll note that the sound builds up again from the level it had faded away to, and if you keep depressing and releasing the key and ensure you're holding the key down for slightly longer than the time you're not you will "pump up" the volume.

A program that would allow us to "test" envelope and waveform combinations would certainly be useful. It would enable us to experiment with the parameters available to create a desired sound. It would be useful if we could 'play' the Commodore-64's synthesizer too, from the

keyboard.

Before we can get that far though, I'd like to explain how to derive the frequencies for musical notes on the Commodore-64.

In a musical scale, the ratio of pitch between one octave and the next is 2:1. If we had the frequencies of the 12 semi-tones of the top octave, we could generate the values of all the lower notes by continually dividing all 12 by 2 to derive the pitch of the semitones in the next octave lower. It is not necessary to go into the math here, but if the ratio between octaves is 2:1, then the ratio between semi-tones is 2 (1/12):1.

Middle A on a piano, is 440Hz. To convert harmonic frequencies to the fundamental frequencies we need to put in the SID registers, we need to multiply the former by a constant which is derived from the frequency of the internal clock in the SID chip, and the system clock.

$$\text{Frequency} = \frac{\text{Fundamental Frequency x system clock speed}}{\text{SID clock speed}}$$

Therefore:

$$\text{Fundamental Freqency} = \frac{\text{Frequency x SID clock speed}}{\text{system clock speed}}$$

Which turns out to:

Fundamental Frequency = Frequency x 16.404 (approx.)

Therefore, middle A which is 440Hz, would be:

440x16.404 = 7217 (approx.)
440x16.404 = 217 (approx.)

'A' in the next octave up would be 880Hz, or 880x16.404 = 14435 in the SID chip.

The maximum value in the SID chip is 65535 (255 in both the low and high byte), therefore, by doubling again...

1760 (1760x16.404 = 28871)

and again...

3520 (3520x16.404 = 57742)

57742 is fairly near the top end of the SID frequency value range, and doubling once more would push it beyond it, so we will base our frenquency range around 3520Hz. To create a 2 dimensional array (subscripts being 'octave', and 'semitone') of frequencies, we could use the following program:

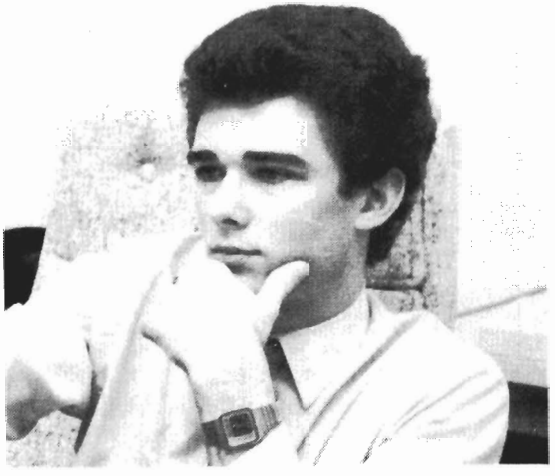
```
100 fr = 3520:rem note 'a' in top octave
110 co = 2^(1/12):rem constant multiplyer for next semitone
120 for i = 1 to 9:fr = fr/co:next:rem start fr at 'c' by going back 9 semitones
130 ss = 16777216:rem sid clock
140 cs = 1022730:rem cpu clock
150 fc = ss/cs:rem frequency multiplying constant
200 dim f(7,11):rem frequency array (octave, semitone)
300 for i = 0 to 11:rem cycle through 12 semitones
310 s = fr*fc:rem calculate sid value of semitone in top octave
400 for j = 7 to 0 step-1:f(j,i) = s:s = s/2
410 next:rem calc value for all 8 octaves
420 fr = fr*co:rem go onto next semitone
430 next:rem continue through all 12 semitones
450 rem
460 rem print out all the frequencies
500 print "frequency table"
510 print "---------------"
520 print "oct sem frequency"
600 for i = 0 to 7
610 for j = 0 to 11
620 print i;tab(4);j,int(f(i,j))
630 next j,i
```

The REMarks in the program explain how it works.

Add the following lines to hear the frequency array:

```
470 s = 54272:rem start address of sid chip
475 for i = 0 to 24:poke s + i,0:next:rem initialise sid chip
480 poke s + 24,15:rem set volume
485 poke s + 5,11:rem attack = 0:decay = 0:sustain = 0:release = 11
624 poke s + 4,32:rem gate off the voice first
625 h = int(f(i,j)/256):rem calc high byte of frequency
626 l = f(i,j)-h*256:rem calc low byte
627 poke s,l:poke s + 1,h:rem put in frequency
628 poke s + 4,33:rem now gate it on
629 for k = 1 to 100:next:rem wait a bit
```

When you RUN the program this time, as the frequencies are listed, each pitch will be sounded.



PAUL HIGGINBOTTOM

A Multiple SID Synthesizer

By Dr. Frank Covitz, Lebanon, NJ

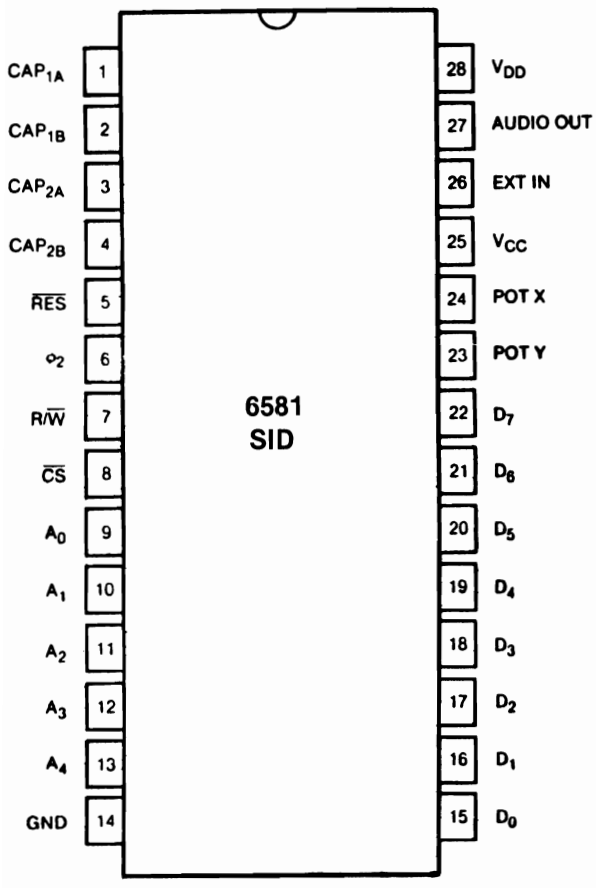
(2 articles combined)

The MOS Technology Sound Interface Device (SID), type 6581, is a single-chip, three-voice sound synthesizer, directly compatible with 650X microprocessors. Each of the three voices can have, under program control, a separate waveform selected from triangle, sawtooth, pulse (with variable duty-cycle), or noise, and each voice can have its own attack, decay, sustain, release (ADSR) amplitude envelope, in which attack rate, decay rate, sustain amplitude, and release rate are defined by 4-bit values. Frequencies may be set to a precision of 16 bits with the smallest frequency step being ca. 0.06 Hz. The chip has overall volume control (4 bits), three filter modes which are additive, (high-pass, band-pass, and low-pass), variable resonance (4 bits), and a cut-off frequency settable to an accuracy of 11 bits. Each voice (as well as an external audio input) can be routed through the filter under program control. Other special features include ring modulation, synchronization and two 8-bit analog to digital converters. The chip at the time of writing is not commercially available although it is in wide use in the Commodore CBM-64 personal computer. The 29-register set and package pinouts are shown below:

REGISTER DESCRIPTION

	A4	A3	A2	A1	A0	REG # (HEX)	DATA D7	D6	D5	D4	D3	D2	D1	D0	REG NAME	REG TYPE
															Voice 1	
0	0	0	0	0	0	00	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE-ONLY
1	0	0	0	0	1	01	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE-ONLY
2	0	0	0	1	0	02	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE-ONLY
3	0	0	0	1	1	03	–	–	–	–	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE-ONLY
4	0	0	1	0	0	04	NOISE	⊓⊓	◿◺	∧∧	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-ONLY
5	0	0	1	0	1	05	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE-ONLY
6	0	0	1	1	0	06	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE-ONLY
															Voice 2	
7	0	0	1	1	1	07	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE-ONLY
8	0	1	0	0	0	08	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE-ONLY
9	0	1	0	0	1	09	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE-ONLY
10	0	1	0	1	0	0A	–	–	–	–	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE-ONLY
11	0	1	0	1	1	0B	NOISE	⊓⊓	◿◺	∧∧	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-ONLY
12	0	1	1	0	0	0C	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE-ONLY
13	0	1	1	0	1	0D	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE-ONLY
															Voice 3	
14	0	1	1	1	0	0E	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE-ONLY
15	0	1	1	1	1	0F	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE-ONLY
16	1	0	0	0	0	10	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE-ONLY
17	1	0	0	0	1	11	–	–	–	–	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE-ONLY
18	1	0	0	1	0	12	NOISE	⊓⊓	◿◺	∧∧	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-ONLY
19	1	0	0	1	1	13	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE-ONLY
20	1	0	1	0	0	14	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE-ONLY
															Filter	
21	1	0	1	0	1	15	–	–	–	–	–	FC_2	FC_1	FC_0	FC LO	WRITE-ONLY
22	1	0	1	1	0	16	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3	FC HI	WRITE-ONLY
23	1	0	1	1	1	17	RES_3	RES_2	RES_1	RES_0	FILTEX	FILT 3	FILT 2	FILT 1	RES/FILT	WRITE-ONLY
24	1	1	0	0	0	18	3 OFF	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0	MODE/VOL	WRITE-ONLY
															Misc.	
25	1	1	0	0	1	19	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0	POT X	READ-ONLY
26	1	1	0	1	0	1A	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	POT Y	READ-ONLY
27	1	1	0	1	1	1B	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	OSC3/RANDOM	READ-ONLY
28	1	1	1	0	0	1C	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	ENV3	READ-ONLY

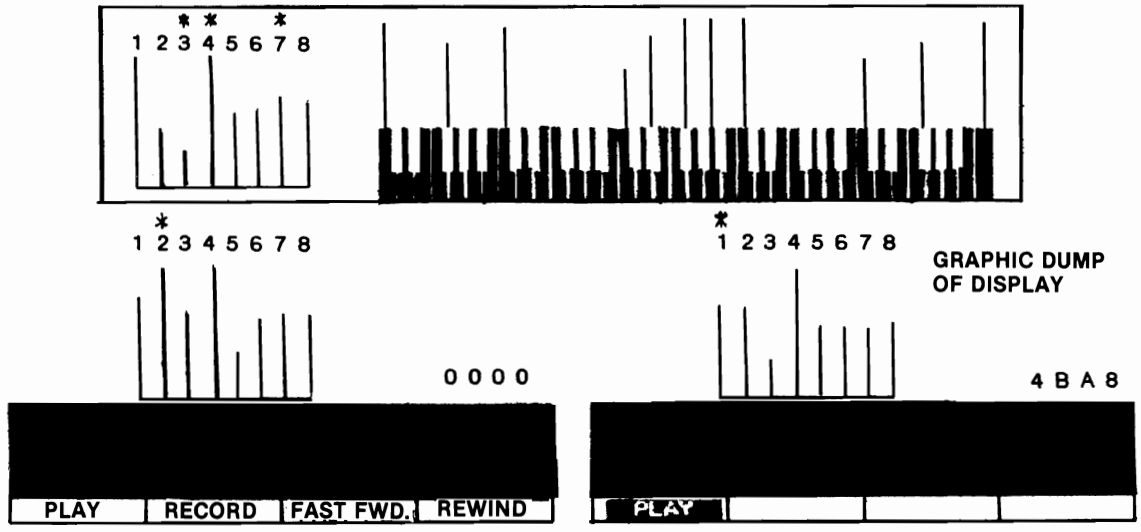
PIN CONFIGURATION



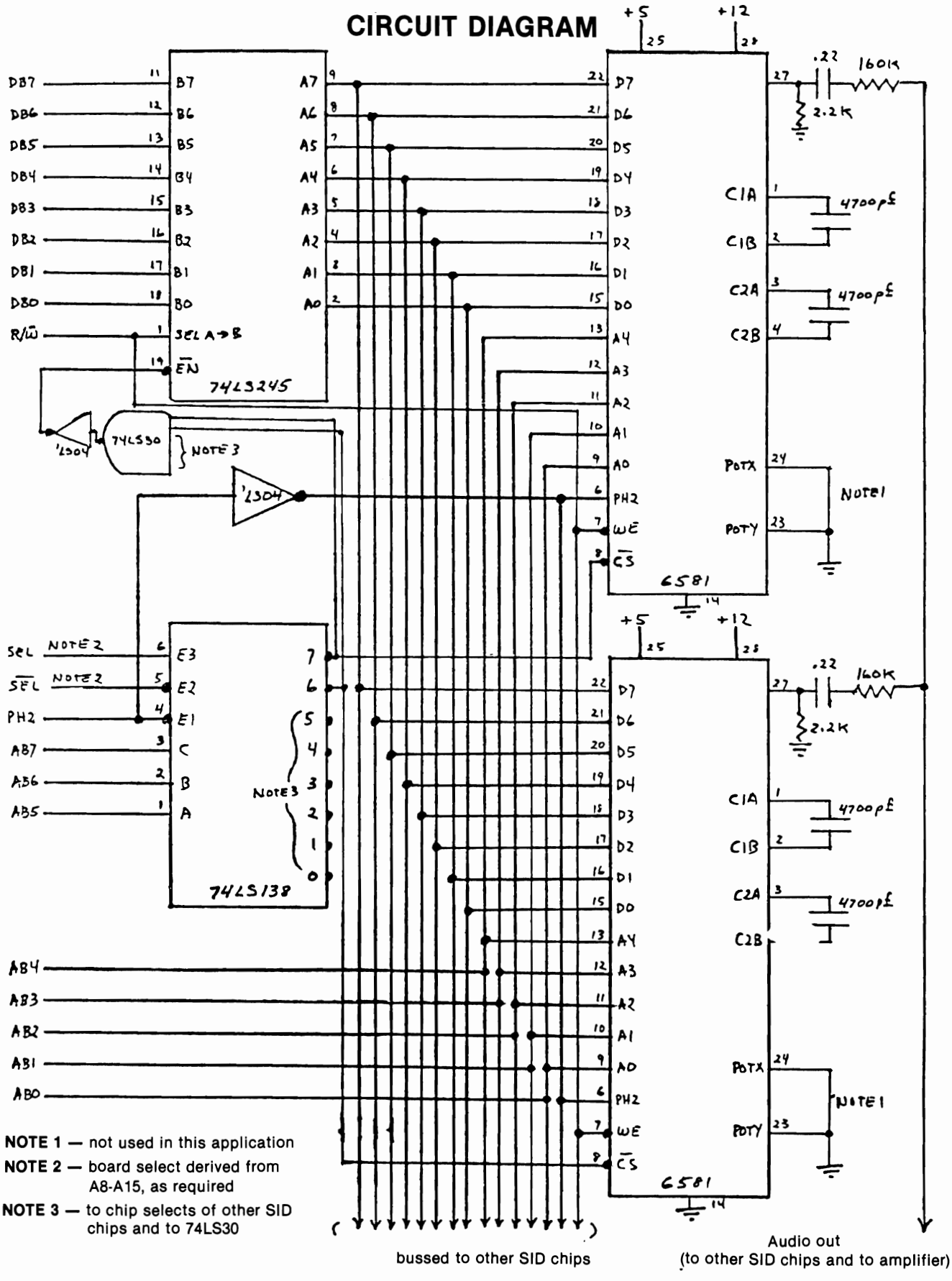
In this article, I will describe a hardware/software system which interfaces an 8 SID chip board and a microprocessor-controlled music keyboard to a MTU-130 (6502-based) computer. The software emulates a dual tape recorder system which permits an arbitrary number of sound-on-sound passes; a total of 24 voices can be active at the same time. Information can originate simultaneously from the keyboard and either or both 'tape recorders'. The digital information stream can be keyboard depression/release events, or switch and slide-pot settings available from the keyboard, which can be used to control the voice characteristics.

THE MULTIPLE SID BOARD

The 8 SID chips interface easily into a 6502-based system. Power requirements are approx. 600 ma at +5 VDC and 200 ma at +12 VDC (both regulated and well-filtered). The data lines of the SID chips connect via an 8-bit tri-state bus tranceiver (74LS245) to the system data bus. The low 5 bits of the system address bus can be directly (or through a tri-state driver) connected to the SID A0-A5 pins. The next three address bits go through a one-of-eight decoder (74LS138), the outputs of which fan out to the individual chip enable pins, and when NAND'ed through a 74LS30 (and inverted with a 74LS04) form and enable input for the data bus tranceiver. The upper 8 system address bits (inverted when necessary) need to be combined to form either an active high or an active-low board select signal. Thus the 8 SID chips occupy 1 full memory page (256 bytes). The audio output pins are connected through capacitor decouplers to provide an AUX OUT signal to an audio amplifier. A somewhat simplified circuit diagram is shown on page 204.



CIRCUIT DIAGRAM



THE MUSIC KEYBOARD

The music keyboard is based on a 61 key (5 octave) Pratt/Reed standard size organ keyboard, with double bus action, that is, each key forms an SPDT switch between upper and lower bus wires. The microprocessor scanning function is described in detail in Chapter 9 of Hal Chamberlin's book, "Musical Applications of Microprocessors", Hayden Book Co. (1980), except that a few mistakes are present in the original edition (if you undertake this project you should contact Hal or myself). Some additional features were later implemented by Hal (switches and potentiometers on keyboard), who also built the particular version of the keyboard I am currently using.

In brief, the swinger of each key is connected through demultiplexers to the microprocessor (a 6502, of course) data bus and therefore the key is accessed as user-alterable (depressed or released) read-only memory. The state of all keys can then be scanned very quickly. An interrupt timer provides time resolution to 1 millisecond.

For key depression, the time between the last break of the upper bus and the first make of the lower bus is also measured by the microprocessor and gives the effect of velocity sensing. (The reverse logic gives velocity for key release although the release velocity is almost never used in music playing.) Each key event (depression or release) generates 4 bytes of information — the 1st byte combines the identity of the key (lower 7 bits) with the type of motion (in the high order bit). The 2nd byte is proportional to the event velocity. The 3rd and 4th bytes together form a 16 bit time in milliseconds, so a time interval between successive keyboard events can be over a minute before 'wraparound'.

The switch settings (8 of them) are also scanned as are 8 potentiometers (through an 8-channel 8-bit A-to-D converter) and are saved when they change from their previous state.

All events are queued in an internal 256 byte (64 event) FIFO buffer, so essentially no events are lost due to lack of response of the host processor, which is interrupted when the FIFO has data. Transfer of data (with 'data available/data taken' protocol) to the host processor (an MTU-130 computer, in this case) is accomplished through the full handshake capability of a 6522-type I/O chip.

Finally, in case it isn't clear, let me state a fact that is probably obvious. The keyboard is purely an information generator, i.e. it is completely divorced from physical sound generation, which is perfectly appropriate since it then can be used with several types of sound synthesizers.

SOFTWARE FEATURES

The software was designed from the outset to have a straightforward "human interface" since the system would naturally be used "live" by a musician. To accomplish this, a "dual tape recorder" emulation was attempted. In other words, the player, in addition to manually playing the keyboard should also have the option of record/playback of his work. The "dual" aspect is designed to provide sound-on-sound capability not only for accompaniment but also to permit indefinite build-up of very complex music (up to 24 voices can be "live" in the present system). Since all data ends up in memory, the piece can be dumped to mass storage (disk or tape) at any time.

The MTU-130 is well suited for this task since it has 8 user-definable function keys and legend boxes (to implement a pair of PLAY, RECORD, FAST FWD, REWIND functions), potentially very large RAM area (four 64K banks in a fully populated system) to hold a significant amount of music (remember each keyboard action — key depression, key release, switch or pot setting — takes 4 bytes), and a full 6522 user I/O chip for data transfer and timing.

In the present software implementation, 5 switches are used to select waveforms from triangle, sawtooth, 3 types of rectangular, and noise; the remaining 3 select the filter mode from high-pass, band-pass, and low-pass. Four of the slide pots adjust attack rate, decay rate, release rate, and overall volume. (Sustain amplitude is set by the velocity-sensing keyboard on each note event.) The remaining 4 potentiometers were physically implemented as a pair of X,Y joysticks, one of which is interpreted by the software to allow adjustment of resonance and cut-off frequency, the other of which is unimplemented by the current version of the software. Although to some extent graphics are a "frill" in the program, the "recorder-like" functions are displayed complete with "tape" and "tape-counter". The keyboard is depicted with vertical lines depicting which keys are active as well as the velocity of the depression. Pot and switch settings are also displayed. A "graphic screen dump" of the display to an Epson MX-80 printer is on page 203.

SOFTWARE DESCRIPTION

Although a complete listing of the current software will not be given here (a crude attempt at a

software flow chart is on page 208), I will try to give enough detail to permit an experienced programmer to construct an equivalent (or better yet, improved) version. The main program loop is exceptionally simple since it just sits in a tight loop scanning the function keys looking for PLAY, RECORD, FAST FWD, or REWIND for either recorder #1 or #2. The function keys and legends were programmed to have a toggling action, so that when, for example, RECORD is depressed for the first time, the legend is reverse field highlighted to indicate that it is active, and the next depression stops the action and restores the legend to normal video (that's why a STOP key isn't needed). All musical events are initiated by interrupts which can come from three separate sources; from the keyboard at any time and from either or both recorders if they are in PLAY mode. A 6522 timer is used as a "master reference" clock from which is derived all the timing data to be saved (which also insures that all events are "meshed" in proper time sequence).

On receipt of an interrupt, the system first determines the source of the interrupt. If it is from the keyboard, the four-byte event is "handshaked" out and a data taken signal is sent. If the interrupt is from "tape", the event data is taken from memory (including the time for the next tape event). Event data is then "queued" into a 5x256 byte FIFO RAM buffer (the fifth byte keeps track of the interrupt source) and the queue input pointer is advanced. On return from interrupt, the program notices that the queue input pointer has advanced, and takes the following steps (can be re-interrupted during this phase): The data is taken from the queue and if it is a depression event, finds an available SID voice (a 24 byte "availability" table is maintained), installs the ADSR parameters (the sustain amplitude is taken from the event velocity) associated with the event source (each can have its own parameter set) into the SID voice, installs the appropriate 16 bit frequency (a 12-tone equal temperament scale was used), sets the appropriate byte to indicate "not available", and then initiates the attack.

If the event was a release event, the appropriate SID voice is put into release mode, and the voice is made available. If the system can not find an available voice, the most recent one is overwritten; this allows sustained chords to have highest priority since they most likely have been held the longest. With 24 voices, it is essentially impossible for a ten-fingered keyboard artist to get into this situation. (With sound-on-sound, there is a potential problem, but try and figure out which of the 24 sounding voices has been killed!!). In any case, the system will not "crash" even if you lay your arm down on the keyboard.

If the event was a switch or potentiometer change, just the appropriate parameter table is overwritten. If RECORD on either or both recorders was active, the event data is stashed into one or both recorder buffers (in the present system each has 40K bytes of "tape" available).

Next, a graphic depiction of the event is displayed. Note identity and velocity is displayed as a vertical line (length proportional to velocity) above the corresponding key on the pictorial representation of the keyboard. Potentiometer settings are displayed in a similar way. Switch settings just get an "*" symbol above the appropriate spot. If the event is from or is being recorded to one or both "tape recorders", the appropriate counter is advanced, and "tape" remaining is updated.

Finally, the queue output pointer is incremented, and the program jumps back to the function key scanning loop. Overall speed is such that even with furious attempts to "overpower" the system, I have not been able to make the system lose its "real time" feel. REWIND and FAST FWD functions were purposely slowed down (really, only pointers are altered) to maintain the "tape recorder" illusion. "Stop" events ($FF in the event ID field) are used to separate segments, so multiple recordings are possible on the same "tape". Continuous sound-on-sound is implemented by the following procedure:

1. Activate RECORD on #1
2. Play keyboard live (the most difficult step!!)
3. Press RECORD on #1 to stop, REWIND #1, REWIND #2
4. Activate RECORD on #2, PLAY on #1
5. Accompany yourself on the keyboard
6. Press RECORD on #2 to stop, REWIND #1, REWIND #2
7. Activate RECORD on #1, PLAY on #2
8. Go to step 2

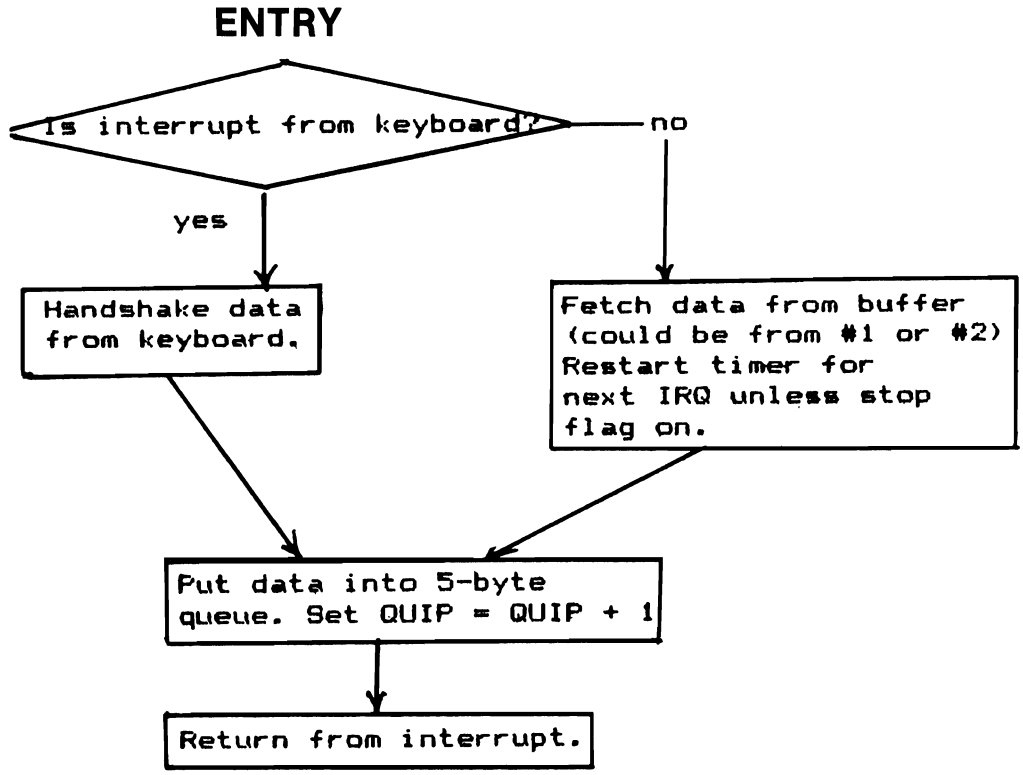
Obviously, for a software intensive system like this, an almost endless variety of improvements and modifications could be made. I will mention a few that come to mind (but I don't promise to implement them!). One very simple modification would allow a variety of temperaments (the specific set of frequencies used in the scale), to be selected, including scales with other than 12 tones. A special event I.D. for "repeat the previous section n times" would be a substantial aid in live performances. Also, splits on the keyboard (different sections having different sound parameters) should be easily implemented, as should easily created and selected "presets". Since nobody is perfect, a single stepper and editor would be essential to creating quality music output with a minimum number of "sessions", and will probably be my next software ad-

vance. Foot pedal controls to replace some of the slide pots would be nice, as would automatic vibrato, glides, and filter dynamics, Single key activation of "far out" sound effects should also be possible. How about changing philosophy and implementing an 8 voice system using a full SID chip for each voice? How about a 61 chip system? **Stop!! Interrupt!!** Obviously I'm getting carried away here.

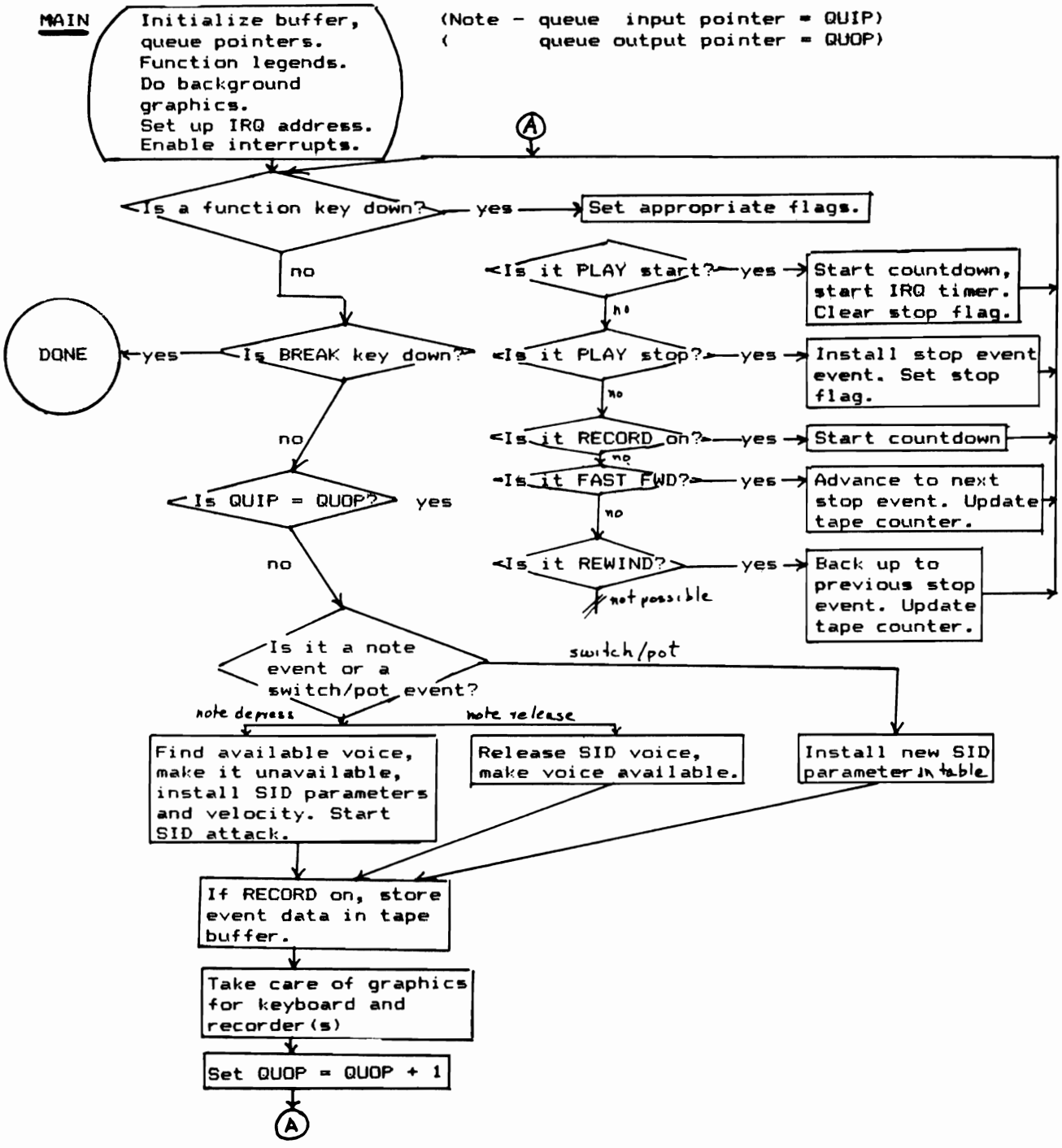
To get back to reality, it would not be fair to end without mentioning some limitations and "glitches" that the current implementation of the SID chip has. Changing the overall volume and installing the filter and filter mode causes a distinct "click" at the audio output, thus preventing these features from being used dynamically. On release, a faint but definitely audible "ghost" of the note persists (from on-chip crosstalk?) which can only be defeated by installing zeroes in the frequency register or starting a new attack. The above flaws are compounded in a multiple-chip system, and prevent the current system from having a professional quality sound. Also, since the attack phase must go to maximum volume before decaying to the sustain amplitude, the sense of "piano-forte" on velocity is not quite right. Using velocity to control attack rate (low velocity gives slow attack) gives a better effect, but you can imagine what happens if you gently depress a key and then hold it down for an appreciable length of time (it continues to attack toward maximum volume). The fixed timbres and limited dynamic timbre available prevent realistic simulation of real and other interesting sounds.

However, given the "complaints" above, I still conclude that the SID chip is a remarkable device. It has many good features including its potential low cost (should become available sometime), its ease of interfacing, and straight-forward programmability in machine language. A lot of quality hardware/software should be available in the near future.

IRQ ROUTINE



PROGRAM FLOW CHART



DR. FRANK COVITZ

Dr. Covitz was born and brought up in the Boston, Mass. area. He received his undergraduate education at the Massachusetts Institute of Technology and was awarded a PhD in organic chemistry from Harvard University. He spent several years as a research scientist at the Union Carbide Corp. research laboratory, at Bound Brook, N.J., where he was author of several technical papers and patents, and was co-author of a textbook on electrochemistry. He is currently employed at the AIRCO Central Research laboratories in Murray Hill, N.J., as a research scientist, where his current work is on the physical chemistry of gas separations and on molecular modelling for drug research.

Although Dr. Covitz' professional area of activity centers around physical chemistry research, he has, for the past several years, been an avid participant in the microprocessor revolution. He has published several magazine articles on 6502 programming and has been one of the innovators in the field of real-time microcomputer generation of music.

VICAID

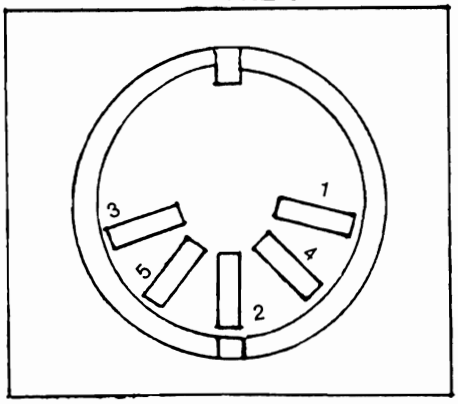
By Paul McClay, Traverse City, MI

I have a VIC-20 and I enjoy using it to play music. Though not exceptional, the VIC's sound generator is quite good and my TV speaker just didn't do it justice. Even with a respectable speaker I would be restricted. What if I wanted to record my latest works? Or experience the sound of some good speakers? I'd be out of luck. What I needed was some way to wrest the VIC's voice from the clutches of the RD modulator and make it avalable to bigger and better things. So I created the VIC Audio Interface Device, or VICAID.

The idea behind VICAID was to divert the audio signal before it got to the RF and leave it free to plug into whatever you had in mind (and in hand). No electronic genius is needed to understand and assemble VICAID. In fact, it's so simple that I refused to believe it would work until it was plugged in and singing away!

If you have the VIC manual or Programmers Reference Guide you may have noticed in the back a bunch of diagrams showing the pinouts for all of the I/o ports. You may also have noticed the one which says "Audio/Video" over it, yes that one, it looks something like this:

FIGURE 1



PIN#	TYPE	NOTE
1	+5V REG	10mA MAX
2	GND	
3	AUDIO	
4	VIDEO LOW	
5	VIDEO HIGH	

All we need be concerned with are pins two and three. Pin two is the ground. This is nothing more than a common point and, if your computer is hooked up right, will eventually wind up in the earth. It is used more or less as an electronic dump. Pin three is the audio signal. This is the

VIC's voice and it is this pin that we are after. To get a signal that you can plug into the microphone jack on most audio equipment simply connect the audio signal and the ground to opposite sides of a phono plug (or similar connector). If you were successful, the audio signal would flow in one side of the plug, through the detector in your stereo and down into the ground. Now that's not so hard, is it?

VICAID is really nothing more than a box with some wires and a switch in it. The only experience needed to assemble VICAID is a little soldering. If you don't know how to solder find a friend who does and him/her teach you.

Before you start, read everything through twice. Collect the following:

1 5 pin DIN plug
1 5 pin DIN socket
1 single pole-double throw switch
1 plug (whatever fits your equipment)
some medium fine wire
a box to put it all in
a length of two conductor cable
a length of five conductor cable

If you don't have any five conductor cable you can make some by threading five wires through a piece of tubing, likewise with the two conductor cables. A local Radio Shack should carry some experimenter boxes, you won't need a big one. Make holes for the two cables, the switch and the DIN socket. Attach the switch and the DIN socket to the box. They should come with screws. Solder each wire of the five conductor cable to a pin on the DIN plug. Feed the cable through the appropriate hole in the box and label each wire by pin number. You can do this by pulling on a wire at one end and watch what moves at the other end. Tie a knot in the cable just inside the box so it won't pull loose.

NOTE: Figure 1 depicts the face of the socket or the back of the plug. For the back of the socket or the face of the plug reverse the diagram. Be careful!

Solder wires one, four and five to the matching terminals on the socket. Double check everything as you go. Solder wire three to the center terminal on the switch (there should be three). Solder one end of a short wire to one of the outside terminals on the switch. Solder the other end to pin three on the socket. Pass the double conductor cable through its hole. Don't forget the knot. Solder one wire to the remaining terminal on the switch. Solder the other wire **and** wire two of the DIN plugs to pin two on the socket. On the other end of the double conductor, solder one wire to each of the terminals on the plug.

Check everything again. Are there any "bridges" or shorts between any of the terminals or wires? If not then close the box. Congratulations! You have just built a VICAID. Are you sure you did everything right? If you think so, then plug it into the port on the back of your VIC (or C64). If you have a VIC then plug the cable to your TV into the box. Attach the remaining cable to some sort of amplifier and you're ready to go. With the switch in one postion you will get sound as usual from your TV. In the other position your computer's voice is limited only by your imagination.

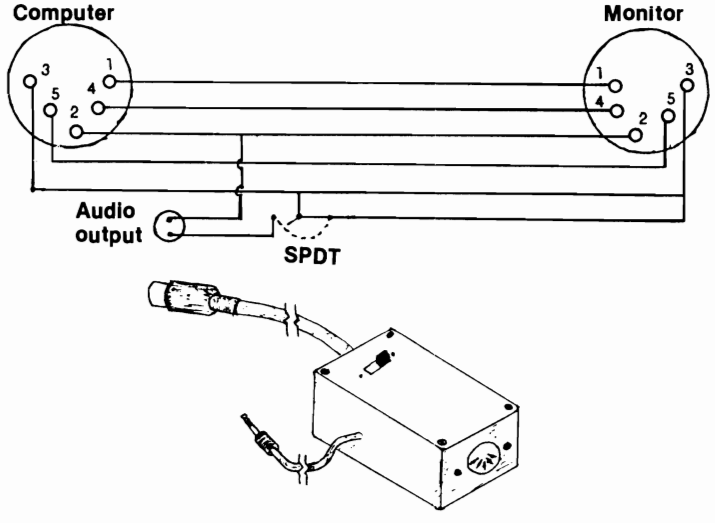


FIGURE 1.

Telecommunications

PAGE

Bulletin Boards 212
Steve Punter, Toronto, Ont.
A description of just what bulletin boards are, how you use them with your computer and what they can do for you by one of main programmers of the software that Commodore users use with bulletin boards.

Bulletin Boards — A Proposal 214
Bruce Beach, Horning's Mills
A proposal for innovative uses of bulletin boards.

Censorship of Bulletin Boards 215
Bruce Beach, Horning's Mills, Ont.
The computer is creating new social issues and one of them is examined here.

Bulletin Boards Review 217
Richard Bradley, Toronto, Ont.
A SYSOP explains what a SYSOP does and the trials in getting a bulletin board going.

Commodore Communicates 218
Robert Kobenter, Victoria, B.C.
How to use your computer to talk to other computers.

VICMODEM Switching 219
Theodore McDowell, Skokie, Ill.
A description of a simple device you can build to enhance the use of your VICmodem.

The Smart 64 Terminal 222
Robert A. Chandler, La Mesa, Cal.
A user reviews a commercially available terminal package.

AIR-1 223
Velda Hardman, Horning's Mills, Ont.
This commercially available hardware and program is not only useful for HAM operators (for which it has many functions) but it also takes MORSE code right off the air and translates it onto the screen for those of us who cannot otherwise understand it.

Bulletin Boards

By Steve Punter, Toronto, Ont.

So, you've just bought yourself a modem for your computer and are anxious to discover to what you now have access. Well, starting from the top, I'm sure you have heard about things such as InfoGlobe. InfoGlobe is a service of the Globe and Mail newspaper in Toronto, and gives you indexed access to all the stories published by the paper, going back to heaven-knows-when. WOW, you say, how do I get on? I'm not sure you'll want to. It costs $125 per connect hour!

Then we have such hobbyist-oriented services as The SOURCE. Systems like this offer a wide variety of things such as electronic mail, CHATTING with other users, access to news services, etc., and they only cost a fraction of the price of InfoGlobe. Trouble is, they **do** cost.

Finally, we come to the service most popular with microcomputer users, the BULLETIN BOARD SYSTEM, or BBS for short. Almost without exception, these systems are **free** of charge and still give you many of the advantages offered by The SOURCE. One drawback is that they only allow one user on at a time, and consequently they are very hard to get onto at certain times of the day.

So what can you get free these days, and why do the operators even bother to keep them up if they make no profit? The first part is easy to answer, the second part is very difficult. All BBS's offer a **Message System** of some sort, but each type of board differs in the way it allows you to manipulate these messages.

I am the author of what appears to be the only PET-based BBS around. There are four of my boards operating in the Toronto area, and at least 40 have been sold throughout North America. The descriptions of BBS features will be based mostly upon my own system, but I will touch upon the advantages and disadvantages of other boards.

GETTING ON A BBS

There are many BBS's available in large centres but in smaller areas there may be none. If there is none in your area you would have to call long-distance, and that would cost money which doesn't quite match up with BBS supposedly being free. Assuming that there is a BBS in your locality, (there are 14 in the Toronto area), getting on is nothing more complex than calling a local telephone number.

Once you have dialed the correct number, one of three things will happen:

1. The system will not be up yet, in which case a human being will most likely answer the phone, and it is only polite to apologize for calling outside of normal operating hours.

2. You will get a busy signal, meaning that someone else is using the board and you'll just have to call back later.

3. You will hear one, or two, rings, followed by a high-pitched tone.

This tone is actually the modem at the BBS sending you what is generally referred to as the Carrier. If you are using an accoustically-coupled modem, place the handset of your telephone into the modem. If you are using a direct connect modem, make sure you are not in a Stand-By mode of any sort. Your modem should respond by sending another tone, of different pitch, back to the BBS -- you are now ready to communicate.

Most boards require nothing more than a RETURN to get their attention; some start immediately. Some give long-winded start-up blurbs, while others do not. But one thing is sure, they all eventually ask for your name. Many systems, including mine, keep what is known as a User Log, and can identify your name if this is not your first time on. Some systems, and that includes mine, require that you give a User Code in order to sign on under your name. The User Code is either assigned by the SYSOP or, as in my case, by YOU when you first sign on. An advantage of a User Code system is that no-one else can sign on under your name and read your private messages or say things on your behalf.

SYSTEM OVERVIEW

Each system offers different things, but for the sake of space, I will confine myself to telling you what my system offers. The most popular item, and the backbone of the BBS itself, is the message system. With it, you can send messages to specific people or read messages from others. You can send PRIVATE or PUBLIC messages, as well as BROADCAST messages to ALL. A series of commands allows you to read only messages to ALL, only messages addressed to you, all messages starting from a certain point, going backwards or forwards in time, or pick off specific messages. Other commands allow you to see a

summary of message subjects, a list of messages sent by a specific user, a list of message recipients, and a list of messages sent TO or BY you.

The topics discussed in these messages vary from board to board, and the seriousness of a given board depends solely upon the SYSOP. The message system is a communications method of the future that you can join in **now**. You will find yourself gaining many penpals.

PROGRAM DISTRIBUTION

Most boards offer some kind of system for sending or receiving programs. Most of these use a simple ASCII transfer system which is compatible with all microcomputers, the drawback of which is that telephone line noise can severely damage a program during its transmission. On my system, I have implemented a special Block/Checksum which will automatically re-transmit any blocks of data that come across incorrectly. The drawback to this system is that only those persons with a terminal program capable of supporting the protocol can participate. If you are a PET owner, I make available, free of charge, a terminal program with just such a protocol.

BULLETIN SECTION

The Bulletin Section is a standard feature of most BBS's, but goes under different names. Regardless of the name, the purpose is always the same: it is a storehouse of useful, and sometimes useless, information. In some ways, it is a lot like an electronic magazine. On my board, I have Movie, TV Show, Book and Restaurant reviews; interviews with RCMP officers concerning computer crime; varied technical articles; political and social commentaries; a WordPro newsletter; system documentation; programming tips; and much, much more. As a matter of fact, I personally have 1/4 megabyte of information online.

WHICH BOARD TO CALL

When deciding which board to make a habit of calling, many things come into play. First of all, if there are only a few BBS's in your area, you'll probably end up calling them all, but, if you live near Toronto, 14 boards are a lot. In this case much will depend on your personal bias and what type of crowd is generally attracted to a specific board. In certain parts of North America, there are such strange BBS's as those dedicated to homosexuals, and another in California called Compu-Smut (need I say more?).

Whichever board you do choose to frequent, make sure you become familiar with its operating hours and, if you pass the number of that board on to anyone, be sure you also pass along the operating hours.

CB STYLE 'HANDLES'

Some boards, though not all, do allow you to sign on under an assumed name but, in all seriousness, I can't see the point. Unlike CB, most BBS people would prefer the systems to remain reasonably sane. Although I can't speak for all SYSOP's, my policy is to remove the names and messages of anyone using an obviously fake name, unless their real identity has been cleared with me first. In this way, if the user does anything seriously wrong or stupid, I will know who he is. Besides, when a joker realizes that his true identity is known, he is less likely to make a fool of himself.

SEX

Aha! That got your attention, didn't it? I'm not referring to the kind of sex you are probably thinking of; what I am going to mention is the apparent lack of females on just about **all** boards. The ratio seems to be something around one female for every 200 males, and sometimes you can go for months without ever seeing a female sign onto the board. This is most likely related to the small percentage of females who have as yet become interested in the fascinating field of computers. If you had any ideas of using the BBS to get yourself a date guys, forget it; there just isn't enough **data** available (and besides, I already tried it and it didn't work!).

IN CONCLUSION

Generally speaking, a BBS is the perfect escape for those of us with modems. It's free, it's readily available, and it gives us ways of starting friendships you would have otherwise not thought of. As of the date of this writing, a group of people involved with my BBS have gotten together four times for brunches, picnics and dinners. Once you get your modem, give a board a try — you have nothing to lose.

PUN-ishment Ylimaki.

If it's at a critical time then it's definitely a NIN-COMPUTER (nincompooter).

Will too many COLD STARTS give your PET frost-BYTE?

Bulletin Boards — A Proposal

By Bruce Beach, Horning's Mills, Ont.

As many bulletin boards as there are, it is reported that it is still very difficult to get onto them because of the amount of traffic.

The time has come for a general expansion in the number and specializations of the bulletin boards available.

This expansion could initially take place as a public service between two groups of people: 1. the knowledgeable sysops in the clubs; and 2. organizations (service, civic, social and governmental) that would be benefitted.

There comes to mind an almost limitless number of types of bulletin boards that could be establsh-ed, and of organizations that should be willing to support them. The cost to the organizations would be substantial, but not really all that much considering the benefits to be received.

A movie theatre association might be willing to support a board that listed theatre showings and that permitted users to make reviews, comments and suggestions.

Various sporting organizations might maintain a bulletin board that gave a schedule of events along with scores and team, or individual standings.

Large game clubs, such as chess, bridge or what have you, might have one enthusiast willing to maintain a bulletin board for their particular interests.

The main drawback to implementing such a proposal is the lack of overlap between computer enthusiasts who have modems and those interested in other activities. But, with the rapidly expanding number of less expensive computers such as the VIC, it will become more and more common for a family with diverse interests to find that one member of the family has a computer.

The same problem faces the implementation of widespread bulletin board systems as faced the implementation of the telephone, radio or television. In actuality, the home computer has spread much more rapidly in a shorter length of time than did any of the other mentioned technologies. With a small amount of effort, we could have a great number of very useful bulletin boards.

The key to rapid deployment is that knowledgeable persons in the computer community offer to become volunteer sysops for some of the new systems, in order to get them started. There are many government agencies that should be willing to support such an idea and be able to provide the funding for the equipment.

Two examples would be the weather bureau and the roads department (the latter for information about road conditions). If separate systems were not felt to be justified at the outset, then perhaps they could share a system. If these two organizations are not sufficiently far-sighted to see the benefits they could provide to the public, then perhaps some organization such as the association for ski resorts would sponsor it in the beginning.

Bureaucratic objections that not everyone has a computer ought to be answered that not everyone has a telephone either, nor does everyone require their particular service, but they still provide telephone answering machine service. It can also be pointed out that this is a growing trend, and that it is especially beneficial to persons who are deaf. In fact, the society for the deaf should be particularly enlisted in making appeals for services that would be beneficial to them.

These are but a few suggestions among many hundreds of others that could be made. A perusal of associations in the yellow pages will give one an idea of the immensity of the opportunity. Some ambitious young person might start up a consultancy business for this very purpose. There is almost no professional organization that could not eventually find a useful purpose for a bulletin board.

The rapid electronic interchange of information by a bulletin board system has many advantages. This is not to say that it is going to replace either telephone conversations or the printed word, because each has its own peculiar advantages, but it does certainly have its place in our society. Continuously available and continuously updated information will be a boon to almost every individual in some aspect of their lives.

Individuals in both agriculture and husbandry could have as up-to-the-minute stock market information as does the individual stockbroker. The business traveller and the recreation seeker could have extensive and detailed information at their fingertips without having to listen to extraneous information that is of no real interest to them.

What small systems, like those being described,

may grow into in the future, it is impossible to tell. Those watching the Wright brothers' plane could never have envisioned the 747. Undoubtedly, changes in technology will have to take place. Many users of a system will eventually require many incoming lines, a larger time sharing computer, and specialists in maintaining its data base.

Best of The TORPET Telecommunications

Censorship of Bulletin Boards

By Bruce Beach, Horning's Mills, Ont.

Censorship is a topic that will ring any editor's bell and this editor is no different. The two extreme positions appear to be:

PRO-CENSORSHIP

True freedom comes from private ownership and private property. The man who pays the bill has the right to call the shots. If you don't like it you are free to start your own forum.

ANTI-CENSORSHIP

Freedom of speech is one of the most precious rights of man. A bulletin board is a public forum and no one has the right to restrain another's expression.

Lest anyone should think bulletin board censorship is an insignificant issue let them remember that bulletin boards are just now emerging from their embryonic form. Procedures easily established and modified now may quickly become quite rigid through custom.

PRO-CENSORSHIP

The most avid advocate of free speech will generally draw the line at allowing someone to falsely yell, "Fire" in a crowded theatre. And institutionalized government will always look to its self preservation and close down any advocates of insurrection — "Everybody bring a gun and we will meet at Nathan Phillip's Square at seven", or "All you Boston Indians meet at the Harbor at 11 for a teaparty."

Aside from advocacy of insurrection most liberals would agree that there should be complete and free discussion of ideas. It is oftentimes not content but rather style that becomes the issue. Is removal of profanity censorship?

I am reminded of the story of two telephone linemen who were called into the supervisor's office regarding a customer's complaint that she had heard one of them using profane language beside her house. "No sir," was his reply, "what really happened was that I was holding the ladder and Bill was up at the top putting some molten lead around a connection when he dropped some down the back of my shirt, and I looked up and said, 'My goodness, Bill, you must really be more careful!"

Given the passion of circumstances or the heatedness of an argument some problems of 'style' such as profanity may creep in. It is one of the functions of an editor to edit them out while preserving the intent of the passage. Given the interactiveness of bulletin boards, they are particularly subject to these problems. A bulletin board is more nearly a 'hot medium' and it needs the coolness of an editor to temperate it.

Pornography is a similar problem of the passions. Those who condone public display of licentiousness often say they are only doing in public what others are doing in private. Yes, that is true, but that is the purpose of bedroom doors. It is also one of the purposes of editors. Pornography is expression whose only purpose is to inflame the passions rather than to enhance reason, and the editor is the door that shields the public from offence.

What is proper and what is not is a matter of social standards, but the liberal will always condone a non-impassioned discussion of any subject as long as it is presented in a style of acceptable taste.

There are also bounds of legality. The community as a whole would frown upon a bulletin board becoming a pot exchange or the basic advertising medium for the world's oldest profession. Advertising, as such, whether free or not, has left the realm of the mere discussion of ideas and, since it is again an appeal to action, is subject to community standards and lies beyond the standards of protection due to freedom of speech.

Legality also proscribes libel. Character assassination, indeed, specific negative character description of any kind, has little claim of protection in the arena of free speech. It has been said that small minds discuss people, greater minds discuss events, and the greatest minds discuss ideas.

ANTI-CENSORSHIP

Now that we have described those matters that are the proper realm of the censor let us turn to those matters which are not. There are those who are pseudo liberals who would protect us from ideas that they consider harmful. For example two unpopular ideas today with humanist liberals are racism and religion.

Personally, I love a racist bigot. Yes, he is my father. A wonderful man. Honest, hardworking, and has done as much for his son as any father could do. Highly respected too, even by some members of the black race, although his views about blacks and Jews are well known. He was a member of the KKK and you may be sure that it was difficult for him to accept that his son had become a community head of the NAACP and taught in black colleges.

I also love a religious bigot. My Jewish son who became a priest up in northern Quebec, but completely condoned the forbidding of his father's religion being taught on the Indian reserves.

The point I am trying to make is that we must separate the author from the idea. Any idea, no matter how distasteful it may be to us, should be open to complete discussion. In fact, bringing erroneous, prejudiced and plain stupid ideas out into the light of truth will do more to destroy them than letting them breed in the dark hidden recesses of the bigot's mind.

There are all sorts of bigotry besides racial and religious bigotry. National and economic prejudices are just as prevalent, to name just two others. The electronic forum is an excellent opportunity for their exposure to the light of truth.

It is an editor's task to make sure a forum is open to all viewpoints and to also make sure a rebuttal is made to those viewpoints which he thinks erroneous. Who is to have the final say? There should be no final say in the continuing dialogue on important social issues.

That is not to say that the ideals presented here are always faithfully followed. At one time it was felt that since only the wealthy owned newspapers and radio stations the poor had no opportunity to counteract the wealthy's propaganda. Thus the communists felt they were the ones to obtain true freedom of the press in Russia because the press was now owned by 'the people as a whole' rather than being the tool of a favored few.

Knowing the lack of freedom of the press in Russia has reinforced the view in the West that freedom of the press comes from private ownership and private property. But does the one who pays the bill have the right to censor the contents?

The answer is, "No!". The operative word is 'right'. There is often a confusion between legal right and ethical right. In our society an owner of a bulletin board may well have a legal right to censor that board in any way that he wishes, up to and including disbanding it altogether. However, it does not mean that he has the ethical right to prevent the free expression of ideas that he does not like.

This is because the common custom of a free society has determined what are the rules of fair play. We would all resent extensive violations of a 'first come, first served policy' by a merchant, but there is no law stating that he must follow that practice. Many similar examples could be given. In any sport or game we expect the provider of the equipment to play by the rules just like anyone else. There is nothing to prevent them from taking their ball and going home if we insist that they play by the rules, but we will think them a terribly poor sport if they do so.

The same applies to the operator of a bulletin board. If he is a racial bigot, or a religious bigot, or a humanist liberal bigot, he may not like certain ideas being expressed. But, while he may have a legal right to censor, or even close the bulletin board, he does not have an ethical right to do so simply because he does not like the ideas being expressed.

The electronic bulletin boards present a marvellous opportunity for the rapid exchange of controversial ideas in a free forum. I doubt that you would find many of them in Russia. Neither will we continue to have them here unless we all make a continued effort to make sure the rules of fair play are understood and followed.

Glasses filled with HOMEBREW cause the hacker and program to get LOADED at the same time.

Ylimaki

Bulletin Boards Review

By Richard Bradley, Toronto, Ont.

SYSOP

I have looked upon the BBS scene from the only two ways possible, the first being from the user's point of view, and the second being the SYSOP's point of view. I began as a new user of the various BBSs throughout Toronto and surrounding areas.

I first got interested in BBS last winter, when one was announced at a local meeting. I was completely baffled as to the use of the BBS, and what benefit I could possibly get out of it. So, I began asking friends about it.

Around this time, a good friend from school, Mike Mckechnie, was in the process of purchasing a Modem-80. Mike had run into a bad time of things (nothing to do with the BBS or the Modem-80) and had somehow managed to blow up his computer. (I, too, have experience in this field — you should see what I can do with a PET). Anyway, I managed to get hold of this Modem-80 from Mike, and he and I got on the BBSs. We tried three before we got on one that wasn't busy, but sounded somewhat friendly.

After many attempts, Mike and I finally managed to get both of us signed on there and, after reading the various messages, we tired of it and tried another one. This time, our luck was good and we got through first time. After typing in my name and being notified that this was my first time on the system, I then completed the sign-on by typing my city, and I created a user code for myself. As far as the user code goes, you make up your own code for your name, and then you will have to remember that code to sign on again.

On this board, we noted that there were many messages, a series of bulletins, and some programs that were available for DOWNLOADing. But, due to the fact that we were using a Modem-80, we were not able to DOWNLOAD anything, because at that time there was no software that would allow us to. (That software situation still prohibits Mike from DOWNLOADing.)

After we had read all the messages and bulletins, we then went on a rampage, and did the same on four other BBSs. From my first impression, I found the PET-based BBSs much easier to get used to than any of the other systems, and I still feel that way now. However, to continue the story, Mike got his PET back, and I got myself a modem that allowed me to DOWNLOAD.

BECOMING A SYSOP

For about two or three months, I continued my nightly rounds of the BBSs, and I suppose became known as a regular user of the systems. Somewhere around the end of May, I remembered someone asking for aid, so I asked if he could use a hand with his board. He told me that, if I was willing, I could help him by taking over the Bulletin section and keeping it up-to-date. I agreed, and in the first weekend spent many hours getting the Bulletins the way I wanted them. I continued this through most of June.

While this was going on, I had made arrangements to get a student BBS going. Our computer club at school had some money left at the end of the year and most of us thought that a BBS would be a good idea. We were told that, if we purchased the BBS program, we would be given a modem and answer circuit with which to run it. After a quick bit of running about, we ordered the BBS and picked up the modem. About June 15, 1982, the NORTECBBS went up for testing. After a rough start, we went on-line from my house for the summer and we ran for about three weeks without much difficulty; then we went down for a while. Since we got the modem back, we have been running every night from 1930 hours to 0900 hours, and 24 hours on weekends.

When we first went up, we received many complaints that our carrier was too low for people's modems. After about a month of this, I decided that I was going to create a BBS with a carrier that would melt the cups of anyone's modem. What I did was simple. I am very surprised I did not think of it sooner. I merely took a stereo power booster like the ones that you hear in the Camaros and Firebirds that are always shattering store windows and drivers' ears. Anyway, I did a little unsophisticated wiring between the modem, added the booster, and the net result was a 60-watt carrier. Well, since then, we have not had any more complaints about the carrier (and I hope you will receive none from Ma B. — ed.), and in fact we have even received a few compliments from our users.

At present, we are still operating from my house but will soon be setting up shop from the school.

I cannot myself decide which is more of a challenge and pleasure — to be a user or to be a SYSOP. Why not try the BBSs out and experience the user side of it?

Commodore Communicates

By Robert Kobenter, Victoria, B.C.

All Commodore owners have two worlds available to them. One is contained within their computer room and takes the form of their particular Commodore used as 'their' personal computer. That era is happening in ever-increasing numbers each and every day. It is relatively new (since 1970), and is still in its infancy. Any Commodore owner has an interface to the outside world. Whether you use the GPIB of the 2001 or a VIC/64 serial interface (for us SPETters there is the RS232), you can phone into a school's host computer or into one of the many Bulletin Boards set up for us to use. A whole new accessible area exists outside of your computer and, with a few relatively inexpensive add-ons, you can enter the SOURCE® or Compuserve® , and yet another world of computing becomes available to you (for a fee of course).

When you use your VICModem at 300 baud, you are configuring your machine to a host computer in which your Commodore is a relatively 'dumb' terminal. With proper software, you can use your immensely powerful cursor keys. I have used both the C64 and the SuperPET in a "dumb" terminal mode with a VAX 11/780 minicomputer at the college I attend. The '64 term' software is a very adequate package that introduces you to advanced computings or to information exchange via the telephone lines. C64 owners are lucky in that an excellent product (though skimpy literature) is available for telecom, that is, the C64 link.

SuperPET owners can obtain (for $15.00 U.S.) John Toebe's Newterm program from SPUG. This is a very advanced package that I use to upload/download programs from my disks to a VAX. A brief description takes a few pages so I recommend the articles presented in the last three issues of the SPET Gazette. An excellent 6502 SPET program for TC appeared in the April 1983 issue of the Micro magazine. For German members of a club, a very interesting V.24 package appeared in 'mc magazin' in the September 1983 issue for use with the CP/M BBS used by 'mc' in Munich. Having tried it via very long distance, I recommend it! Europeans, due to the non-progressive ways of their PTTs (post offices) have yet to fully recognize the power of telecomputing. The 'mc' BBS is one of the few that I am familiar with. If you know of any others, please write me with a description of formats and type (including telephone number).

Other BBSs are relatively easy to access. An excellent (though incomplete) article appeared in the September issue of COMPUTING NOW! on page 6. A good introductory article appeared in the May 1983 issue of Computing Now! Consult with your local dealer for any information which he may have on a local network. Also, check your club library for some good public domain software on telecommunications.

MR DRACULA **by Quillan**

VICModem Switching

By Theodore McDowell, Skokie, ILL

This switching device will simplify use of the VICmodem 1600 with VIC 20 and C64 computers.

Normal VICmodem use necessitates manually transferring the handset cord connector to the modem. It is at best an awkward manoeuvre. If workspace is crowded, the repeated unplugging and plugging in becomes a downright nuisance.

The switching system described below is applicable to telephones in which the line input connector to the phone base is the same size and configuration as the handset connectors. Parts are generally available from well-known electronics suppliers. Some of the connectors and cords may be available from local phone centre stores.

For reference purposes, parts used in a working setup will be described by brand name and part numbers:

— wire junction (box) with modular plug...Audiotex, GC Electronics, Rockford, IL, PN 30-9800...$7.00 U.S.

— modular extension cord (6-ft.)...Audiotex PN 309540...$2.80

— BAT handle toggle, miniature 4-pole, double-throw switch, rating 5A-125VAC/28VDC ... Audiotex PN 35-038...$6.40

— modular inline coupler...Gemini Ind. Inc., 215 Entin Rd., Clifton, N.H., Gemini PN TA68...$2.69

The switch listed is a three-position On-Off-On. A two-position on-off switch would probably work as well, provided there is no overlapping internal contact during switching. Generic purchased or working junk parts substitution should pose no particular problems since the essential core of the mechanism depends primarily on the action of the 4PDT switch and not on the specific configuration of the cabinetry and connectors. Correct wiring, however, is important.

TOOLS NEEDED FOR ASSEMBLY

— hand drill or drill press, for use in drilling switch hole

— pliers or wrench, to tighten switch mounting nuts

— small saw, jeweller's saw, coping saw or hack saw to cut connector and cord ports

— small flat file, to smooth cut edges

— light-duty soldering pencil or gun, to complete internal wiring connections

— 60-40 resin core, 1/16 diameter wire solder

— knife or small-gauge wire stripper, to prepare wire ends for tinning

— screwdriver, size and head configuration to match screw heads inside the junction box

— plastic model cement, to anchor inline coupler inside box

The wire junction box supplied by GC Electronics is an easily-machined plastic box with a bank of double terminals color-coded to match modular telephone connector wires. The snap-on cover can be pried loose with a small screwdriver. Cement the inline coupler, small slot (for spring tail on plug) up, in the open corner of the box next to the black terminal. See Figure 1. Check for switch body clearance in the remaining space (near red and green terminals), then locate and drill a switch body clearance in the remaining space (near red and green terminals); then locate and drill a switch mounting hole in the cover.

Saw a coupler access port in the cover near the appropriate corner. Smooth rough edges with a file. Cut a slot for the extension cord-to-modem in the box at the knockout port near the green terminal. Cut the modular extension cord about 1½ inches from one end. The short section will be used for internal connections; the long section will plus into the modem. See Figure 2 for a block set-up and wiring.

Carefully strip exposed wire ends to prepare for tinning. Another small section from the longer length of cord can be cut for terminal strip-to-switch connections. Order of wire connections to switch is not important so long as connections among the three cords are color-matched (black to black, yellow to yellow, etc.).

Some cords are not color-coded. If you keep the wires properly oriented left to right with respect to the connectors you have a fair chance of getting them correctly connected the first time. With a

VOM meter, you could trace and label the wires and save yourself the possible aggravation of having to rewire the switch.

The telephone is used in the normal manner with the switch on "phone side" (your label) for conversation and on "modem side" to receive incoming calls or to "dial" out.

Some (U.S.-made) push-button telephones will not work with the VICModem. In particular, Western Electric single-party phones identified CS2500DMG, manufactured since the beginning of 1983, will not function. The difficulties have been traced to a change in the push-button mechanism from a full-stroke, buttoning contact on older phones to a click-stop, half-stroke push button on the new phones.

If you must purchase a new push-button phone to get your system up and running, check whether your incoming phone line is compatible with touch-tone systems, and whether the button system will work as intended. It may be necessary to purchase an older phone.

If you have a phone with two different-sized connectors — line-to-base is different from base-to-handset — and you just happen to have another phone available, the above-described system can be modified to switch between two phones rather than between handset and modem. See Figure 3. Procedure is the same.

Phone 1 is used to place and receive calls in the normal manner. To access another computer, dial from phone 1 and listen for the connect tone. Flip the switch to the modem (phone 2) side when the audio signal sounds. Receiver on phone 1 may now be replaced in its cradle without disconnecting the set-up, since the signals are routed into phone 2. Phone 2 may be left permanently connected to the modem. Handset for phone 2 is not used.

Don't forget to switch back to phone 1 after sign-off.

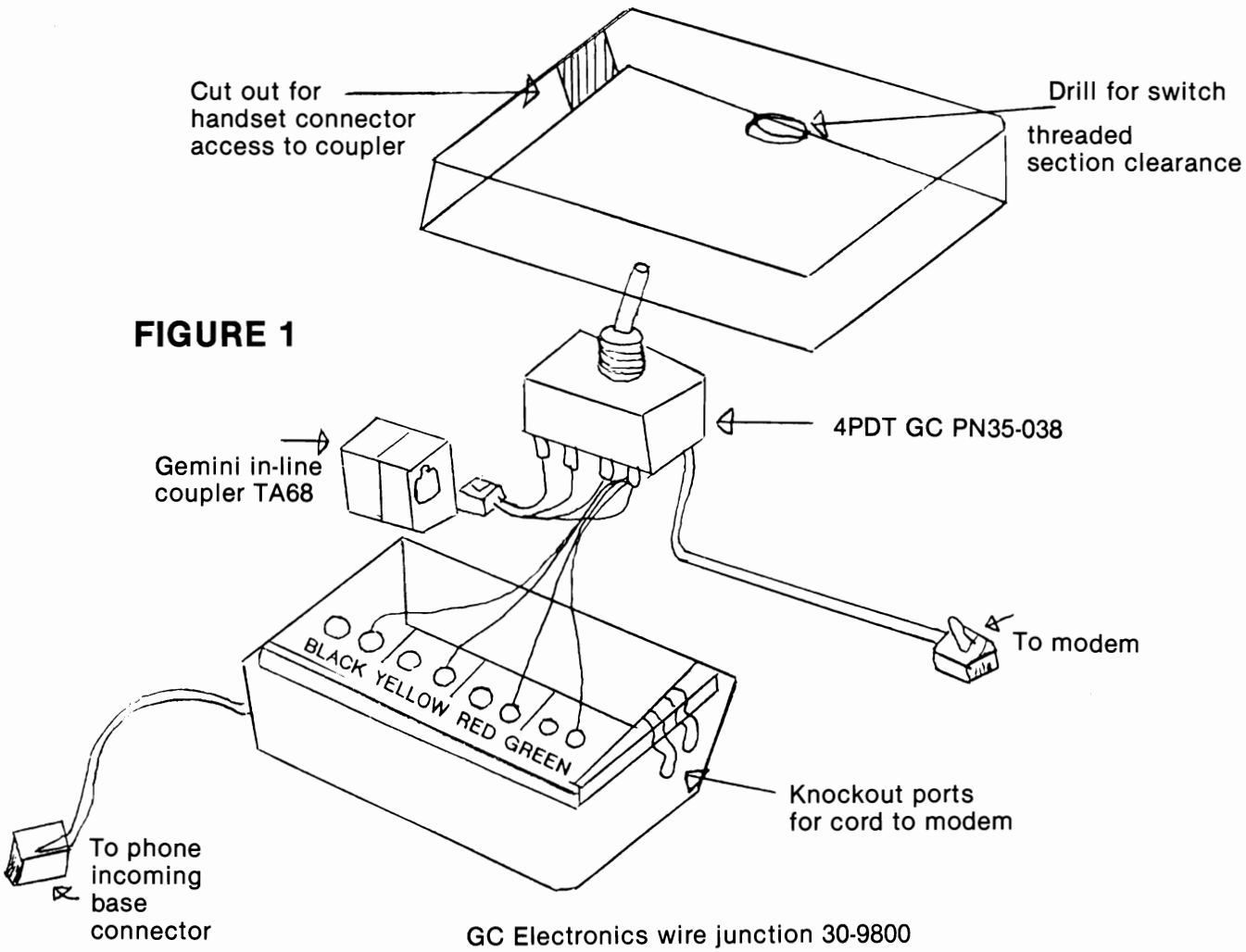


FIGURE 1

GC Electronics wire junction 30-9800

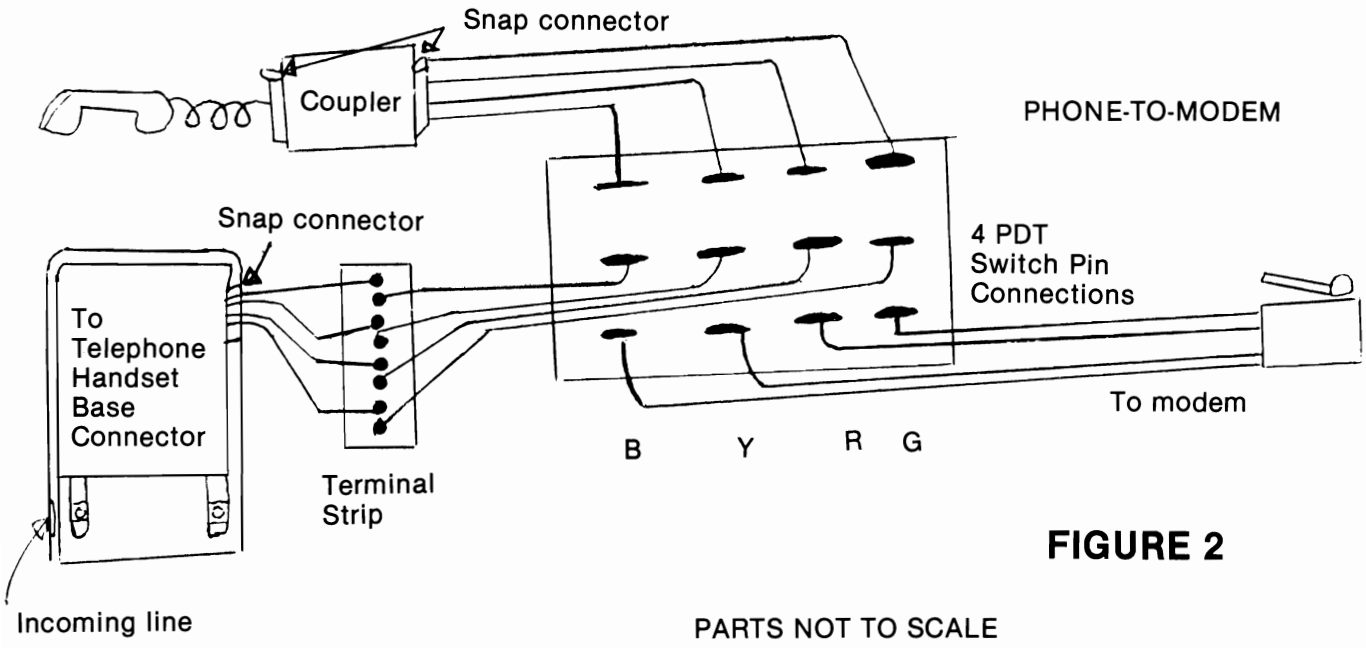


FIGURE 2

PARTS NOT TO SCALE

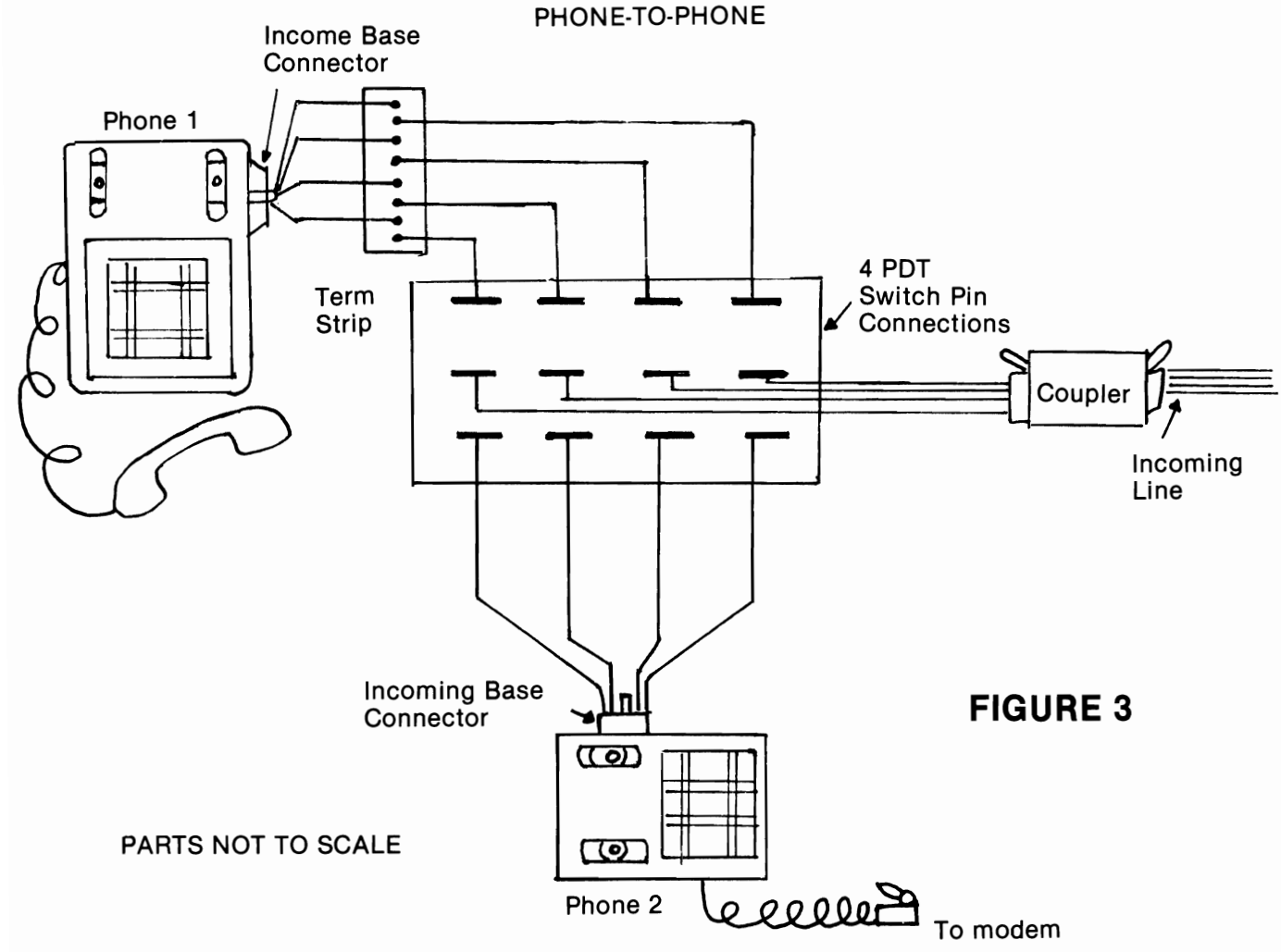


FIGURE 3

PARTS NOT TO SCALE

The Smart 64 Terminal

By Robert A. Chandler, La Mesa, CAL

Are you tired of not being able to upload and download from your terminal program? Have you had bad experiences with so-called fast running programs that actually give you enough time to get a cup of coffee in the time it takes to print a screen? After scrounging up the money to buy your hardware, do you find yourself leery of spending big bucks trying to find a terminal program that will do what it is supposed to do, and will run on your C-64? Then friends what you need is the Smart 64 Terminal program.

That may sound like a pitch from an old medicine show, but it pretty well describes how I felt before I found The Smart 64 Terminal.

Now for the technical stuff. The Smart 64 Terminal is a menu-driven program that I found to be extremely user friendly, a grave necessity for a person with my limited knowledge. The program is available on either disk or tape, and is accompanied by an 8½x11-inch, twenty-four page manual. The manual is relatively complete and instructs the user in the building of a custom system disk. Though I feel the program was designed with Compuserve-type systems in mind — the building of a custom system disk allows the user to tailor the disk for use with whatever system he wishes to log on.

When you first build and run your system disk, you will be asked to set the colours that you want to see (border, screen and character). Once set, these will be permanent, unless you choose to change them via the menu. Next you will be asked to define each of four function keys that you can set up to print repetitive commands. I have one disk set up with all of the passwords I use on the local systems in my area. Once you have done this you will be asked to set your I.D. and password function keys. After setting these the I.D. will print on the screen but the password will not be seen, an added security measure for those times when other eyes are watching your screen. There is also a printer option that you will be asked to define. This is to allow the program to be used with a 1515, or 1525 printer with upgraded ROMs.

Once all of that is done you are ready to start. After loading the program via the boot, the screen will show the various loading functions taking place, and when finally loaded you are presented with the function menu. The menu gives the user fifteen options to select from. They are as follows:

1. **Online**: pretty self explanatory.

2. **New File**: this allows the user to re-open the download file.

3. **Close File**: allows closing of the download file, and empties the buffer to allow for extracting, changing disks etc.

4. **Print File**: gives you a hard copy of what you have downloaded.

5. **Extract**: this gives you the ability to create individual files from the downloaded text.

6. **Text to BASIC:** lets you create a BASIC program from a downloaded sequential file so you don't have to type it out.

7. **BASIC to Text:** the opposite of the above. Will allow you to transmit the file in PETSCII if you name the file with the first letter being an "x".

8. **Editor Link**: will load an editor or word processor to allow you to create upload files.

9. **User I.D./Password**: lets you change your password and I.D. number.

10. **Function Keys**: lets you change the user defined function keys.

11. **Colours:** gives you the ability to change the colours you have set.

12. **Printer**: lets you change your printer set-up.

13. **Modem:** This is pre-set to the standard defaults, but selection of this function will allow you to change the defaults to whatever you need.

14. **Disk Commands**: selection of this gives you the ability to manipulate your disk with the DOS 5.1 commands (C-64 wedge).

15. **End**: this is an exit from the program. It is necessary to use this to make sure any open files are closed properly.

Well there it is. Seems like a lot doesn't it? Hold on though because there is more.

Included in the users manual is a program that will allow the user to define and customize the transmit and receive tables used by the program.

This means you can define your keyboard to transmit whatever you want it to, within ASCII limitations of course, and that you can set up the program to read incoming data that may be exclusive to a particular system you use. Standard control key functions are pre-implemented but this also allows you to change these if you wish.

The program is compiled via Petspeed, and in my opinion runs faster than some of the machine language programs I have tried. The download buffer is 28k big and gives you the choice of either allowing auto-dump to the disk, or selective clearing if you don't want to keep what you have downloaded. The buffer is also dynamic, allowing you to turn it on and off as you desire. On is signified by a little box with a down arrow in the upper right corner of the screen. Another little nicety (if you are a tightwad like me) is a timer you can set. This counts down your online time, and upon expiration signals you with an audible tone and a flashing box in the center of the screen that says "Time To Quit".

In all honesty, however, I do have to admit that I find having to create my upload files with an outside word processor to be somewhat of an inconvenience. Also I found that not all word processors will create the right type of files compatible with the upload feature. I use Wordpro 3 + /64 and find that it works perfectly. This is a small inconvenience and I feel it is offset by the many other features the program offers.

CUSTOMER SUPPORT

I know this may be a term that you Commodore users have lost touch with. I can't start closing this review without throwing in a word along this line, though. My experience in computing is still at the novice stage and, being so, I find that, at times, the simplest solution to a problem can be completely out of sight. I found the creator of Smart 64 Terminal receptive to all of my questions, good or bad, and willing to give me unlimited assistance with whatever my problem was. In my experience, this type of customer support is very hard to find these days.

At this point I was going to include a few lines about the updated version, planned for release around the first of July. However, after just getting off the phone with creator Joe O'Hara, I think the updated version will deserve a review of its own. So, for now I'll just tell you that it will have all of power I've just told you about and many more new and exciting tools as well.

On a scale of one to ten, I give The Smart 64 Terminal eight stars.

THE SMART 64 TERMINAL
From:
Microtechnic Solutions Inc.,
P.O. Box 2940,
New Haven, Conn. 06515.

AIR-1

By Velda Hardman, Horning's Mills, Ont. Ham Radio #VE3L1B

The Air-1 microlog is a radio interface cartridge for the Commodore, VIC 20 or 64. It is a complete terminal for sending and receiving RTTY and Morse code. The only parts needed are the VIC 20 or 64 and monitor or TV and the Air-1 and, of course, my Ham radio. If I want to save the memory or other data, I need tape, disk drive or printer.

The video display has a split screen so that the text received is displayed on the lower half, and what is in the buffer to be transmitted is displayed on the top half. The split screen option can be turned on or off. I can remove the split screen and use the entire video display to receive text, and I can still enter text into the buffer but I'll be unable to see it until it is transmitted.

The top line of the screen indicates the mode by displaying a (T) for the transmit mode, or (R) for the receive mode. It also tells me if I am operating in RTTY (Baudot) or ASCII or Morse code, and displays the words per minute (it allowed + /- 50 errors in actual speed). In Morse, I can easily change the speed at any time and, if I desire, lock it in so it will stay at that speed. Otherwise, it changes automatically, which are speeds from 5 to 149 wpm.

RTTY (Baudot) speeds are 60, 66, 75, 100, 132 wpm, and ASCII code speeds are 110 and 300 baud. These speeds can be changed as desired.

("ASCII" stands for American Standard Code for Information Interchange, whereas "Baudot" pro-

nounced "Bau Dough" is a man's name.)

There is also a clock with four-digit display of hours and minutes and three characters in the top right-hand corner (time zone) that can be set as desired. I can transmit the time and zone on command at any time I am in the transmit mode and it will be displayed on the lower half of the screen as it is being transmitted. When the unit is first turned on, the clock is reset to 00:00.

On the top line in the right-hand corner of the screen, there is a red dot tuning indicator that flashes on/off in synchronism with the incoming tone when properly tuned. As a further aid in tuning, a tone is available on the monitor audio channel through the speaker, just as various sound effects are heard when using game programs. The tone should sound clean and should be synchronized with the tuning indicator. It is designed to hear an 800HZ note.

The Air-1 has a number of programmable memories — eight message memories of 128 characters each; two ID memories of 64 characters each; a WRU (who are you) memory of 11 characters; and two selective print memories of 11 characters each. The Commodore key sends CQ CQ CQ DE in all codes. I can store my call sign in the ID memory and my contact's call sign into the second ID memory. I find this set-up very convenient.

Up to 11 characters may be stored in the WRU (who are you) memory. Whenever the Air-1 hears or receives the memory stored in the WRU keyword exactly, it automatically goes into the transmit mode and, after a short delay, sends whatever is stored in the ID memory (my call sign), along with the contents of the text buffer, and then returns to the receive mode. When the system hears the specific string stored into the 9 memory, the printer port will be enabled and, when it hears the string corresponding to the one stored in the 0 memory, the printer port will be disabled.

The Air-1 has a number of transmission modes. In the character mode, each character is transmitted as it is typed. In the word mode, an entire string of characters is transmitted as soon as the space bar key is pressed. In the line mode, the characters are transmitted as soon as the return key is pressed (in this mode, I can edit my text before transmitting). When a word is typed, it will be displayed above the split line and, as I transmit, the word is displayed below the split line.

A segment of my text can be repeated when transmitting as many times as I wish, and I can set it to stop at any time. I can also have it repeated continuously until I stop it. This is handy for sending CQ and my call sign.

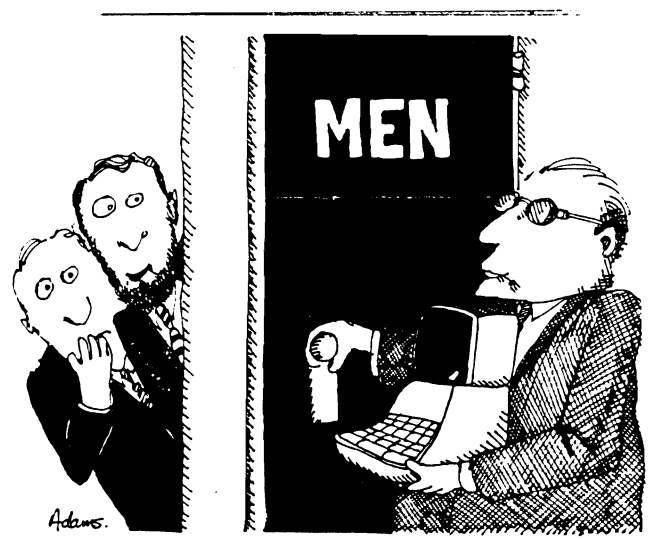
The hand key can be connected to the jack in the rear of the Air-1. With the set in Morse code and with the rig in receive mode, sending code by hand can be observed in the video display and the speed that I am sending will show on the top line. I can also listen to my code by turning up the volume of the audio on the monitor. Experimenting with the hand-sent code assists me in becoming more familiar with the Morse decoding algorithm.

To Save on tape or disk I have to go to VIC BASIC, then save it by using my own File Save program in BASIC, then return to Air-1 without erasing the user memory's type. When I first start up, I go to BASIC, load the memory area, then return to Air-1.

I can also recall and transmit two permanently-programmed messages, 'QUICK BROWN FOX' and 'RYRYRYRY....' messages.

I really enjoy using the Air-1. It helps to make me more confident, and I hope to spend many hours meeting new contacts on the air, making it more exciting.

Air-1 is available from:
Microlog Corp.,
18713 Mooney Dr.,
Gaithersburg, Maryland 20879.



"BOY, HE USES THAT COMPUTER FOR EVERYTHING!"

Education

PAGE

Computers in Education 226
Colin A. Haig, Mississauga, Ont.
The voice of experience speaks

Audio Teach with Commodore Computers 228
Ron Byers, Truro, N.S.
Here is a neat little device you can build that will bring audio interaction to your computer. Great for teaching languages and such.

TORPET Education Disk 236
G. R. Walter, Proton Station, Ont.
A description of the Best of The TORPET Disk which contains Prof. Ponzo's tutorial programs that teach BASIC and machine language programming.

Commodore Introduces Home Education Programs for the VIC 20 238
A description of a commercially-available sub-set of the Education Disks catalogued in the next article.

Education Disks Catalogs 240
This is a complete list of all the Commodore public domain education disks that have been released at the time of this printing. Where to get them is explained at the end of the previous article.

Computers in Education

By Colin A. Haig, Mississauga, Ont.

What is REALLY going on in Computer Education out there?

The purpose of this article is to clear up a few points and to present a different perspective on the school situation from that presented by David Bradley in the September issue of The TORPET.

I am quite heavily involved with Computer Education on many levels, both in the Peel Board as a student and part-time employee, and at the Ontario Institute for Studies in Education, where I have been involved in a project directly relating to the Canadian Educational Microcomputer (unofficially known as "The Bionic Beaver").

I've been a member of a club for quite some time now - (I joined as #1320 but I believe we have almost 10,000 members now). I've seen a lot of changes in the computer world and I feel it's about time I made some comment.

When I started with computers, just over five years ago, video games were basically unheard of, except for that most fascinating game — YAWN — known as PONG. Of course, at this time, there weren't any ***VIDEO GAME FANATICS*** as they are currently known. BUT there WERE a few hackers (or is "Techno-Computer Enthusiast" more appropriate?). The most interesting fact was that most of this bunch didn't spend too much time playing games. Instead, they involved themselves with actually learning to use the computer, by writing useful programs, instead of wasting their valuable computer time on keyboard-smashing games.

Presently, we see all kinds of people who go to the local computer shop to buy a VIC or a C-64, with the main purpose being to play games. Only after the games have lost their novelty do most of these people realize that they can actually PROGRAM the computer! In a lot of cases, however, the machine just sits, unused. (I hope that I have not offended any VIC or C-64 owners out there, but you have to admit that it does happen.)

TYPES OF USERS

As a result of this great increase in the number of students who have or use computers, we find that there are several different categories of users. There are **USERS**, the sort of person who simply uses the computer to get a job done; the **GAMERS**, the kind who spend much of their time twiddling joysticks and bashing buttons; the **HACKERS** or **WHIZ KIDS**, who do all kinds of stuff with their machines, ranging from practical to plain dumb; and then the **BRAT KIDS**, (referred to as B/K's) who own all kinds of goodies, and generally don't make much good use of their stuff.

I personally prefer to be known as a HACKER, but I have been called a B/K from time to time.

These same groups of students can usually be found in ANY computer classroom.

The USERS generally get their assignments done, and they tend to consider the computer to be another piece of junk for them to either learn to use or learn to hate.

The GAMERS are usually the ones who sit at the far back corner of the class playing Space Invaders, while the teacher drones on about the advantages of the FOREIGN X loop (or was it the FOR NEXT loop?). You can usually spot the Gamers, because they are the ones leaning all over the monitor screaming about the new high score.

The HACKERS are normally busy planning new ways of disabling their neighbour's stop key by remote control, or they are working on a new way to sort matrices backwards. A casual observer would note that the hackers are unable to type in a standard fashion, but that they know where all the keys are on 13 different keyboards. Those who try to learn how to type usually give up out of frustration as soon as Commodore puts out a different model with a different keyboard. (You will have noticed this if you have ever tried pushing the stop key on a 4032 and then tried to find it on an 8032! Usually you get an '@' symbol.)

The main advantage of this approach is that learning can actually be FUN!!!

The B/K's are usually the ones who think they know it all until it comes time to do any real work, when, of course, they turn to the Hackers and Users for help. These folks generally have a Superiority Complex until they find that their latest computer has the power of a wrist-watch and the speed of a 110 Baud TeleType. The B/K's are also the ones who pride themselves on the fact that they can actually type but, as soon as the keyboard changes, they are doomed.

One of the problems with a lot of the B/K's and some of the Hackers is that they hold the belief that they know everything that is to be learned from a resource. I'm sure that a large number of people would agree that there is always SOMETHING to be learned, whether it is a new command, a new technique or just some general pointers on programming style.

APPROACHES TO TEACHING

Another area of interest is the different approach of various teachers, schools and school boards to the way that computers are maintained, student access is controlled, and the way that the courses are presented. At Lorne Park Secondary, the school that I attend in Mississauga, the computer courses are presented in fairly effective ways. Instead of having the students copy the program out of the textbook or some other source, the students are presented with a new command, structure, or technique, and then they are presented with a challenge - they must come up with a program that will accomplish a task. Examples include card games, a banking simulation, generating report cards, a driving test, a slot machine. When the program is finished, the student is usually quite pleased with what he/she has produced. The main advantage of this approach is that learning can actually be FUN !!! However, it isn't quite this easy; the students are also required to DESIGN the programs on paper, using flowcharting, pseudocode, and related techniques prior to actually using the computer. In many cases, teamwork is required. This tends to improve the quality of the final program, and it helps the students to understand the necessities of working together. No more than 30% of the final exam at Lorne Park is written on paper. The remainder of the marks are given for creating and debugging a program and for documenting it. In some cases, the students are provided with a partially complete program which they must complete and expand upon.

One of the difficulties encountered at the present time is that a large number of teachers in the schools have very little experience with computers. The Peel• Board offers a course called Computers and Educators, which the teachers are encouraged to take. The course has an introduction to computers and terminology, and it leads up to small scale programming in BASIC. This and the few other activities are helping to make staff more comfortable with the computer. One attitude that I strongly disagree with is the belief that teachers should be given a year off, and a computer to work with. First of all, this is **VERY** expensive, and secondly, how many of the teachers would actually spend their time with a computer?

With respect to computer maintenance, we have never had a piece of equipment out of service for longer than three days. The main reason for the good service record is the fact that there are people who actually care about the equipment. Myself, and two fellow hackers try to make sure that all of the teachers know how to check the power cord, the fuse, and the power switch. We also make sure that they know how to get keys unstuck and other things of that nature. If a larger problem develops, of an electrical form, one of us can usually fix it, sometimes with a little help from the electronics shop. If it is something too big, then the machine is sent for servicing. Of the schools in our board, Lorne Park has one of the lowest service bills, if not **the lowest.**

A fair bit of concern has been expressed about the Canadian Educational Microcomputer (CEM). This is the machine which the Government of Ontario has contracted. Some people have expressed concern about the machine and when it will appear. For starters, the first of the machines with the languages from Waterloo will have been delivered by September 1st, 1983. Secondly, the machine actually does work, and is ready to be set up for mass production. Also, the price is considerably different from what you may have been led to believe. A class set of C-64s would cost $12,000., according to Mr. Bradley. That makes for about 20 computers.

BUT let's add in 20 monitors. That is about $8,000. That comes to a total of around $20,000. The cost of a CEM is about $2,500, which includes a built-in monitor and trackball. For 20, that is $50,000. But, the Ministry of Education is funding 75% of the initial purchase, leaving us a total of $12,500. Sounds like a good deal for Telidon graphics, a 99 key keyboard, 128 K RAM, trackball, network interface, and an Intel 80186 16 Bit Very Fast processor. Also, it runs enhanced versions of the industry-standard Waterloo languages available on the SuperPET. To add a filesaver, which contains a 10 Megabyte Hard Disk, and a 1 Megabyte floppy, which can be used by all the machines on the network, add $8,000. Now tell me which is the better buy!

Quite simply, the computer is doing fairly well in education, and the situation is getting better. Hopefully, many of you will have similar good experiences in the future, and will appreciate the improvements that are occurring in the educational system.

Audio Teach with Commodore Computers

By Ron Byers, Truro, N.S.

One of the things you may want your computer to do is to talk to your students. While this is possible through the use of expensive speech synthesizers, an easier solution for the average teacher may be to combine the audio instruction capabilities of a tape recorder with the control functions of a micro-computer.

Through the use of the user port on your PET, VIC 20 or C64 computer, the switching on and off of external devices under program control is fairly simple. A POKE statement to the user port will tell the computer to communicate with an external device. Another POKE can tell the device to turn on or off; your program and the computer's built-in clock can tell the device which to do and when to do it.

Can you connect your tape recorder directly to the user port? Well...not quite. However, don't despair. The interface to do the job is quite simple to build. (This might be the time to search among your students' parents for an electronics enthusiast with an itchy soldering gun trigger finger.) A trip to a well-known electronics parts 'shack' with a few of your deflated dollars will provide the necessary hardware, with the exception of the connector for the user port. This connector (e.g., AMP 530654-38205) should be available from your Commodore dealer. Refer to the 'Building the Interface' section for construction details.

Assuming that you have been able to build the necessary interface, or have it built, the next step is to obtain the program. This program is really two programs in one. One part will be seen only by the teacher and the other by the students. The program you use is designed to time your audio instructional tape so that it will stop at the right places in preparation for the correct answers which you place in the program for each student's response. The instructions given in the teacher's part of the program explain how to do this, as in the print-outs shown.

This program would be particularly suitable for spelling lessons, foreign language, or any instruction which is best done through an audio presentation followed by student responses. It may look complicated at first, but it really is not difficult after the first time. Just make a tape recording using many pauses for student responses. Each segment of your audio tape must end with a question or specific request for student input. Jot down the word(s) you want the student to type in when he/she gets to that point in the tape. You may wish to use a bell or some sound cue to indicate where the tape is stopped for each section; however, the sound of the pause control being pushed may be all that is needed.

When you run the teacher part of the program, just follow the instructions and be sure to press 'E' promptly at the end of each of your messages and 'L' after the last message. Instruction C. must be followed exactly, since this is the part which creates the DATA statements to make the student program work properly. All of the data statements will be listed for you, and you must press CLR/HOME (don't press shift) and then press RETURN once for each line. Delete line 110 and the program will be ready to SAVE for student use.

You might want to SAVE the student program at the beginning of the other side of your audio tape. Give it a name determined by the content of the lesson.

With the changes mentioned in the REM statements near the beginning of the program, this teaching aid will work on a PET, VIC or C64. As with any A/V media, try the finished program yourself before class use.

SCREENS FROM TEACHER'S PROGRAM

INSTRUCTIONS A.

```
THE FIRST STEP IN PREPARING A TAPE FOR
THIS PROGRAM IS TO RECORD YOUR MESSAGES
AND/PROGRAMS ON THE AUDIO TAPE
RECORDER.
WHEN RECORDING YOUR MESSAGE USE THE
PAUSE CONTROL AT THE END OF EACH
MESSAGE.
EACH MESSAGE MUST END WITH A QUESTION
OR SOME PROMPT FOR STUDENT INPUT.
JOT DOWN THE WORD(S) YOU WANT AS THE
STUDENT RESPONSE TO THAT MESSAGE.
THEN RELEASE THE PAUSE CONTROL AND
REPEAT THE PROCEDURE FOR THE NEXT
MESSAGE AND STUDENT RESPONSE.

YOU MAY ONLY MAKE 20 MESSAGES BUT THEY MAY BE AS
LONG OR SHORT AS YOU LIKE.

WHEN YOU HAVE FINISHED RECORDING ALL OF YOUR
MESSAGE TAPE,

PRESS * TO CONTINUE.
```

It may be possible to use this program in an unexpanded VIC if the lines below are left out and Instructions A, B and C from this article are used instead of having them on the screen. Lines to omit include the following: 20, 30, 700 to 860, 1200, 1330, 1380, 1500, 1640, 150, 1660, 1720, 1730, 1980 to 2120, and 2150 to 2240.

INSTRUCTIONS B.

```
AUDIO TAPE INTERFACE (BLACK BOX)
MUST BE PLUGGED INTO THE USER PORT
(NEXT TO THE CASSETTE INPUT).

THE MINI PLUG FROM THE BLACK BOX
MUST BE PLUGGED
INTO THE REMOTE INPUT ON THE
AUDIO TAPE RECORDER.
IT WOULD BE BEST TO TURN THE COMPUTER
OFF AND DO THIS AND THEN LOAD AND RUN
THE PROGRAM AGAIN.

TURN BLACK BOX SWITCH ON.

REWIND THE AUDIO TAPE.

PRESS * TO CONTINUE.
```

```
PRESS PLAY ON THE
AUDIO TAPE RECORDER.
AND THEN PRESS *.
```

```
WHEN YOU ARE READY
FOR ME TO LISTEN TO
YOUR MESSAGE,
PRESS B.

WE WILL CALL THIS
MESSAGE 1.
```

```
LISTENING TO MESSAGE 1

PRESS E WHEN YOU HEAR
THE SOUND WHICH SIGNALS
THE END OF
MESSAGE #1.
```

```
TYPE THE WORD YOU WANT
FOR THE STUDENT
RESPONSE FOR
```

```
PRESS L IF THIS
IS YOUR LAST MESSAGE.

PRESS C TO CONTINUE.

STOP TAPE.
```

INSTRUCTIONS C.

```
READ CAREFULLY.
AFTER YOU PRESS C
I WILL LIST THE DATA
FOR THE MAIN PROGRAM.
YOU MUST PRESS
THE HOME KEY AND
THEN PRESS RETURN
ONCE FOR EACH LINE.

DON'T GOOF IT UP.
YOU ONLY GET ONE
CHANCE TO DO IT!
ALSO YOU MUST
SAVE THIS PROGRAM
BEFORE STUDENT USE.
DELETE LINE 110
BEFORE SAVING.

READ THIS AGAIN
AND MAKE NOTES!
THEN PRESS C.
```

SCREENS FROM STUDENT'S PROGRAM

```
ASK YOUR TEACHER
IF THE AUDIO TAPE
IS READY. IF SO,
BE READY TO LISTEN
TO THE AUDIO TAPE.

PRESS PLAY NOW.

PRESS C TO BEGIN.
(ADJUST VOLUME WHEN
THE TAPE BEGINS.)
```

```
LISTENING TO
MESSAGE #1
```

```
TYPE YOUR ANSWER
AND
PRESS RETURN
```

```
SORRY!

THAT'S NOT THE
ANSWER I'M EXPECTING

PRESS SPACE AND
TRY AGAIN.
```

```
THE ANSWER I WANT IS

PET

TYPE THIS ANSWER
PLEASE.
```

```
RIGHT!

WELL  DONE!

PRESS  SPACE
```

```
TURN  OFF  TAPE  PLEASE.

YOU  HAVE
FINISHED  THIS
PROGRAM.
HERE  ARE  THE  WORDS
YOU  MISSED:
PET
VIC

THANKS  FOR  YOUR
EFFORTS!

YOU  HAD 2 ERRORS
IN  6 QUESTIONS.

READY.
```

BUILDING THE INTERFACE

Use of a dual IC board and an IC socket will simplify construction. The 7404 and the relay can each be mounted on this and some extra holes should be available for the transistor, diode and resistor. A LED may be added to give a power-on indicator. Depending on the size of the relay, it may be possible to use a cassette tape case as a box for the circuit. However, a larger box would allow the batteries to be mounted inside.

Four penlight cells in a battery holder may be used as the power supply. This gives 6V rather than the recommended 5V, but has worked okay so far. It should be possible to obtain the 5V from pin 2 on the C64, although this has not been tried as yet.

PARTS LIST

Circuit board (dual IC)	#276-159
NPN Transistor	#276-2009
Diode IN34A	#276-1123
Digital IC hex inverter (7404)	#276-1502
Dip socket (14 pin)	#276-1999
Switch (spst)	#275-0862
Battery snap	#270-325
Battery holder (4 AA)	#270-391
Submini phone plug	#274-291
Resistor 1K	#271-023
SPDT Dip relay (5V)	#275-243
User port connector	#AMP 530654-38205
Hookup wire and suitable plastic box	

*Considerable savings may be realized by ordering parts from a mail-order electronics supplier.

THEORY OF OPERATION

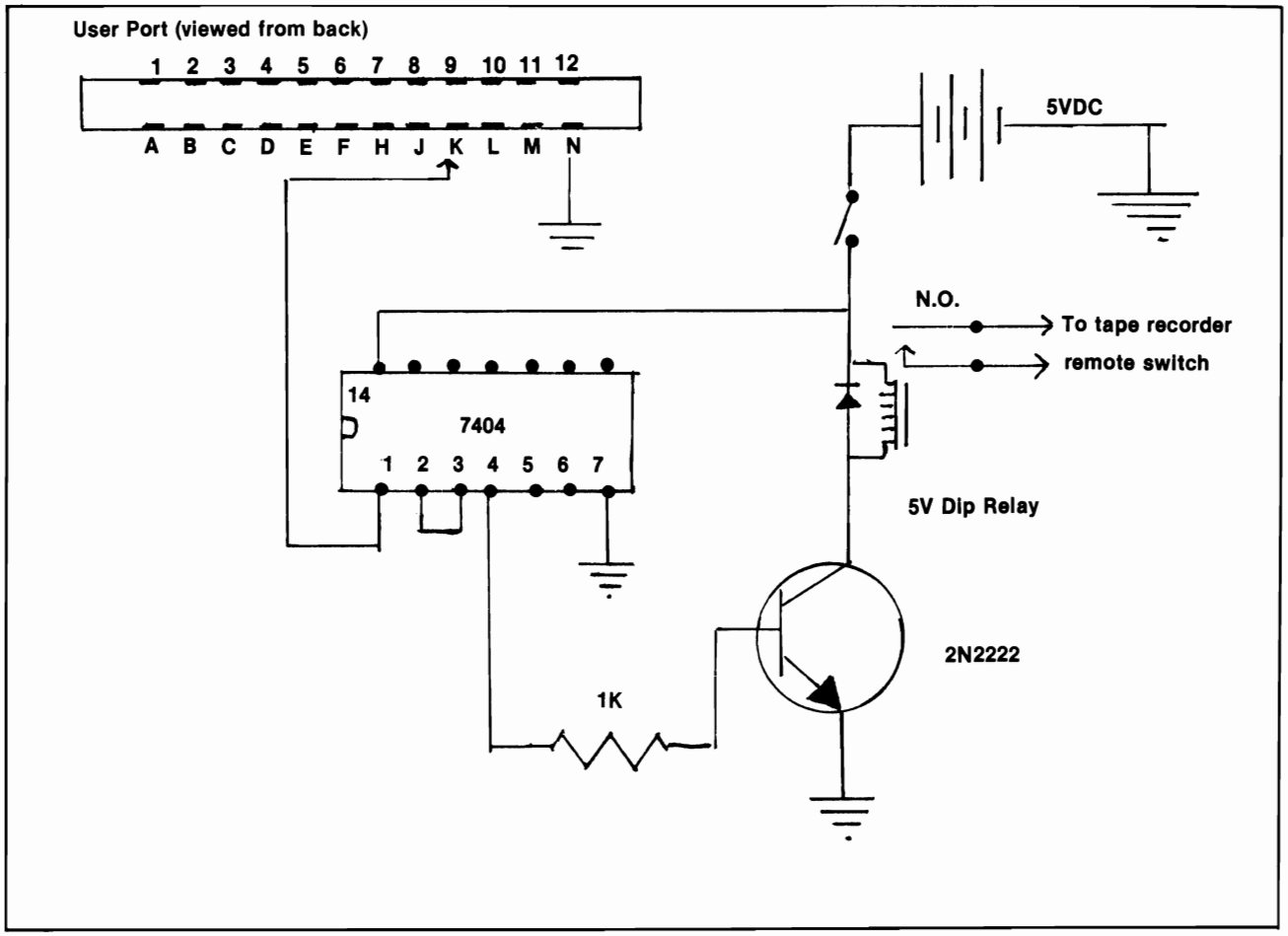
The 7404 is used as a buffer. It, together with the relay, isolates the computer from the device to be controlled. A logic 1 (high) from the computer will, after transferring the signal through two inverters to the IC, turn the transistor on and cause current flow through the relay. This, in turn, allows the tape recorder to play.

When the computer is turned on, the data direction register (DDRA), which controls whether the data lines to the user port are inputs or outputs, is set for input on all data lines (address $E843 or 59459 decimal in the PET). Near the beginning of the program, the statement POKE E, 255 sets the DDRA to output data. Data, if any, in the output register (ORA at address 59471 decimal) will be sent out to devices connected to the user port. The interface is connected to pin "K" or bit 6. Therefore, if the ORA is poked with a number which puts a 1 in binary bit 6, e.g., 64 decimal, the interface will turn the tape recorder on. A zero will turn it off.

The program is written so that the computer will stay in a loop for a period of time while the internal clock counts and waits for the teacher to indicate the end of each taped message. The numbers thus generated are used in the student program to determine when to POKE the ORA for a 0 to stop the tape at the right places in the tape.

HAM NOTE

For the benefit of amateur radio operators who might read this, I will mention that this same interface may be connected to a transmitter key to send morse code from the PET keyboard. Use of the MORSE WITH PET program from Kinetic Designs PET library, 401 Monument Road, #171, Jacksonville, Florida 32211 (or similar program) will provide send and receive capability. This program will come with a diagram for a simple interface, also using one IC and one transistor, which will enable one to copy cw on the PET (see Kilobaud Magazine, November 1978). By changing two POKE values, it will work on the C64 or expanded VIC 20 as well. The two interface circuits may be packaged in the same box and use the same power supply.



LISTING

```
10 REM**PROGRAMMER**R.E.BYERS*(VE1AMU)
20 REM**200 BURNYEAT ST.,TRURO,N.S.**
30 REM**B2N 4R1**FEB.1983***
40 E=59459:P=59471:REM***PET***
50 REM VIC**E=37138:P=37136:REM VIC**
60 REM C-64*E=56579:P=56577:REM C-64*
70 N=64:Q=0:X=255
80 POKEE,X:POKEP,N:DIMT(25):DIMR$(20):DIMA$(25)
90 PRINT"■■■■■■      ■AUDIO TEACHER■   "
100 PRINT"■■■■■■■■      PRESS SPACE "
110 PRINT"■■■■■■■■■■TEACHERS USE ■*■."
120 GETS$,S$,S$
130 GETS$:IFS$=""THEN130
140 IFS$=CHR$(42)THEN170
150 IFS$=CHR$(32)THEN970
160 PRINT"■":GOTO90
170 PRINT"■■INSTRUCTIONS■ A."
180 GOSUB19■0
```

```
190 PRINT"■PRESS ■*■ TO CONTINUE."
200 GETS$,S$,S$
210 GETS$:IFS$=""THEN210
220 IFS$=CHR$(42)THEN240
230 PRINT"■":GOTO170
240 PRINT"■■■INSTRUCTIONS■ B."
250 GOSUB2140
260 PRINT"■PRESS ■*■ TO CONTINUE."
280 GETC$:IFC$=""THEN280
290 IFC$=CHR$(42)THEN310
300 GOTO250
310 POKEP,Q
320 PRINT"■■PRESS ■PLAY■ ON THE"
330 PRINT"AUDIO TAPE RECORDER."
340 PRINT"AND ■THEN■ PRESS ■*■."
350 GETC$,C$,C$
360 GETC$:IFC$=""THEN360
370 M=1
380 PRINT"■■■WHEN YOU ARE READY "
390 PRINT"FOR ME TO LISTEN TO "
400 PRINT"YOUR MESSAGE,"
410 PRINT" PRESS ■B■.":GETM$,M$,M$
420 PRINT"■WE WILL CALL THIS"
430 PRINT"■MESSAGE■ ";M :GETM$,M$,M$
440 GETM$:IFM$=""THEN440
450 IFM$<>"B"THEN 380
460 POKEP,N:PRINT"■■■■LISTENING■ TO MESSAGE ";M :TB=TI
470 PRINT"■PRESS ■E■ WHEN YOU HEAR"
480 PRINT"THE SOUND WHICH SIGNALS "
490 PRINT"THE END OF"
500 PRINT"MESSAGE #";M :GETM$,M$,M$
510 GETM$:IFM$=""THEN510
520 IFM$<>"E"THEN 510
530 T(M)=TI-TB:POKEP,Q
540 PRINT"■■TYPE THE WORD YOU WANT"
550 PRINT"FOR THE  STUDENT"
560 PRINT"RESPONSE FOR"
570 PRINT"■MESSAGE # ";M
580 PRINT"■AND PRESS RETURN."
590 GOSUB1780:A$(M)=ZI$:M=M+1
600 PRINT :PRINT:PRINT
610 PRINT"■PRESS ■L■ IF THIS "
620 PRINT"IS YOUR LAST MESSAGE."
630 PRINT"■PRESS ■C■ TO CONTINUE.":GETC$,C$,C$
640 GETC$:IFC$=""THEN640
650 IFC$="L"THEN670
660 GOTO380
670 PRINT"■■STOP TAPE.■"
680 PRINT"■INSTRUCTIONS■ C."
690 PRINT"■READ CAREFULLY■."
700 PRINT"AFTER YOU PRESS ■C■"
710 PRINT"I WILL LIST THE DATA"
720 PRINT"FOR THE MAIN PROGRAM."
730 PRINT"YOU MUST PRESS"
```

```
740 PRINT"THE ■HOME■ KEY AND"
750 PRINT"THEN PRESS ■RETURN■"
760 PRINT"ONCE FOR ■EACH LINE■"
770 PRINT"■■DON'T GOOF IT UP"
780 PRINT"YOU ONLY GET ■ONE■"
790 PRINT"CHANCE TO DO IT!"
800 PRINT"■ALSO■ YOU MUST"
810 PRINT"■SAVE■ THIS PROGRAM"
820 PRINT"BEFORE STUDENT USE."
830 PRINT"■DELETE■ LINE 110"
840 PRINT"BEFORE SAVING. "
850 PRINT"■■READ THIS AGAIN■"
860 PRINT"■AND MAKE NOTES!■"
870 PRINT"■THEN■ PRESS ■C■."
890 GETC$:IFC$=""THEN890
900 IFC$<>"C"THEN700
910 PRINT"□";
920 FORY=1TOM-1
930 PRINT(Y+1200);"DATA";T(Y);",";CHR$(34);A$(Y);CHR$(34)
940 NEXTY
950 POKEE,Q:POKEP,X
960 PRINT"□□":END
970 :          REM*MAIN PROGRAM**
980 POKEE,X:POKEP,Q
990 PRINT"□■ASK YOUR TEACHER"
1000 PRINT"IF THE AUDIO TAPE"
1010 PRINT"IS READY. IF SO.."
1020 PRINT"BE READY TO LISTEN"
1030 PRINT"TO THE AUDIO TAPE.    "
1040 PRINT"■PRESS PLAY ■NOW■."
1050 PRINT"■PRESS ■C■ TO BEGIN."
1060 PRINT"(ADJUST VOLUME WHEN "
1070 PRINT"THE TAPE BEGINS)."
1080 GETC$,C$,C$
1090 GETC$:IFC$=""THEN1090
1100 IFC$<>"C"THEN990
1110 M=1:W=1:D=1
1120 GOSUB1240:REM*** LISTEN***
1130 IFT=150000THEN1630
1140 GOSUB1340:REM*** TYPE ANS.**
1150 GOSUB1780:REM MULTI-GET***
1160 IFZI$=A$(M)THEN1490:REM*RT.ANS***
1170 GOSUB1390:REM*WRONG ANS****
1180 IFZI$=A$(M)THEN1160
1190 GOTO1140
1200 :REM**DATA GOES HERE *****
1201 DATA 150,"PET"
1220 :
1230 DATA 150000,"LAST"
1240 PRINT"□■LISTENING TO"
1250 PRINT"MESSAGE #";M
1260 READT,A$:T(M)=T:A$(M)=A$
1270 IFT<150000THEN1290
1280 RETURN
```

```
1290 POKEP,N:TB=TI
1300 TA=TI-TB
1310 IFTA<TTHENTA=0:GOTO1300
1320 POKEP,Q:RETURN
1330 REM************************
1340 PRINT"■■■TYPE YOUR ANSWER■"
1350 PRINT"      AND"
1360 PRINT"  ■PRESS RETURN"
1370 RETURN
1380 REM************************
1390 PRINT"■■■SORRY!■"
1400 PRINT"■THAT'S NOT THE"
1410 PRINT"ANSWER I'M EXPECTING"
1420 PRINT"■PRESS ■SPACE■ AND"
1430 PRINT"TRY AGAIN."
1440 R$(W)=A$(M):D=D+1:GETC$,C$,C$
1450 GETC$:IFC$=""THEN1450
1460 W=W+1
1470 IFD=3THENGOSUB1560
1480 RETURN
1490 PRINT"■■■RIGHT!■"
1500 PRINT"■WELL DONE!"
1510 R=R+1:M=M+1:D=1
1520 GETC$,C$,C$
1530 PRINT"■PRESS ■SPACE■."
1540 GETC$:IFC$=""THEN1540
1550 GOTO1120
1560 REM*ANS.WRONG TWICE****
1570 PRINT"■■■THE ANSWER I WANT IS■"
1580 PRINTA$
1590 PRINT"■TYPE THIS ANSWER"
1600 PRINT"PLEASE.":D=1
1610 GOSUB1780:RETURN
1620 REM**************
1630 PRINT"■TURN OFF TAPE PLEASE."
1640 PRINT"■YOU HAVE "
1650 PRINT"FINISHED THIS"
1660 PRINT"PROGRAM."
1670 PRINT"HERE ARE THE WORDS"
1680 PRINT"YOU MISSED:"
1690 FORV=1TOW
1700 PRINTR$(V)
1710 NEXTV
1720 PRINT"THANKS FOR YOUR"
1730 PRINT"EFFORTS!"
1740 PRINT"■YOU HAD";W-1;"ERRORS"
1750 PRINT"IN";M ;"QUESTIONS."
1760 POKEE,Q:POKEP,X
1770 END
1780 REM**************
1790 ZI$="":ZL=21:REM MULTI-CHARACTER GET
1800 ZI$=""
1810 GETZG$:IFZG$<>""THEN1830
1820 PRINT"_■";:GOTO1810
```

```
1830 PRINT" ■■";:Z9=ASC(ZG$):Z8=LEN(ZI$)
1840 IFZ9=20THEN1920
1850 IFZ9=13THEN1960
1860 IFZ8=ZLTHEN1810
1870 IFZ9=34THEN 1810
1880 IFZ9>90ORZ9<32THEN1810
1890 PRINTZG$;:ZI$=ZI$+ZG$
1900 GOTO1810
1910 IFZ8=0THEN1810
1920 PRINT"■■ ■■";
1930 IFZ8=1THEN1800
1940 ZI$=LEFT$(ZI$,Z8-1)
1950 GOTO1810
1960 IFZ8=0THEN1810
1970 RETURN :REM********
1980 PRINT"■THE FIRST STEP IN PREPARING A TAPE FOR  THIS";
1990 PRINT" PROGRAM IS TO RECORD YOUR MESSAGES AND/OR";
2000 PRINT" QUESTIONS ON THE AUDIO TAPE      RECORDER."
2010 PRINT"WHEN RECORDING YOUR MESSAGE USE THE     PAUSE ";
2020 PRINT" CONTROL AT THE END OF  EACH      MESSAGE."
2030 PRINT"EACH MESSAGE MUST END WITH A QUESTION ORSOME ";
2040 PRINT"PROMPT FOR STUDENT INPUT."
2050 PRINT"■JOT DOWN■ THE WORD(S) YOU WANT AS THE    STUDENT";
2060 PRINT" RESPONSE TO THAT MESSAGE."
2070 PRINT"THEN RELEASE THE PAUSE CONTROL AND      REPEAT";
2080 PRINT" THE PROCEDURE FOR THE NEXT MESSAGE AND STUDENT RESPONSE."
2090 PRINT"■YOU MAY ONLY MAKE ■20■ MESSAGES BUT THEY";
2100 PRINT"  MAY BE AS LONG OR SHORT AS YOU LIKE."
2110 PRINT"■WHEN YOU HAVE FINISHED RECORDING ■ALL■ OF YOUR";
2120 PRINT" MESSAGE TAPE,"
2130 RETURN
2140 REM*SET-UP INSTRUCTIONS********
2150 PRINT"■■■■AUDIO■ TAPE ■INTERFACE■ (BLACK BOX) "
2160 PRINT"MUST BE ";
2170 PRINT" PLUGED INTO THE USER PORT (NEXT TO THE  CASSETTE INPUT)."
2180 PRINT"■THE MINI PLUG FROM THE BLACK BOX MUST BEPLUGED";
2190 PRINT" INTO THE ■REMOTE■ INPUT ON THE     AUDIO TAPE RECORDER."
2200 PRINT"IT WOULD BE BEST TO TURN THE COMPUTER "
2210 PRINT"■OFF■ AND DO THIS AND THEN ";
2220 PRINT"LOAD AND RUN   THE PROGRAM AGAIN."
2230 PRINT"■TURN BLACK BOX SWITCH ■ON■."
2240 PRINT"■REWIND THE AUDIO TAPE ."
2250 RETURN
```

Alas, as the price of micros drops, the price of micro repairs rises...in other words, the cost of fixing a computer that's DOWN is UP!

— Ylimaki

TORPET Education Disk

By G.R. Walter, Proton Station, Ont.

These programs are on The Best of Torpet Disk #5

AN EASY WAY TO LEARN PROGRAMMING

The TORPET Education Disk (No. 5 of the Best of TORPET series) contains educational programs written by Prof. Ponzo of Waterloo University. They are programs which teach you how to program your C64 in BASIC and machine language, and how to create and use SPRITES and the various graphics modes of which the C64 is capable.

DISK CATALOG

"PONZO TUTOR-1" (teaches BASIC)
"PONZO TUTOR-2" (teaches BASIC)
"PONZO TUTOR-3" (teaches BASIC)
"PONZO TUTOR 4" (teaches BASIC)
"PONZO TUTOR 5" (teaches Machine Language)
"PONZO TUTOR 6" (teaches Machine Language)
"PONZO TUTOR 7" (teaches Machine Language)
"SPRITES TUT-1" (teaches how to create/use sprites)
"SPRITES TUT-2" (teaches how to create/use sprites)
"GRAPHIC TUTOR" (teaches how to get/use the various graphics modes)
"SMOOTH SCROLL" (example program)
"EXAMPLE SPRITE" (example program)
"MOVE SPRITE" (example program)
"EXPANDED SPRITE" (example program)
"MAKING SPRITES" (example program)
"PROGRAM CHARS" (example program)
"MULTICOLOR CHARS" (example program)

All of the programs use the same technique for teaching you. They show you something about your C64, then they give you an example or two (which they work through with you so that you understand what is happening). In addition, in the 'teach BASIC' sections, you usually are then given an opportunity to actually try out what you have learned, i.e., you exit the program, experiment with what you have found out, and then re-enter the program which you exited by pressing the "@" (at sign) — not by typing RUN!

These programs will all work with the C64 LINK by RTC, but it is not necessary for their use.

(Note: some of the figures, e.g., top of memory = 32768, are the PET figures, not the C64 figures. In this instance, the slip-of-the-keyboard doesn't matter, but it is of interest.)

Here is a brief summary of what each of the programs teaches you.

PONZO TUTOR-1 explains how to do arithmetic on your C64, PRINT the results, start writing programs, use the GOTO, GOSUB, INPUT, FOR/NEXT statements, and several other statements, commands (e.g., LIST) and functions. It shows you how numeric (i.e., numbers only) variables work and how you can use them. You learn how to use the [DEL/INST] key for editing lines. By the time you are done with this program, you should know how to write simple little programs involving numbers that work.

PONZO TUTOR-2 shows you all of the various cursor controls, e.g., [HOME], and how you can use them. String variables, i.e., "anything" can go into these, are taught, along with most of the functions used to manipulate string variables. READing DATA is explained, as are the IF/THEN statements, among others. It also explains how the GET statement works and what the keyboard buffer is. By the time you are done with this program, you will have done a program which draws bar graphs.

PONZO TUTOR-3 requires a Machine Language Monitor to be loaded (a good one is Jim Butterfield's SUPERMON64 on the Best Programs Disk. BITs and BYTEs are explained, and you are shown how to PEEK and POKE. Certain key locations in the C64 are explained, and you learn how to use them, e.g., the C64's internal clock (variables TIME and TIME$). How to read and write data files to tape and disk is explained, as is how you use Random Numbers. Several other minor things (such as LISTing a program on a printer) are gone into. The Machine Language Monitor is introduced.

PONZO TUTOR-4 shows you a simple memory map of the C64 and explains the significance of some of the memory locations, e.g., top of memory pointer is .., etc. You find out how programs and string variables are stored in memory. Then you are given a little quiz on what you have learned.

PONZO TUTOR-5 introduces you to machine language, looping, the various addressing modes, and several of the commands.

PONZO TUTOR-6 teaches you more by showing you example routines from the BASIC ROMs (e.g., NEW), and by explaining exactly how they work.

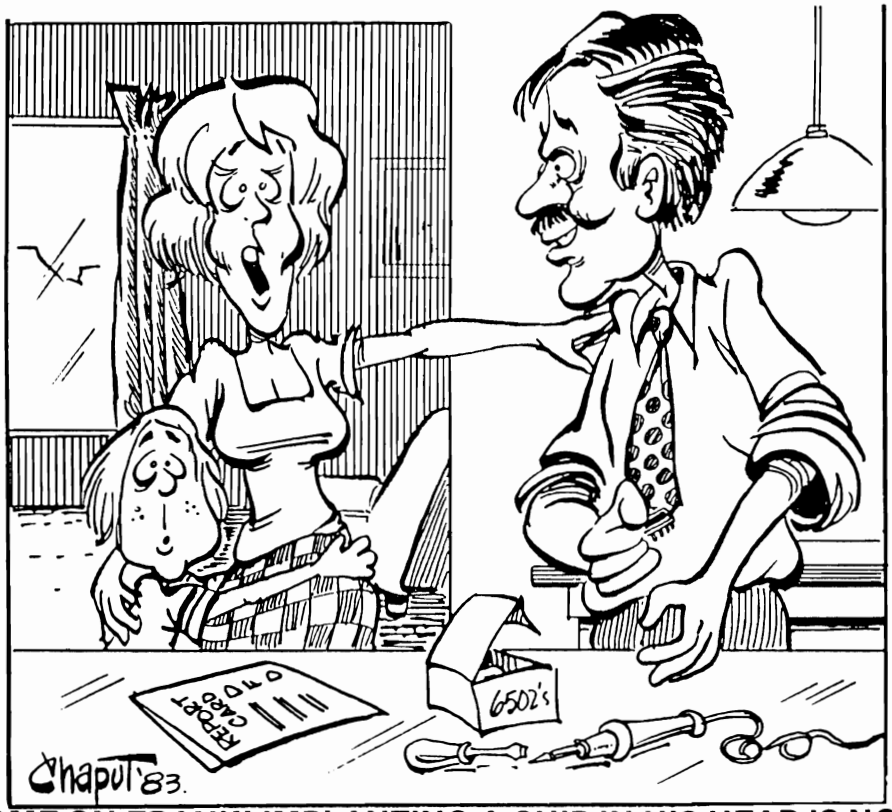
PONZO TUTOR-7 shows you the entire 6510 command set. The Status Register is explained, and you are taught how to work with it (e.g., CLearing flags and SEtting flags) and how to test it (e.g., Branching on flags).

SPRITES TUT-1 teaches you how to create sprites, work with them, and use them in programs. In short, you find out all you ever want to know about sprites!

SPRITES TUT-2 introduces multi-color sprites and how to create, work with and use them.

GRAPHIC TUTOR explains several of the various graphics modes that the C64 is capable of (e.g., multicolor mode, hires mode, etc.). You are shown how to start utilizing these special modes.

If you are a beginning C64 programmer and would like to get better, then this disk would be a valuable addition to your library.



COME ON FRANK! IMPLANTING A CHIP IN HIS HEAD IS NOT GOING TO HELP HIS SCHOOL WORK!

Commodore Introduces Home Education Programs for the VIC 20

These programs are a subset of public domain education programs listed on pages 240-260 for the C64, and available from TORPET Diskettes.

Education programs for the VIC 20 home computer/video game — a new and valuable aid for children's education — was announced today by Commodore Computers.

"The Computer Educator System was designed primarily to assist children at various grade levels with their school work by providing a fun and stimulating learning series to help them become more involved in education,", said James Copland, National Sales Manager, Consumer Division. "Many adults, however, will also learn a great deal from these programs."

The Computer Educator Programs offer series in science, language, social science and math. The science series is developed to appeal especially to people from the third to the tenth grades. The language series is designed to improve grammar, spelling and vocabulary. The social science series helps impart a greater understanding of key events that have shaped world history and important facts regarding world geography. The math series is aimed at increasing students' general proficiency in mathematics.

"The year 1983 is Commodore's 25th anniversary in Canada where it was founded in Toronto. We intend to celebrate this occasion by introducing more new hardware and software products than ever before, especially for the 200,000 VIC 20 units we expect to make in Canada and sell worldwide this year", said Copland.

The VIC 20 Computer Educator Systems come in packs of six tapes each with course books where relevant, for a suggested retail price of $69.95 per pack.

In more detail, the series content is as follows:

SCIENCE SERIES

BOOK 1: ELEMENTARY SCIENCE

Grades one through six. These programs deal with the life sciences and explore the planets, the atom, cells and other interesting science phenomena, at the lowest level.

BOOK 1: PHYSICS

Explores electricity through specific programs relating to resistors, capacitors and power. Motion and force are also examined as is the subject of levers. Generally, this is an introductory program to high school physics.

BOOK 1: CHEMISTRY

This book introduces the student to chemical phenomena. For the first time he will learn about the elements, ions and compounds. We also introduce the student to chemical equations in preparation for a more extensive study of this subject in high school.

LANGUAGES

BOOK 1: ENGLISH

Elementary English; grades one through six, is designed to develop good basic reading and writing skills — spelling is emphasized.

BOOK 2: ENGLISH

Intermediate English; deals with the sentence and its parts and is designed to build a better vocabulary. The student is also introduced to Shakespeare.

BOOK 1: FRENCH

This is an elementary course in French. Since the basis of a foreign language requires an extensive vocabulary, all six tapes in this book are designed to increase the student's proficiency with French and deals exclusively with vocabulary. Some exercises are French to English translations and some are English to French.

SOCIAL SCIENCE SERIES

BOOK 1: SOCIAL SCIENCE

Grades one through six, examines Canada, the North American continent, the North American explorers and the founding nations.

BOOK 1: ELEMENTARY GEOGRAPHY

This book is designed to familiarize the student with world geography, with specific emphasis on Canada and North America.

BOOK 1: HISTORY

Introduces the student to Early Man; examines the Egyptian, Greek and Roman Empires. It takes the student through the Middle Ages to 1500 B.C. Basically the programs trace the development of man from his earliest beginnings through the Middle Ages.

MATHEMATICS

BOOK 1: ELEMENTARY MATHEMATICS

Grades one through six, is an introduction to basic mathematics. The four elementary operations in mathematics are addition, subtraction, multiplication and division. This book deals specifically with these, as well as their application to fractions and percent.

BOOK 2: ADVANCED ELEMENTARY MATHEMATICS

Grades six through ten. In this book the student is introduced to algebra and geometry and, while these programs are designed only as an introduction to these subjects, they nevertheless represent a substantial step forward in the student's general knowledge of mathematics.

BOOK 3: ALGEBRA

For students grades eight through eleven, is an introduction to algebra. Algebra is really an extension of arithmetic. While the same operations can be used in algebra, in arithmetic, numbers always have specific values while in algebra there is no need for a specific value. This book deals with basic algebra and the operations that are fundamental to this subject.

The above Commodore Education series is available from Commodore dealers (you will have to check with them for the price as they vary). However, these programs are a subset of the Public Domain Education Series which is **listed directly following** and which is available from several sources. One method of obtaining the series, since the complete education series is all public domain, is to either copy the programs from or to your friends.

If you don't happen to be able to get the diskettes from someone else you can get them by writing to:

TORPET DISKETTES
Horning's Mills
Ontario L0N 1J0
Canada

or

TORPET DISKETTES
1 Brinkman Ave.
Buffalo, N.Y.
14211 U.S.A.

Enclose $10 per diskette requested or $300 for the complete set of 58 education diskettes (which makes this the cheapest source we know of next to getting them from a friend).

If you wish to get the programs on tape you may get them through TPUG at 1912-A Avenue Rd., Toronto M5M 4A1, Canada. If you are not already a member you will be required to join. They also have the programs available on diskette. You should write them for costs and details as they vary from time to time.

The Education Series is also available from Aurora Software, P.O. Box 1394 Haileybury, Ontario, P0J IK0, Canada.

(E)AA - Administration

Name of Program	Cat	Grade	PST	Mem	Description
ANALYSIS.40	O		000	16k	This program takes a set of marks and calculates median, average, standard dev. and students passing/failing.
ANSWER BOX.40	U	JI	202	16k	This is a universal quiz-making utility program. Answers are stored in data lines; question worksheet required.
DOG.40	S	I	100	32k	User becomes a science teacher facing a student who wants to perform exploratory surgery on a live dog.
EXAM2.40	U	C	000	16k	Teacher enters exam marks with weightings as percents or actual marks; program determines student's final mark.
FIGHT.40	S	I	201	32k	Simulation of a hostile student-teacher confrontation in which the user takes the teacher's role.
GRADES.40	U	C	000	32k	Allows a teacher to order and print out student marks by name and grade.
QUIZFRAMEWORK.40	U	C	000	16k	Enables the teacher to construct a multiple-choice quiz.
READABILITY.40	U		503	32k	This program takes a sample from a text and determines the readability level using standard measures.
SCHOOL-MARM.40	DG	PJ	400	16k	A useful fill-in-the-blanks quiz with hints, designed for teacher adaptation to various subjects and levels.
SEX ED.40	S	I	100	32k	Simulates potential situations faced by a teacher who plans to show a childbirth film in a sex education class.

(E)BA - Business

Name of Program	Cat	Grade	PST	Mem	Description
ACCOUNTING.40	DT	S	502	32k	An excellent tutorial on basic accounting practices.
AMORT'N TABLE.40	U	S	000	16k	Constructs an amortization table from user-input data; useful for a homeowner or for checking student tables.
AMORTIZATION.40	U	IS	300	16k	Program calculates payments and amortization periods for loans, based upon user-input data.
BONDS.40	U	JIS	300	16k	Program calculates present value of bonds, including coupon values, etc.
CALENDAR.40	U		000	16k	Prints out a calendar for any given month and year.
COMMODITY.40	S	I	200	16k	This program simulates the buying and selling of commodities.
COMP TYPING.40	D	I	322	16k	A simple variable-speed typing drill. User types letters printed by computer; mistakes are highlighted at end.
COSTGOODSSOLD.40	DT	IS	320	16k	Tutorial/drill on calculating CGS with 4 types of inventory calculatiions (FIFO, LIFO, Average, etc.).
DATE.40	U		000	16k	Finds a date 'n' days from a given base date; limited application.
DEBIT&CREDIT.40	DS	S	201	16k	Given sample situations, the student must correctly debit and credit accounts.
DEBITCREDIT.40	D	S	201	16k	A program of 10 questions that test a student's accounting skills.
DEPREC&PAYMT.40	U	S	000	16k	User inputs data and the computer calculates depreciation rates and payments.
DEPRECIATION.40	U	S	000	16k	Prints a depreciation schedule using straight line, sum of digits and double declining depreciation methods.
F.I.F.O.40	TU	IS	200	16k	Calculates value of inventory using 'first-in-first-out' method (F.I.F.O.).
GROSS PAY.40	D	I	200	16k	A program designed to drill the student on various simple salary problems.
ICE CREAM.40	GS	JIS	222	16k	Student manages an ice-cream parlour, attempting to maximize employee/customer satisfaction and minimize costs.

(E)BB - Business

Name of Program	Cat	Grade	PST	Mem	Description
INVESTMENTS.40	U	S	300	16k	Calculates regular withdrawals, initial/minimum investment, effective/nominal interest, investment value, etc.
KEYBOARD TEST.40	D	I	311	16k	A program which drills various sections of the PET keyboard as selected by the user.
LEMONADE.40	GS	IS	302	16k	Student operates a lemonade stand for 10 weeks, taking into account costs, price, quantity and other variables.
LIFE TABLES.40	U	IS	200	16k	Calculates life insurance and annuity tables for any given interest rate.
MARKET CRASH.40	S		100	32k	A stock market simulation.
MARKET.40	GS	IS	320	16k	User manages a production company by determining production and advertising budgets and setting retail prices.
MONEY FLOW.40	GS	JI	202	16k	Student traces the flow of money from household to business to government.
MORTGAGE.40	U	S	000	16k	This program produces a mortgage table which would be of use to a homeowner or business student.
OBJECTIVE 1.1.40	DT	IS	302	16k	Program presents a lesson on the Balance Sheet, then tests the student on it.
PORTFOLIO.40	U	IS	300	16k	Program keeps track of stock options and the total value of a portfolio.
SIMP INTEREST.40	D	I	200	16k	Presents a variety of simple interest problems for the student to solve.
STOCK MARKET.40	GS	IS	200	16k	Student buys and sells 5 stocks as the prices randomly fluctuate.
STOCK.40	S	I	000	16k	A simple game that simulates stock market activities.
TYPING DRILL.40	D	JI	200	16k	Designed to drill students on finger reaches and familiarize them with the location of graphics characters.

(E)BC - Business

Name of Program	Cat	Grade	PST	Mem	Description
TYPING.40	T	ISC	292	32k	A very good typing drill with plenty of data for practice, including 700 common words.
WATER II.40	GS	IS	320	16k	A water resource management simulation. Student must manage water supply for a town during a drought.

(E)CA - Computer Science

Name of Program	Cat	Grade	PST	Mem	Description
BASE CONV.40	U	JI	200	32k	Converts decimal, Roman numeral, hexidecimal, binary and BCD numbers one into another.
BINARY.40	U	IS	200	16k	Converts decimals in the range from 0 to 65536 into 16 bit binary numbers.
COMMANDS.40	DT	JI	202	16k	A tutorial and drill on BASIC concepts.
COMP CONCEPT.40	ST	PJ	000	16k	A virtual machine simulation preceded by a short tutorial.
COMP. HISTORY.40	D	I	204	16k	Program quizzes the student on the history of computers.
COMPUTING.40	D	JIS	266	16k	A drill on basic computer knowledge, but adaptable to any subject.
DEMO SORT.40	S	IS	500	16k	Demonstration of a sort called 'selective replacement'.
DISK CMD.40	T	JI	100	16k	A tutorial on Basic 4.
FEATURES QUIZ.40	DT	JI	222	16k	A tutorial and quiz on the basic features of Commodore computers.
HEX DEC.40	U	IS	000	16k	Converts hexidecimals to decimals and vice versa.
HEX DEMO.40	U	SC	300	16k	Converts decimal numbers between 0 and 255 into hexidecimals, showing high and low nybbles.
HYPO ASSEM.40	ST	IS	600	32k	An introduction to assembler language. Runs a virtual machine with a small language set; good practice.
HYPO II.40	SUO	SC		32k	Allows the student to program in simulated machine language and execute programs step by step.
PETUNIA.40	T	JI	100	16k	Allows a schematic of a petunia music box and instructions on how to use it.
PILOT.40	S	JI	400	32k	A simple 'pilot' language interpreter including edit, list, load, save and run commands.

(E)CB - Computer Science

Name of Program	Cat	Grade	PST	Mem	Description
POGO.40	U	JI	800	32k	A version of 'LOGO' using character graphics. Allows definition of 'Macro' (subroutines) in a limited way.
SIMULATION.40	ST	I	000	16k	Simulates a small BASIC program on a virtual machine with simple internal architecture.
TURTLE 1.40	P	JI	900	16k	This program mimics 'LOGO' turtle graphics using PET graphics. Draws in a limited number of directions.

(E)D1 - COMMODORE 64

Name of Program	Cat	Grade	PST	Mem	Description
MC.MASTER CAT.					Disk catalogue program designed to work with 4040 disk drive.
MC.MAKE MASTER					Combines the disk and program name files & into 1 MASTER file.
COMMANDS					Do not load this program. See the following 3 programs.
COM.TEXT DEMO					Demonstrates text manipulation on the C-64 screen.
COM.SORT DEMO					Demonstrates sorting. A no. of student records are displayed & then sorted using different parameters.
COM.HOW FAST					100 random names are created and then sorted.
HR.GRAPHICS PAL					Assembler version of machine language program, HR.GRAPHICS OBJ
HR.GRAPHICS INST					Instructions on how to use the hires package HR.GRAPHIC OBJ.
HR.GRAPHICS OBJ					Allows you to draw on the hires screen and SAVE your screens.
HR.GRAPHICS DEMO					LOADS in the graphics package and demonstrates a hires screen.
HR.GRAPHICS LOAD					LOADS a hires screen.

CONT. (E)D1 - COMMODORE 64....

HR.PICTURE 1	Demo hires screen is LOADed auto matically by HR.GRAPHICS LOAD.
HR.HIRES TO 1525	Print a hires screen to your 1525 printer.
MU.PLAYER PAL	Assembler version of the machine language program, MU.MACHINE OBJ
MU.C64 MUSIC	Instructions & demos for music package.
MU.MACHINE OBJ	Allows you to create & SAVE musi cal pieces with simple commands.
TEXTMASTER	Simple wordprocessing package which allows you to create, edit, SAVE and LOAD documents.
TM.INSTRUCT 1	Detailed instructions--TEXTMASTER
TM.INSTRUCT 2	" " "
TM.INSTRUCT 3	" " "
AN.INSTRUCTIONS	How to use the animation package.
AN.ELLIPSE	Subroutine LOADed automatically by AN.ANIMATION DEM.
AN.WIBBLE	Subroutine LOADed automatically by AN.ANIMATION DEM.
AN.ANIMATION PAL	Assembler version of machine language program AN.ANIMATION OBJ
AN.ANIMATION OBJ	Create and SAVE quarter graphics screens and animate them.
AN.ANIMATION DEM	Load for a demo of some screens created by the animation package.

(E)D2 - COMMODORE 64

Name of Program	Cat	Grade	PST	Mem	Description
MAIN MENU 64					Instructions for Adventure Pack Reading Series
THE GAME.1					Story is displayed for reading, followed by 3 short tests.
CHOCOLATE GOO.2					Story is displayed for reading, followed by 3 short tests.
MONSTER WAVE.2					Story is displayed for reading, followed by 3 short tests.
FIREFIGHT.3					Story is displayed for reading, followed by 3 short tests.
THE HUNTER.1					Story is displayed for reading, followed by 3 short tests.
SNAIL.C64.BOOT					Will load SNAIL.C64 INST and SNAIL.C64 automatically.
SNAIL.C64.INST					Instructions for SNAIL.C64.
SNAIL.C64					Draw simple pictures on the screen by inputing the snail's direction & length of movement.
MASTERMIND					A logic game where you are challenged to break a colour code

(E)D3 - COMMODORE 64

Name of Program	Cat	Grade	PST	Mem	Description
45.64					Class mark management program called MARK MANAGER.
AVERAGE CLASS					Example of an average class
FULL CLASS					Example of a full class
CLASS OF 20					Example of a class of 20
MM.INST.0.PCLIP					Instructions for MARK MANAGER 45.64
MM.INST.1.PCLIP					Instructions for MARK MANAGER 45.64
MM.INST.2.PCLIP					Instructions for MARK MANAGER 45.64
VOWELS AT BEGIN.					Concentration type game called PHONCENTRATION, teaching phonics.
VOWELS AT MID.					Concentration type game called PHONCENTRATION, teaching phonics.
VOWELS AT END					Concentration type game called PHONCENTRATION, teaching phonics.
DOUBLE VOWELS					Concentration type game called PHONCENTRATION, teaching phonics.
CONSONANT BLENDS					Concentration type game called PHONCENTRATION, teaching phonics.
PHONCENTRATION					Concentration type game called PHONCENTRATION, teaching phonics.

(E)EA - English

Name of Program	Cat	Grade	PST	Mem	Description
A JOURNEY.40	DG	JI	400	32k	User enters computer-specified parts of speech which are then arranged into a story, with humourous results.
A OR AN.40	DT	EP	232	16k	Student completes random sentences with 'A' or 'AN'; 2 incorrect answers bring a review of pertinent grammar.
A STORY.40	DGU	PJ	400	16k	Student fills in necessary parts of speech (nouns, verbs, adjectives,etc.) and computer generates a story.
AFFECT EFFECT.40	TD	I	402	16k	Student must choose whether to use 'affect' or 'effect' in order to correctly complete a number of sentences.
ALPHA'ZATION.40	D	PJ	63	16k	A well-written drill on 'N' letter alphabetization for 3-letter words. Good graphic prompts and rewards.
ALPHA.40	D	J	202	16k	Program requires student to alphabetize random lists of 2-9 words.
ALPHABET QUIZ.40	D	P	442	16k	This program tests the student on knowledge of the alphabet and location of library books.
ALPHABET WORM.40	DT	P	222	16k	Teaches students alphabetical order as they make alphabet 'worms'; tallies no. of completed 'worms' at end.
ALPHABET.40	D	P	422	16k	The student inputs the missing letter in an alphabetical series.
ALPHABETIZING.40	T	PJ	773	16k	A well-written tutorial in alphabetization. Has 4 levels of difficulty.
ALPHABETTER.40	DT	PJ	553	16k	A very well-written drill and tutorial in alphabetization with comprehensive marking.
ALPHASHIFT.40	DG	PJI	412	32k	Computer reprints words according to a hidden rule. Excellent drill/game using logic, math and the alphabet.
ANTONYM.40	G	PJI	701	16k	A 'CONCENTRATION' game for 1 or 2 players using antonyms as the final objective.
ANTONYMS CONC.40	GDT	PJI	403	32k	A 'CONCENTRATION'-type antonym drill with musical rewards (except on C-64).
CINQUAIN.40	GT	PSI	300	32k	Allows user to write a seasonal poem of 5 lines (cinquain).

(E)EB - English

Name of Program	Cat	Grade	PST	Mem	Description
CLOZE TESTV1.40	U		557	32k	Write a cloze test to printer with x copies, answer sheet and test analysis (reading level, etc.) For teachers.
CONTRACTIONS.40	DT	I	400	16k	Teaches students the correct formation of contractions and drills them on same.
CRYPTO.40	GU	PJISC	400	16k	Student devises own cryptogram and solves it. Program can find the frequency counts of the cryptogram as well.
DEF-N-SPELL.40	U	C	462	16k	Enables teacher to create and use a file of words for testing vocabulary and spelling.
DEFIN'N MATCH.40	D	JI	202	32k	Student must match 6 words with 6 definitions within a self-appointed time limit of 15, 25 or 35 seconds.
DEFINE&SPELL.40	TU	PJISC	462	16k	Given a user-input file of words, this program creates a test on vocabulary and spelling. Useful for teachers.
DEFINITION.40	D	I	202	16k	Program selects 10 of 30 multiple-choice vocabulary questions which may be modified by teacher, if desired.
ENGLISH.40	D	JI	200	16k	Student matches a pair of words with 1 of 4 other pairs that exhibits a similar relationship.
FLASH.40	D	PJI	452	16k	A word or phrase is flashed on the screen for a specified time. User must correctly retype what was flashed.
FLOOGEPRINTER.40	U	S	060	16k	Prints out about 700 nonsense words of 3 phonemes each. Uses: character names, action words for games, etc.
GRAMMAR 2.40	DT	JI	202	16k	Tutorial/drill on parts of speech. Student categorizes 60 words as verbs, adjectives, prepositions, etc.
GRAMMAR.40	D	I	202	16k	Student names various parts of speech in higlighted sections of a sentence - nouns, verbs, prepositions, etc.
GUESSTHATWORD.40	G	I	442	16k	A vocabulary game in which the student must discover a word by guessing letters.
HAIKU.40	TU	JI	000	16k	Program explains and 'writes' Haiku, randomly selecting stored words to produce poems with a computer theme.
HANGMAN (6).40	DG	JI	500	32k	A 'HANGMAN' game with graphics and wide range of words. No. of errors allowed before 'hanging' is adjustable.
HANGMAN 1.40	G	JI	322	16k	A game of 'HANGMAN' with graphic support and 5 categories of words.

(E)EC - English

Name of Program	Cat	Grade	PST	Mem	Description
HANGMAN 2.40	G	JIS	501	32k	Traditional 'HANGMAN', complete with graphics. The computer knows 215 unusual words.
HANGMAN 3.40	DG	I	222	16k	Student tries to identify a hidden word by guessing letters; too many guesses and player is 'hanged'.
HANGMAN 5.40	DG	JI	502	16k	A traditional 'HANGMAN' game which features 40 words and allows up to 11 incorrect guesses.
HANGMAN 6.40	DG	J	422	16k	A game of 'HANGMAN'. Student inputs letters until he/she is able to guess the secret word, or chances run out.
HANGMAN 7.40	G	I	400	16k	The traditional guessing game - solve the hidden word to avoid 'hanging'.
HOMO CONC.40	G	J	202	16k	A 'CONCENTRATION' game in which the student matches up words that sound the same.
HYPHEN.40	D	I	400	16k	The student is required to hyphenate a displayed word. Words are randomly chosen and hints are available.

CONT. (E)EC - ENGLISH....

INIT DIGRAPHS.40	D	P	302	16k A simple, well-written drill on the digraphs 'TH', 'SH' and 'CH'; multiple-choice questions.
INSERT.40	DG	J	200	16k Student inserts letters from a list into a given word, making a new word. Program hints at nature of new word.
JOTTO.40	G	PJI	402	16k Player tries to match PET's hidden 5-letter word. Computer reveals number of correct letters in each guess.
KEYWORDS.40	D	J	442	16k An interesting three-part test of student spelling skills.
LETTER RECOG.40	D	P	222	16k Tests students' ability to recognize letters of the alphabet and the numerals 1-9.
LETTER SEQU.40	D	JIS	402	16k Student must recall random letters flashed on screen. Choice of up to 25 letters and 5 flash speeds.
MACBETH QUIZ.40	D	IS	402	16k Program poses questions on the content of Shakespeare's 'MACBETH'; student has 3 chances to answer correctly.
MADLIB.40	DGU	PJ	400	16k Enables student to create 3 short stories by inputting nouns, adjectives, verbs, etc. requested by the program.
MATCH LET.40	D	P	202	16k This program drills beginning students of the alphabet on letter matching.

(E)ED - English

Name of Program	Cat	Grade	PST	Mem	Description
MATCH MEAN 5.40	D	J	202	32k	Student has to match a given word with its synonym, chosen from a list. Grade 5 level, 30 questions.
MATCH MEAN 6.40	D	J	202	32k	Student has to match a given word with its synonym, chosen from a list. Grade 6 level, 30 questions.
MATCH MEAN 7.40	D	J	202	32k	Student has to match a given word with its synonym, chosen from a list. Grade 7 level, 30 questions.
MEDIAL VOWELS.40	D	PJ	402	16k	After reading a sentence, the student chooses the best word to fill in the blank.
MISSING LET.40	D	EP	200	16k	Computer displays an alphabet missing 1 letter, which the student must enter. Used letters are not repeated.
MISSPELLING.40	D	I	402	16k	Student is given a list of 5 words, one of which is misspelled. This word must be identified and corrected.
MM ADVFORMS.40	D	PJ	202	16k	Mr. Mugs: Drilling students on correct application of adverb forms. Refer L6 P201: 'IT'S SATURDAY'.
MM AVB FORMS.40	D	P	202	16k	Mr. Mugs: Drilling students on correct application of adverb forms. Refer L5 P14: 'MR. MUGS IS KIDNAPPED'.
MM CR COMP.40	D	PJ	202	16k	Mr. Mugs: Identification of question types (who, what, when, where). Refer L6 P101: 'MR. MUGS IS KIDNAPPED'.
MM CRL 1.40	D	P	202	16k	Mr. Mugs: Reading comprehension based on current story. Refer L6 P5: 'MR. MUGS IS KIDNAPPED'.
MM DARK WOOD.40	D	PJ	222	16k	Mr. Mugs: Vocabulary drill. Refer L4 P281: 'IN A DARK WOOD'.
MM HOMONYMS.40	D	PJ	202	16k	Mr. Mugs: Choosing the correct word of two that sound the same. Refer L6 P202: 'IT'S SATURDAY'.
MM LAD VF.40	D	P	202	16k	Mr. Mugs: Drilling students on applying verb forms. Refer L3 P348: 'MR. MUGS IS LOST'.
MM SAD STORY.40	D	P	202	16k	Mr. Mugs: Student selects ending to complete sentence. Refer L3 P333: 'MR. MUGS IS LOST'.
MM SHARE TIME.40	D	PJ	202	16k	Mr. Mugs: Vocabulary/comprehension drill. Refer L4 P39: 'SHARING TIME'.
MM VB FORMS 1.40	D	P	202	16k	Mr. Mugs: Drilling students on applying verb forms (fill-in-the-blanks). Refer L5 P95: 'MR. MUGS AT SCHOOL'.
MM VB FORMS 3.40	D	P	222	16k	Mr. Mugs: Drill on correct verb forms from Series 3. Refer L5 P61: 'MR. MUGS AT SCHOOL'.
MM VB FORMS 4.40	D	P	222	16k	Mr. Mugs: Drills students on verb forms of the Fourth Series. Refer L5 P191: 'IN THE RAIN'.

(E)EE - English

Name of Program	Cat	Grade	PST	Mem	Description
MM VB FORMS 5.40	D	P	222	16k	Mr. Mugs: Drills students on verb forms from Series 5. Refer L5 P203: 'IN THE RAIN'.
MM VB FORMS 6.40	D	P	222	16k	Mr. Mugs: Drills students on verb forms from the 6th Series. Refer L5 P230: 'IN THE RAIN'.
MM VB FORMS 7.40	D	P	222	16k	Mr. Mugs: Drills students on verb forms to be found in Series 7. Refer L5 P256: 'MR. MUGS TO THE RESCUE'.
MM VB FORMS 8.40	D	P	222	16k	Drills students on verb forms to be found in the 'Mr. Mugs' text, 8th Series.
MM VB FORMS 9.40	D	P	222	16k	Drills students on verb forms found in the 9th Series of the 'Mr. Mugs' book.
MM VERB FORMS.40	D	PJ	202	16k	Student selects the correct conjugation of a verb to complete sentence; 3 tenses of the verb are offered.
MM WORD 2.40	D	P	202	16k	Mister Mugs: Drilling students on sentence completion.
MM WORD MEANS.40	D	P	202	16k	Mr. Mugs: Drill on word meanings. Refer L3 P328: 'MR. MUGS IS LOST'.
MM WORDS 1.40	D	P	202	16k	Mr. Mugs: Sentence completion. Student fills the blank in a sentence by choosing a word from the list provided.
NEW TACHISTO.40	G	JISC	922	16k	Words are flashed on screen and student must enter each one; speed of flash increases as player improves.
NOT SO EASY.40	G		400	16k	User attempts to find the 'secret' underlying a mysterious paragraph.
NOUNS.40	DT	JI	222	16k	A tutorial and drill on nouns which provides definitions, questions and multiple-choice testing.
OLD PROVERBS.40	U	JIS	000	16k	Provides the student with a number of 'old proverbs' to read.
P'BLEM P'NOUN.40	D	I	204	16k	A multiple-choice drill of variable difficulty which gives the student practice in pronoun usage.
PARTS SPEECH.40	D	I	202	16k	Student determines whether a word is an adjective, verb, noun or preposition in this multiple-choice drill.
PAWS.40	D	PJ	412	16k	User corrects unpunctuated quotes selected by Toby the Tiger from the Jungle of Punctuation.
PETPITPATPOT.40	DG	I	402	16k	Student must guess 10 words having the prefixes 'PET', 'PIT', 'PAT' or 'POT'.

(E)EF - English

Name of Program	Cat	Grade	PST	Mem	Description
PLURALS.40	D	P	602	32k	Drills student on the basic plural forms. Excellent graphics encourage and reward answers to the 42 questions.
POEMS.40	G	JI	000	16k	A random poetry generator which can be easily altered.
POET.40	U	I	000	16k	Computer randomly arranges lines of poetry. The probability of line repitition, etc. is shown.
POETRY.40	T	JI	400	32k	Allows user to write simple poems using either 'is like' or 'I used to/ but now' constructions. User-friendly.
Q'S AND Z'S.40	D	J	400	16k	The student tries to find words beginning with the letters 'Q' or 'Z' which match given meanings.
READ LEV&EVAL.40	TU	PJISC	462	16k	This program enables the teacher to analyse the student's reading level.
READER.40	D	PJISC	100	32k	The program is designed to improve reading speed and comprehension. Teacher may modify text to required level.
RHYMING.40	D	P	202	32k	Program is a test, with graphic rewards, of user's rhyme-identification skills. Keeps score and elapsed time.
ROMEO&JULIET.40	D	SI	312	32k	A quiz on Shakespeare's 'ROMEO AND JULIET'.
S-SPELL.40	DG	J	202	16k	Student finds hidden word by filling in letter blanks; 20 letter guesses allowed; point bonus for getting word.
SCRAMBLE 4.40	DG	J	402	16k	Student must unscramble 10 randomly-chosen words, Grade 4 level. The first letter is given.
SCRAMBLE 5.40	DG	J	402	16k	Student must unscramble 10 randomly-chosen words, Grade 5 level. The first letter is given.
SCRAMBLE 6.40	G	J	402	16k	A well-designed word puzzle game, for Grade 6, which asks student to decipher a scrambled word (no time limit).
SCRAMBLE 7.40	G	I	402	16k	A well-designed word puzzle game, for Grade 7, which asks student to decipher a scrambled word (no time limit).
SCRAMBLE 8.40	G	I	402	16k	A well-designed word puzzle game, for Grade 8, which asks student to decipher a scrambled word (no time limit).
SCRAMBLE.40	G	P	400	16k	Student types in lines and they appear scrambled on the screen; letters then creep 'home' to re-form sentences.
SCRAMBLEDWORD.40	G	I	402	16k	Student is called upon to correctly unscramble various words.

(E)EG - English

Name of Program	Cat	Grade	PST	Mem	Description
SENT ANALYSIS.40	D		400	16k	Student is called upon to input various parts of speech into a given sentence.
SHAKESPEARE.40	D	IS	462	16k	Consists of 'Who am I' and 'Who said' questions about Shakespearean dramas. Requires some study/preparation.
SNERD.40	T	PJ	400	16k	A writing program which encourages student creativity and use of descriptive words.
SNOWYDAYNOUNS.40	DG	P	400	16k	This program challenges the student to find all the hidden nouns in a picture.
SP'G ERRORS 5.40	D	I	402	16k	Student identifies and corrects various misspelled words.
SP'G ERRORS 6.40	D	I	402	16k	Student identifies and corrects various misspelled words.
SPEED READ 2.40	D	PJIS	300	32k	A short phrase is flashed briefly on screen; student must repeat it. Variable levels of speed and difficulty.
SPEED READING.40	D	PJI	494	16k	Program flashes 1-9 digits which the user must recall. Duration of flash is variable.
SPEED SPELL 2.40	DG	P	422	32k	A word flashes on screen and the student types it. Flash speed is determined by spelling accuracy (Grade 2).
SPEED SPELL 3.40	DG	J	422	32k	A speed-spelling drill for Grade 3 (see 'SPEED SPELL 2.40'). Has 358 words.
SPEED SPELL 4.40	DG	J	422	32k	A speed-spelling drill for Grade 4 (see 'SPEED SPELL 2.40'). Has 483 words.
SPEED SPELL 5.40	DG	J	422	32k	A speed-spelling drill for Grade 5 (see 'SPEED SPELL 2.40'). Has 483 words.
SPEED SPELL 6.40	DG	J	422	32k	A speed-spelling drill for Grade 6 (see 'SPEED SPELL 2.40'). Has 450 words.
SPEED SPELL 7.40	DG	I	422	32k	A speed-spelling drill for Grade 7 (see 'SPEED SPELL 2.40'). Has 447 words.

(E)EH - English

Name of Program	Cat	Grade	PST	Mem	Description
SPEED SPELL.40	D			16k	A word-flash spelling drill.
SPELL BEE.40	D	PJI	410	16k	A word is flashed on the screen and the student must type it correctly; 6 levels of difficulty, modifiable.
SPELLER.40	D	J	402	16k	This program is a quiz on the meanings of 20 words; data may be modified to suit any grade level.
SPELLING 1.40	D	P	402	16k	A word is flashed on the screen and the student is asked to re-type it exactly.
SPELLING 2.40	D	P	402	16k	Computer scrambles various words entered by the student, who must then spell the words correctly.
SPELLING.40	D	PJ	202	16k	Student responds with 'Y' or 'N' depending on whether a given word is spelled correctly or incorrectly.
SPELLINGTUTOR.40	P	PJ	401	16k	A teacher inputs up to 50 words. The student must unscramble these words and also correct those misspelled.
STORY WRITER.V11	U	P	100	16k	A very popular simple word processor designed for primary students. Prints out in enhanced print if desired.
SWAP OLD ROM.40	D	J	422	32k	Student must swap word positions on a list until it has been put in alphabetical order.
SWAP.40	GT	J	420	16k	Specific words entered by the student are moved into alphabetical order to sound accompaniment.

CONT. (E)EH - ENGLISH....

SYLLABLES.40	D	J	602	32k	Student has to state the number of syllables in a word, then divide the word in the appropriate places.
TACHISTISCOPE.40	GD	PJ	551	16k	Increases reading speed by flashing short phrases on the screen; duration of flash varies with user's accuracy.
THEIR THERE.40	DT	J	402	16k	Presents a review and drill on the meanings and uses of 'THEIR', 'THERE' and 'THEY'RE'; 25 questions in all.
THEWORDMARKET.40	G	J	402	16k	Student must correctly spell a given word in order to 'purchase' word from the 'Word Market'.
TWENTY QUESTN.40	G	J	400	16k	Student selects a category; the computer poses questions entered beforehand by the teacher.
TWO TO TOO.40	DT	JI	402	16k	Teaches student the correct uses of 'TO', 'TOO' and 'TWO'.
UNSCRAMBLE.40	DG	I	402	16k	The student is required to unscramble words of various types.

(E)EI - English

Name of Program	Cat	Grade	PST	Mem	Description
VERB CHOICE.40	D	P	302	16k	Student fills in the blank with the correct one of three verb tenses presented; there are 10 questions.
VOCABULARY3.40	D	P	312	32k	A Grade 3 vocabulary test in multiple-choice form.
VOCABULARY4.40	D	J	202	16k	A Grade 4 vocabulary test which focuses on synonyms.
VOWEL MAGIC.40	D	PJ	260	16k	Student enters any word and is quizzed by the computer on the number of vowels the word contains.
WORD DEMO.40	O	JIS	400	16k	User enters up to 10 phrases and the computer rearranges them in different orders; no instructions.
WORD DRILL.40	DT	JI	202	32k	Student selects one of two homonyms to complete a sentence; if answer is incorrect, definitions are displayed.
WORD HUNT.40	D	JI	400	16k	The program gives clues in 'wanted poster' format. Student must identify the fugitive word.
WORD INVADERS.40	G	PJ	700	16k	The student must shoot the empty space in a moving word, then guess the missing letter that completes the word.
WORD MACHINE.40	DG	EP	302	16k	User must select 5 correctly spelled words from various 3-letter combinations that move across the screen.
WORD POWER.40	U	ISC	402	16k	Student or teacher can write word files or prepare a test on definitions. A good test-writing program.
WORD POWER2.40	T	JI	202	16k	Student selects proper definition of given word; adapts to all levels. For sample data load 'WORDPOWERSAMPLE'.
WORD QUEST.40	G	I	424	16k	Program creates a word-search puzzle. Student must locate the hidden words within a group of random letters.
WORD SEARCH.40	GU	JISC	000	16k	Program creates a search game by hiding user-selected words inside a crossword puzzle; print-out if desired.
WORDSHOOT.40	G	PJI	200	16k	Student 'shoots down' misspelled words and must spell them correctly afterwards.

(E)FA - French

Name of Program	Cat	Grade	PST	Mem	Description
FRENCH DRILL.40	D	JI	432	16k	A thorough drill in simple French vocabulary.
FRENCH NUMS.40	D	P	102	16k	A simple drill on French numbers.
INTERET CMPSE.40	DT	SC	311	16k	Questions relating to compound/accrued interest, capital and percentage. Interest table + calculator required.
LE PENDU.40	DG	JI	332	32k	A French version of 'HANGMAN' which utilizes common words and offers clues.
LES FRACTIONS.40	D	I	411	16k	Drills +, -, * and / with fractions; correct answers provided.
MATRICES MATH.40	DT	S	412	16k	Gives 10 examples on how to solve matrices. Allows student to input answers, then gives solutions.
PROGRES. GEOM.40	D	IS	202	16k	Drills student on problems of geometrical progression.
RACINE CARREE.40	D	I	202	16k	Student must calculate the square root of a given number.
REVUE PASSE.40	D	JI	220	16k	A good review of passe compose (requires instruction in passe compose beforehand).
SERIE 1.40	DT	I	000	16k	This program teaches student how to recognize patterns in a series.
VERBES.40	D	I	100	16k	Student must select the verb form (past and present are given) which correctly completes a sentence.

(E)GA - Games

Name of Program	Cat	Grade	PST	Mem	Description
ABSTRACT.40	G	JI	422	16k	A 'MASTERMIND' game with 3 numbers; a good test of logic and memory.
AFO!.40	G	PJ	000	16k	A Japanese variant of an 'invader' game. Player tries to down the 'AFO' with a laser, without being hit.
ARROW.40	G	PJI	211	16k	Player guides a 'snake' to hit target boxes while avoiding boundaries and the snake itself.
ATTRIBUTE BLK.40	G		200	16k	A logic game in which the player must find out which items belong to each card.
BATTLESHIP.40	G	*	212	16k	User plays against computer. Each has 5 ships hidden on a grid; winner is first to find and sink other fleet.

CONT. (E)GA - GAMES....

BIOCOMPAT.40	G	IS	402	16k	Calculates the compatibility of 2 persons according to their biorhythms.
BOWLING.40	S	PJ	202	16k	A bowling simulation for 1-3 players.
BREAKOUT.40	G	PJI	212	16k	Player's paddle deflects a ball into a wall of blocks until a 'break--out' is achieved.
CAREFUL.40	G	JI	000	16k	A fast action game whose objective is to avoid the perimeter and the obstacles which are continually appearing.
CASTLE QUEST.40	G		012	16k	An adventure game set in a castle.
CHASE.40	G		222	16k	A 2-person game of computer chase in which each player attempts to 'tag' the other.
CONCENTR'N 2.40	G	JIS	212	16k	A game which challenges the player to recall pairs of matching patterns.
CRAZY BALLON.40	G		112	16k	Player has 4 chances to guide a balloon through some prickly stars without hitting any of them.
CYCLON BATTLE.40	S	JI	902	16k	Player attempts to center cyclon fighters in a gunsight and shoot them down.
DRACULA.40	G	PJ	200	16k	Player searches a haunted house for Dracula's resting place, which must be found before midnight.
DRAGON ISLAND.40	G	J	000	16k	An unseen dragon is chased through its caves until either player or dragon is destroyed, or time has expired.

(E)GB - Games

Name of Program	Cat	Grade	PST	Mem	Description
DRAGON MAZE.40	G	PJ	200	16k	Object: escape an invisible maze before dragon arrives. Sections of maze walls become visible when struck.
DRAW 3.40	U	PJIS	200	16k	Permits the user to draw pictures on screen using graphic characters.
DRAW POKER.40	S	S	602	16k	Simulates a one-on-one game of 'DRAW POKER', with 5 cards and one draw of 3.
DROIDS.40	G	JI	212	16k	A game for up to 4 players. Object is to mine the most ore.
DUCK SHOOT.40	S	PJ	202	16k	The object is to hit a flying duck in the body (head and tail don't count). Bird dog retrieves downed ducks.
DUNGEON.40	G	JIS	242	32k	A adventure game of dungeon escape, with interesting creatures, a map and a key.
ELIZA.40	GS	IS	300	16k	User reveals personal problems to 'ELIZA' and receives sympathetic responses which encourage self-analysis.
FACES TO MAKE.40	G	P	100	16k	Enables user to make up a variety of faces by choosing from a collection of different noses, eyes and mouths.
FISH.40	S	JI	202	16k	Player tries to estimate number of trout in 3-20 lakes by catching, marking and returning fish.
FOOTBALL.40	S	PJI	204	16k	A simulation of American football. User has 7 plays to call on; probability of success differs with each.
FROG RACE.40	G	P	202	16k	Program allows 1-16 players to bet on a frog race. The different odds on each frog in the race are supplied.
HAMLET.40	G	JIS	422	16k	The game of 'OTHELLO' (which is a version of the Chinese 'GO') played against the computer.
HARD INVADERS.40	G	I	312	16k	A 'SPACE INVADERS' game done completely in machine language.
HURKLE.40	G		010	16k	Find the 'hurkle' hiding in a grid. One of the better grid/search games.
KENO.40	G	PJ	202	16k	A roulette-type board game. Player chooses up to 9 numbers to bet on; computer chooses 20. Match to win.

(E)GC - Games

Name of Program	Cat	Grade	PST	Mem	Description
KINGDOM.40	S	JI	332	32k	Player governs an agrarian kingdom, making decisions concerning food production, land purchases, etc.
LETTER 15.40	G	PJ	332	16k	A version of the logic game '15' using letters.
MARTIANS.40	G	J	222	16k	Player strives to catch the last remaining 'Martian' hiding in a grid.
MASTERMINDSP.40	G	JIS	345	32k	The game of 'SUPER MASTERMIND' with C-64 color.
MAZE.40	G	JI	100	16k	Program generates a maze (3 sizes), then times progress through it. Player may watch generation if desired.
MAZES.40	G	I	100	16k	Draws 3 different sizes of maze for player to traverse.
MILLE BORNES.40	G	JIS	232	32k	Reproduces the original card game. Player and computer vie to be the first to go 1000 miles by 'automobile'.
MIMIC.40	G	PJIS	952	32k	The game of 'SIMON' with music and graphics, offering 5 levels of play on a 3x3 grid.
MOUSE MAZE.40	G	IS	602	16k	Player/'mouse' must negotiate a maze in order to reach some cheese waiting at the exit.
NERVES.40	G	PJ	252	16k	A simple game that tests the user's ability to judge short intervals of time; 3 play levels.
NIM.40	G	PJI	240	16k	Player competes with the computer to be the last one to remove an object from 3 piles.
NUMBER-TOE.40	DG	PJ	222	16k	A version of 'TIC-TAC-TOE' in which player must make the first two numbers in a row add up to the third number.
OSCAR LUNAR.40	S		222	16k	A lunar-lander simulation in which all relevant information is updated on the screen during the descent.
PETALS-ROSE.40	G	JIS	302	16k	A puzzle involving the scoring of 5 dice. Can you figure out how the total score is calculated?
PETMAN 2.40	G	JI	200	16k	A 'PAC-MAN' game for the PET or C-64. One screen with 3 levels of difficulty.

(E)GD - Games

Name of Program	Cat	Grade	PST	Mem	Description
PINBALL.40	S	JI	111	16k	A simulation of a 'PINBALL' game.
PLANET PROBE.40	S	PJ	201	16k	Player must apply correct thrust against gravity to make a close pass over a planet. Gravity is variable.
PONG.40	G		201	16k	A game in which a bouncing ball is deflected to hit a target.
POSTAL ROUTE.40	G	JIS	122	16k	Player attempts to drive through a postal route without using the same path twice.
RAGING ROBOTS.40	G	JI	202	16k	Player tries to survive inside an enclosure by destroying 'raging robots' that are programmed to attack.
ROAD RALLY.40	G	JI	112	16k	Player attempts to drive a car five kilometers on one half-litre of gas.
ROAD TRACK.40	S		111	16k	A solo game in which player tries to move a 'car' around a track without hitting any walls.
ROADRACE.40	S	J	200	16k	Player attempts to negotiate track without colliding with the walls; speed is variable.
ROCKETS!.40	G	P	232	16k	A 1-player game in which the object is to guide a rocket through stars, etc. to the top of the screen.
ROTATE.40	G	JIS	200	16k	Player attempts to arrange letters in alphabetical order within a square by rotating 4 at a time, clockwise.
SHOOT.40	G	JI	000	16k	Two lines shoot across the screen; player presses 'F' and tries to hit the top line.
SNAKES.40	G	JIS	312	16k	Player tries to entrap opponent by creating a maze. 3 modes: comp. vs comp., player vs comp., or 2 players.
SPACE ATTACK.40	G	PJI	112	16k	A variation on the 'SPACE INVADERS' theme.
SPACE PILOT.40	S		212	16k	Player attempts to destroy arms warehouses by means of aerial bombardment.
SPADES.40	G	JISC	202	16k	A card game for 1 player. User must know rules of 4-handed spades to play; instructions included.
STAR TREK IV.40	S		222	32k	A passable 'STAR TREK' game which lacks the 'galaxy scan' feature of the Butterfield version.
STAR TREK.40	S	JIS	332	16k	A classic 'STAR TREK' game which puts player in command of a well-armed starship with a mission to fulfill.

(E)GE - Games

Name of Program	Cat	Grade	PST	Mem	Description
SUPERDRAW!.40	GU	PJ	200	16k	A drawing program in which user is able to change the character under the cursor.
TIC-TAC-PRO.40	G		222	16k	User plays 'TIC-TAC-TOE' with the computer, which 'learns' from its mistakes after a number of games.
TICTACTOE 2.40	G	PJ	221	16k	A variation of the game 'TIC-TAC-TOE'.
TORCH MAN.40	G	I	122	16k	Player must find Lois Lane in a 100 x 100 x 100 palace and rescue her from the evil 'Torch Man'.
TORP. BOMBER.40	S	PJI	902	16k	Player, as the pilot of a torpedo bomber, attempts to hit submarines lurking beneath the ocean surface.
WAREHOUSE.40	S	IS	552	32k	Simulates a day at a warehouse experiencing various levels of activity. User co-ordinates orders and shipments.

(E)RA - Geography

Name of Program	Cat	Grade	PST	Mem	Description
AFRICA & ASIA.40	D	I	202	16k	Quizzes students on their knowledge of Asian and African capitals.
ANCHORAGE-2.40	S	IS	300	32k	Simulates the navigation of a boat to a safe anchorage. Student has to take bearings, set course and speed.
CAN GEOG QUIZ.40	D	I	201	16k	Quizzes the student on miscellaneous Canadian geography facts.
CANADA QUIZ.40	D	JIS	302	16k	Asks 10 questions concerning Canada's political personalities, provinces and capitals.
CANADA.40		I	212	32k	A quiz on parts of Canada displayed on a map. Student must identify provinces, then name and locate capitals.
CAP CITIES.40	D	I	211	32k	Quizzes student on the capitals of world regions; choice of region is left to the student.
CAPITALS.40	D	I	211	32k	Allows student the choice of being drilled on Canadian, American or world capitals.
CITY SMOG.40	S	JIS	200	16k	Uses graphs to simulate various conditions that affect smog. The student inputs all data.
CLIM GRAPHS.40	O	JIS	202	16k	Tests a student's skill at reading and analyzing climate graphs.
FOREST FIRE.40	S	I	202	16k	Simulates a forest fire whose outcome depends on student's decisions.
GEO QUIZ.40	D	J	202	16k	A multiple-choice Canadian geography quiz.
GEOG. TEST.40	D	I	202	16k	A quiz on miscellaneous facts of American geography.
GEOGRAPH QUIZ.40	D	I	302	16k	A general quiz on Canadian geography.
GEOGRAPH.40	D	I	201	16k	A quiz on general geographic facts - exports, capitals, etc.

(E)RB - Geography

Name of Program	Cat	Grade	PST	Mem	Description
GEOGRAPHY.40	D	I	202	16k	A general quiz of miscellaneous geographical knowledge.
HAMMURABI.40	S	I	300	16k	Player 'rules' a country, making economic decisions concerning land management, distribution of food, etc.
HISTOGRAM.40	U	S	000	16k	Groups data into a histogram.
ICE.40	ST	S	000	32k	A good graphic simulation of glacier formation and behaviour.
ITALIAN QUIZ.40	D	I	201	16k	A general quiz on Italian geographic facts and aspects of Italian life (in English).
KOPPEN.40	D	IS	322	16k	Asks 10 questions on classifying weather patterns (temperature, precipitation, etc.) under the Koppen System.
LAKE DISTRICT.40	DG	JIS		16k	A 'HANGMAN'-type game, testing knowledge of mountains, towns and waterfalls in the Lake District of England.
LIMITS.40	S	IS	200	16k	Determines population-related growth rates. Factors include birthrate, deathrate, food, pollution, etc.
MALI LIFE.40	S	I	202	16k	Simulates the decision-making process faced by a Mali tribesman selling cattle to support his family.
MAP DIRECT.40	D	J	302	16k	Tests the student's ability to find directions using a compass.
MILEAGE.40	U	IS	200	16k	Student keys in latitude/longitude of 2 or more places; computer returns distance between them in miles/km.
NORTH EAST.40	DG	JIS	202	16k	A 'HANGMAN'-type game, testing knowledge of rivers, towns, landmarks, etc. in north-eastern England.
OCEAN QUIZ.40	D	I	202	32k	Tests student's knowledge of ocean geography.
OPEN PIT MINE.40	GS	S	990	16k	A game simulating some of the hazards of open pit mining.
POP DYNAMICS.40	GS	JIS	200	16k	Simulates an ecological system involving rabbits, hawks and wolves. User manipulates various key factors.

(E)RC - Geography

Name of Program	Cat	Grade	PST	Mem	Description
POP LIMITS.40	S	S			Simulates population-related growth rates using student/standard sets for births, deaths, indust. output, etc.
POPULATION.40	S	JIS	200	16k	Draws a graph of population distribution by age from set data, or data input by the student.
STATES & CAP.40	D	I	312	16k	Tests user's knowledge of American states and capitals. Offers option of fill-in-the-blanks or multiple choice.
STATES & REG.40	D	I	200	16k	Quizzes the student about which region a particular state is found in.
SYMBOLOGY.40	D	JIS	202	16k	Students have to read a map, identifying symbols used on it.
U.S. POP.40	S	IS	320	16k	Student adjusts factors such as birth and mortality rates in order to alter human population and distribution.
VOLCANO SIMU.40	GS	JIS	220	32k	Player attempts to escape a volcanic eruption by making rational decisions as to a course of action.
WEATHERMAN.40	U	I	300	16k	Permits conversion between temperature scales; computes wind chill factor and humidity index.
WORLD ATLAS.40	T	I	000	16k	Graphically depicts maps of various countries on the screen.
WORLD CAPS.40	D	I	212	16k	Quizzes students on their knowledge of world capitals.

(E)JA - Language

Name of Program	Cat	Grade	PST	Mem	Description
FILIPINO.40	D	J	201	16k	A basic drill on common Filipino words. Simple format: a word is presented and user enters the translation.
FINGER SPELL.40	DT	J	100	32k	Uses graphics to teach the hand symbols for letters, and tests the student's recognition of these symbols.
LATIN 123.40	D	S	210	32k	Drills student on translation from English to Latin, or vice versa. Gives levels of difficulty & hints.
LATIN VOCAB.40	D	I	101	16k	Drills translation of simple English words into Latin. Displays words missed on first try at the end.
SWEDISH QUIZ.40	D	I	101	16k	Presents the student with English words to be translated into Swedish; no levels of difficulty provided.

(E)LA - Logic and Problem Solving

Name of Program	Cat	Grade	PST	Mem	Description
A V OR M.40	G	PJ	202	16k	Program gives the name of an object and student decides whether it is animal, vegetable or mineral.
ANDROID NIM.40	G	JI	212	16k	Player and computer take turns eliminating androids. The one who eliminates the last android wins the game.
BOTTLECAPS.40	G	JI	102	16k	Player and computer take turns removing bottlecaps; the one to take the last bottlecap loses the game.
BUTCH & SLIM.40	G		322	32k	Given certain facts about a robbery, the student uses logic to answer relevant questions.
CHESS BOARD.40	G	JIS	322	32k	A computer game of chess for two players.

CONT. (E)LA - LOGIC AND PROBLEM SOLVING....

CONCENTRATION.40	G	PJI	312	16k	Player must remember an assortment of patterns and match them up once they have been concealed.
CRAPS.40	S	JIS	202	16k	A simulation of the dice-rolling game called 'CRAPS'.
CRYPTOGRAM.40	G	IS	492	16k	Student tries to decipher a message by solving the encoding method.
CUBE.40	G	JIS	232	32k	The computer scrambles a Rubik's cube, and user attempts to solve it.
DRAW CAVE.40	G	JI	312	16k	An adventure game set in a maze. Player has to find treasures before starving, dying of thirst or being eaten!
ENERGY.40	G		332	16k	Requires the student to use logic to reason out various thermometer settings, given a number of clues.
FLIGHT SIMUL.40	S	JIS	432	16k	The computer simulates the flying of a small plane.
FUR TRADE.40	GS	JIS	312	16k	A simulation of fur trading in Early Canada. Player sells furs while trying to avoid disasters, ambushes, etc.
GUNNER.40	GS		222	16k	User tries to hit a target by providing the correct angle of fire for a cannon.
HI-Q.40	GS	JIS	502	16k	A simulation of the game 'HI-Q'. Object is to remove as many pegs as possible by jumping into empty holes.
IN-ORDER.40	G	JI	422	16k	Computer 'thinks' of a 3-digit number and the player tries to guess it with the aid of clues.

(E)LB - Logic and Problem Solving

Name of Program	Cat	Grade	PST	Mem	Description
IQ-TEST.40	DG	JISC	219	16k	Asks 20 mathematical sequence questions on each run and gives their solutions.
KALAH.40	G		332	16k	The ancient Egyptian 'pit-and-pebble' game. Player distributes pebbles so as to take over an opponent's pits.
KNIGHT'S TOUR.40	G	JISC	100	16k	A chess game which uses Warndorf's rule. Computer moves a 'knight' to every position on the chessboard.
LABYRINTH.40	G	JI	231	32k	The object, as the title suggests, is to find one's way through a maze.
LOGIBLOCK.40	G	I	422	16k	Player attempts to guess the two attributes of a block that the computer is 'thinking' of.
MAGIC SQUARE.40	G	PJIS	722	16k	A fascinating, frustrating logic puzzle.
MASTERMIND 2.40	G	JIS	432	16k	A computer version of 'MASTERMIND', involving the breaking of a code through use of logic.
MASTERMIND 3.40	G	JIS	432	16k	Computer version of the logic game 'MASTERMIND'. Player tries to break a 5-color code; variable difficulty.
MATCHES.40	G	JI	322	16k	A 'NIM'-type game played against the computer. Object is either to take the last match, or not to take it.
MAZE GENERAT.40	GU		200	16k	Generates mazes and draws them out on a printer.
MUGWUMPS.40	G	JI	322	16k	Object: find 4 hidden 'mugwumps' on a co-ordinate grid in 10 moves. Computer advises on proximity of targets.
OBJECT.40	DG	PJ	622	16k	A program for testing pupils' ability to distinguish between various shapes in groups.
OSERO.40	G	JIS	222	16k	The game of 'OTHELLO' played against the computer.
OTHELLO 2.40	G	JIS	200	16k	A computer version of the popular game of logic and capture. Opponent's pieces are taken by enclosing them.
OTHELLO FOR 2.40	G	JIS	212	16k	A 2-player game whose object is to capture an opponent's tokens by enclosing them.

(E)LC - Logic and Problem Solving

Name of Program	Cat	Grade	PST	Mem	Description
OTHELLO.40	G		232	16k	In this version of 'OTHELLO', player and computer attempt to capture each other's tokens by enclosing them.
PUZZLE.40	GU	JIS	000	32k	Allows student to design and solve crossword puzzles. Sample data may be obtained by loading 'PUZZLE.DATA'.
QUEST 3.40	G	JI		16k	An adventure game in which the player searches for treasure in a pirates' cave.
REMEMBERING.40	G		222	16k	Student, playing against the computer, tries to match hidden objects.
REVERSE.40	G	JI	202	16k	Player attempts to put 9 numbers in numerical order by reversing the first 'N' numbers.
RHYMECONC.40	G	JI	210	16k	A version of 'CONCENTRATION' with 2 players, using hidden rhyming words instead of cards or objects.
SEVEN GABLES.40	G	ISC	442	32k	An adventure game which traps the player in an old house containing numerous treasures.
SLOT MACHINE.40	S	JIS	402	16k	Offers a graphic simulation of playing a slot machine.
SNARK.40	G	J	322	16k	Player finds a 'snark' on a grid by entering the center and radius of a circle in which it might be hidden.
SOLITAIRE.40	S	JIS	732	16k	Lets user play all 3 versions of 'SOLITAIRE' on the computer.
TIC TAC TOE.40	G	PJ	202	16k	Student plays 'TIC-TAC-TOE' against the computer.
TOWER HANOI.40	G	JIS	332	16k	Move a pile of different sized blocks from one peg to another, without putting large blocks on smaller ones.
TRACE-A-WORD.40	G	PJI	402	16k	Student tries to find hidden words within a time limit.
TWENTY QUEST.40	G	PJ	592	16k	Computer acts as an 'artificial intelligence', asking questions to increase its knowledge in various areas.

(E)LD - Logic and Problem Solving

Name of Program	Cat	Grade	PST	Mem	Description
US CIVIL WAR.40	S	JIS	322	16k	A Civil War simulation. Object is to win as many battles as possible. Facts and figures have historical basis.
WATCHPERSON.40	G	JIS	221	16k	Player tries to find a way to walk through town without retracing steps.
WEIGH.40	S	JI	000	16k	User must find the odd weight, and determine if it is lighter or heavier, given only 3 chances at the scales.
WESTWARD HO.40	GS	JI	442	32k	An entertaining adventure program which simulates life in the Wild West at the time of the Gold Rush.
WUMPUS.40	G	JI	332	16k	An adventure game in a dodecahedron. Player hunts the 'wumpus' through a series of imaginary tunnels and rooms.
YAHTZEE.40	G	JIS	322	16k	A computer version of the game of 'YAHTZEE' in which player tries to roll various combinations with 5 dice.

(E)MA - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
ADD AND SUB.40	DT	I	402	16k	Drills the student on addition and subtraction of signed numbers.
ADD DRILL.40	D	J	202	16k	Student has option of doing addition problems with 0 to 9 rows of numbers, or 0 to 9 digits in each number.
ADD TEACHER.40	DT	P	224	16k	Teaches student how to add numbers together and provides cumulative marking.
ADDING DRILL.40	D	P	202	16k	A simple drill on adding two numbers in the range 1-100.
ADDING QUIZ.40	D	P	202	16k	An adding drill which utilizes numbers with up to 4 digits and points out incorrect digits in the answers.
ADDITION GAME.40	D	P	202	16k	Student is given 10 timed addition problems.
ADDITION RACE.40	DG	J	202	16k	Addition drill game. Players advance the two men on the screen by correctly answering addition problems.
ADDITION.40	D	IS	202	16k	A drill made up of 10 random addition problems; entry of digits is left-to-right.
ADDS AND SUBS.40	D	PJ	200	16k	Drills addition or subtraction and lets student count objects if answer is incorrect; good incentive graphics.
AGENT BLOTTO.40	D	P	222	16k	This program uses math questions to solve a mystery. Each correct answer reveals a letter in a secret message.
ALG. VECTORS.40	D	S	202	16k	Drills nine sub-topics under algebraic vectors.
ALGEBRA DRILL.40	DT	I	440	16k	A drill/tutorial in simplifying algebraic expressions.
ARITH DRILL.40	D	J	202	16k	A timed drill on +, -, and *, with optional levels of difficulty.
ARITHMETIC 1.40	D	J	422	16k	A drill in +, -, and * with levels of difficulty.
ARITHMETIC.40	D	J	202	16k	Practice with simple +, -, and *.
B.T.C. ADD.40	D	J		16k	A drill in addition facts against user-set time limit.
B.T.C. DIVIDE.40	D	P	202	16k	Poses division questions which are to be answered within a time limit set by teacher or student.

(E)MB - Mathematics

Name of Program	Cat	Grade	PST	MEM	Description
B.T.C. FRACT.40	D	J	222	16k	Practice in multiplying fractions within a user-set time limit.
B.T.C. MULT.40	DG	PJ	422	16k	Multiplication questions must be answered within a time limit specified at the beginning of the game.
B.T.C. SUBTRT.40	D	P	202	16k	Subtraction facts with up to 2-digit regrouping flash against the clock. In second part, player vs computer.
BAIRSTOW NTH.40	U	SC	200	16k	Uses Bairstow's iterative method to find successive quadratic factors of an nth order polynomial.
BALANCE.40	DS	JI	422	16k	Drills student in balancing various metric weights on simulated scales.
BASIC MATH.40	D	PJ	202	16k	A drill in basic +, -, * and /.
BEADS IN JAR.40	T	JI	200	16k	Provides an illustration of probability by drawing beads from a jar at random.
BETWEEN.40	D	P	200	16k	Student attempts to guess a secret number between given limits: 0 < number < 10.
BIG BINARY.40	U	IS	200	16k	Converts decimal numbers (up to 511) into binary form.
BIG DIVIDE.40	D	J	200	16k	A drill in simple division yielding 1 to 2-digit results.
BIG MATH.40	D	P	202	16k	Responses to 5 vertically arranged math problems (choice of +, -, * or /) are keyed in from right-to-left.
BIG SUBTRACT.40	D	I	200	16k	Drills the subtraction of whole numbers using large numerals in the screen display.
BIGTIME.40	DGU	P	200	16k	Creates a large 12 or 24 hour digital clock, with alarm.
BINOM. EXPAN.40	DT	S	302	32k	A drill in expanding binomial products using 'F.O.I.L.'
BINOMIAL DRIL.40	D	IS	202	32k	A drill on expansion of binomial multiplication.
BINOMIAL EXP.40	TU	I	200	16k	Explains and calculates binomials using Pascal's triangle and large graphics.

(E)MC - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
BODMAS.40	D	J	202	16k	A drill on the order of operations.
BOMB ADDITION.40	DG	J	200	16k	A drill/game using 2-digit addition problems. If answer is incorrect, a 'bomb' explodes.
BRAIN CRANE +.40	D	J	200	16k	This program builds up student addition skills by drill method. Graphics feature a crane which moves numbers.
BRAIN CRANE /.40	D	PJ	222	16k	Uses drill method and incentive graphics to 'build up' student's division skills.
BRAIN CRANE X.40	D	PJ	222	16k	This program 'builds up' the student's multiplication skills by drill method.
CALCULUS.40	DT	S	422	16k	A drill on simple calculus problems involving acceleration and velocity.
CAR RACE MULT.40	DG	P	202	16k	Two players race their 'cars' across the screen by answering multiplication questions.
CASH REGISTER.40	D	P	202	16k	After a 'purchase', user is required to give out correct change in the smallest number of bills and coins.
CHANGEMAKER.40	ST	J	200	16k	Simulates the buying of items in a store. Computer totals prices, adds sales tax and shows how to make change.
CHOICES.40	T	I	200	16k	A study in probability. Illustrates the number of ways to select 'R' items from 'N' items.
CLOCK.40	D	P	202	32k	Computer displays a clock face and student enters appropriate digital time. Total of 10 questions.
CO-ORDINATES.40	DT	JI	922	32k	A good introductory lesson on the cartesian coordinate system.
COIN PUZZLE.40	DG	JI	200	16k	Student must weigh coins on a balance in order to discover odd coin.
COLLECT TERMS.40	D	JI	200	16k	A drill in collecting like terms in algebraic expressions; three levels of difficulty.
COLLECTERMS 1.40	D	I	202	16k	A drill in collecting coefficients of like algebraic terms.

(E)MD - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
COLLECTERM 2.40	D	I	202	16k	Gives practice collecting coefficients of like algebraic terms.
COPY CAT.40	D	JI	200	16k	Drills student in remembering numbers and letters.
COUNT 1 to 10.40	T	P	200	16k	Teaches student counting from 1-10.
COUNT FIVE.40	T	P	000	16k	This program uses graphics to aid the student in learning how to count to ten.
COUNT TEN.40	D	P	200	16k	This program uses graphics to drill student in counting from 1-10.
COUNTING.40	D	EP	202	16k	This program helps youngsters learn counting by asking them how many objects are on the screen.
CURVE FIT.40	TU	SC	500	32k	Teaches evaluation of a polynomial to fit a set of points, integration and plotting included.
DARTS.40	DGS	JI	202	16k	Student answers problems in +, -, * and / in order to score points on a dart board; good range of difficulty.
DECIMAL ARITH.40	D	JI	402	16k	Computer generates random decimals for a quiz made up of 4 math problems (+, -, * and /).
DECIMAL SIZE.40	D	J	202	16k	The student selects the largest of three numbers with identical digits, but with different decimal position.
DECOMPOSE.40	DT	I	201	16k	This program teaches and drills the factoring of trinomial equations using the method of decomposition.
DERIV OF POLY.40	U	S	200	16k	This program finds the derivative of polynomials entered by the user.
DICE THROW.40	S	IS	000	16k	Demonstrates distribution of dice-sum frequencies using variable no. of dice and no. of sides to the dice.
DIV DRILL.40	D	J	202	16k	A drill in basic division with divisors from 1-10.
DRILL.40	D	P	202	16k	Drills addition, subtraction (to 20), division and multiplication (to 9 times table).
DRILLS.40	D	PJ	202	16k	Provides practice in +, -, * and /.
ELLIPSE TRANS.40	U	S	200	16k	Student inputs the variables (values less than 12) for computer-drawn elipses and transformations.

(E)ME - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
ENGGAME TWO.40	D	J	200	16k	User performs 4 operations on 5 numbers to solve a mathematical puzzle.
EQU'N MANIP.40	DT	JI	202	16k	Drills student on problems involving the isolation of a single variable in a simple equation; poor explanation.
EQUATION EXA.40	DGS	I	222	16k	Student finds how many marbles are in a bag by balancing bags against loose marbles on a simulated scale.
EQUATION X-Y.40	U	I	200	16k	User inputs A, B and C for linear equations and the program graphs the resulting line.
EQUATION.40	DT	I	202	16k	Student solves a linear equation in 1 unknown. Computer shows solution if requested.
EQUATIONS.40	D	I	200	16k	Emulates equation-solving procedure by asking student to find number of marbles in each bag on a balance scale.
EXPONENT MULT.40E	D	I	220	16k	A program which drills a student in simple algebraic multiplication involving monomials.
EXPONENTS.40	DT	I	202	16k	This program instructs and drills the student in multiplication and division of exponents.

CONT. (E)HE - MATHEMATICS....

FACTOR DRILL.40	D	IS	202	16k A drill on factoring polynomial equations to the 6th degree. Requires 32k memory for higher option levels.
FACTOR TRI.40	D	I	200	16k Provides practice in solving quadratic equations.
FACTOR TRINO.40	D	IS	202	16k A drill on factoring trinomials into linear equations.
FACTOR WHOLE.40	D	JI	200	16k Student must break various numbers down into their prime factors.
FACTORIAL.40	U	IS	200	16k Provides answers to factorial up to 500.
FACTORS.40	U	JIS	200	16k This program calculates the prime factors of whole numbers input by the user.
FAST MATH.40	DG	J	202	16k Players compete in a game/drill whose object is to answer addition problems as quickly as possible.
FC'N GRAPH.40	U	I	200	16k A good graphing utility. Self-modifying; inserts user-input functions into line 1025.
FLASHCARDS.40	D	J	202	16k Drills students on +, -, and *.

(E)MF - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
FRAC EST/SOUN.40F	DG	J	822	16k	A fraction estimation game in which the student must guess the correct fractional distance to a target.
FRACTION GAME.40	DG	J	000	16k	A target appears on a number line from 0 to 2; user must guess the fractional value the target represents.
FRACTION PRAC.40	D	J	202	16k	Drills the user on conversion between decimals and fractions.
FUNC MACHINE.40	DGS	JI	500	32k	A simulated machine cranks out a number; student must guess the secret operation that's been performed on it.
FUNC PLOT.40	T	S	200	16k	Student can request examples of functions (circle, parabola, ellipse, etc.) and change the defining equations.
GAUSS REDUCT.40	U	S	200	16k	Student enters the coefficients of a system of linear equations and the computer calculates the answer.
GEOMETRY.40	D	JI	602	16k	A geometric shape recognition drill in which the student must name various polygons.
GEOMETRYTERMS.40	DT	JI	202	16k	Explains the geometric terms angle, point, line, line segment and ray and presents a quiz afterwards.
GRAPH PLOT.40	U	I	200	16k	Plots the graph of a user-defined function.
GRAPH PRINTER.40	U	JIS	200	16k	Program draws a graph according to user-specified parameters.
GRAPH SNAP.40	S	IS	300	16k	Graphs any equation. Allows the user to move a window around on the graph and to change the window's specs.
GRAPH.40	T	I	200	16k	User inputs co-ordinate and the computer plots it OR computer plots co-ordinate and the user names it.
GRAPHING.40	T	I	200	16k	Illustrates and compares 3 methods of graphing points.
HI-CALC.40	D	IS	200	16k	Student must use calculus to maximize an algebraic equation.
HI-LOW.40	G	PJ	200	16k	Computer guesses a number between 1 and 1,000,000 in less than 20 guesses.
HYPBOL TRANS.40	U	S	200	16k	From coefficients input by the student, a graph is plotted in standard position and with transformation.
IN-BETWEEN.40	G	JI	101	16k	Player bets on whether or not a third card's value is going to fall between that of two cards dealt face-up.
INDIRECT EVID.40	D	I	202	16k	Gives student practice in formulating hypotheses and drawing conclusions.

(E)MG - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
INT.ADD.FAST.40	D	JOI	202	16k	Drills students in integer and whole number addition.
INT/EX ANGLES.40	D	JI	200	16k	Drills students on the relationships between interior and exterior angles.
INTEGER ADD.40	D	J	202	16k	Drills student on the addition of single-digit signed numbers.
INTEGER ARITH.40	D	I	202	32k	A good drill in integer addition and subtraction.
INTEGER LINES.40	U	I	400	16k	Student inputs the coefficients of two linear equations and the computer gives their point of intersection.
INTEGERS 2.40	D	JI	202	32k	Drills student in +, -, * and / of integers.
INTEGERS.40	D	J	202	16k	Program offers a series of problems in +, -, and *.
INTEGRATION.40	T	I	200	32k	Demonstrates calculation of areas under curves by summing increasingly narrower strips of rectangles.
INTERPOLATION.40	DT	S		16k	A program on interpolation and determination of a circle.
INTERSECT LIN.40	U	IS	200	16k	Finds intersection point of two lines input by user.
INTERSECT PT.40	D	I	200	16k	The student finds the point of intersection of two lines by inference from information about the points input.
INTERSECTION.40	D	I	202	16k	A drill on the angles formed by intersecting lines.
INTGR PYRMD.40	D	I	202	16k	For each correct answer to an addition or subtraction problem, another level is added to a 'pyramid'.
LADDER MULT.40	DG	J	202	16k	Drills multiplication tables. Correct answers move student up steps of a ladder.
LAZER MATH.40	DG	PJ	202	16k	Student must answer an addition problem before a laser destroys the block. Choice of number of digits (1-8).
LIMIT CIRCLE.40	TU	IS	400	16k	Calculates the limit of an equation for the area of a circle.

(E)MH - Mathmetics

Name of Program	Cat	Grade	PST	Mem	Description
LIN EQN.40	D	IS	200	16k	Drill on solving linear equations.
LINE INTERSCT.40	U	J	200	16k	User inputs the parameters for two lines and the computer returns point of intersection.
LINEAR EQUAT.40	U	I	200	16k	Draws a graph of linear line with values for A, B and C supplied by the user.
LINEAR SYS.40	D	I	202	16k	Student can choose to solve up to 4 equations and 4 unknowns at one time; correct answer follows each turn.
LONG DIVISION.40	D	JI	202	16k	Drills student in integer long division with selectable levels of difficulty.
MAKING CHANGE.40	D	P	202	16k	Student must give out correct change using the fewest bills and coins possible.
MARBLE STAT.40	S	S	500	16k	Simulation of a probability machine with marbles dropping over a matrix of pegs.
MATCH UP NUM.40	DG	P	000	16k	Student is required to determine which two numbers on the screen are the same.
MATH DICE.40	D	P	200	16k	Develops counting skills by requiring the student to total the dots which come up with each throw of 2 dice.
MATH DRILL.40	D	P	204	16k	Drills basic addition, subtraction, multiplication and division using large-sized graphics.
MATH FACTS.40	D	PJ	202	16k	Drills students in +, -, * and /.
MATH MANIA.40	DG	J	000	16k	Student moves a character about the screen until a question mark is hit; a simple arithmetic question follows.
MATH PACK.40	T	S	200	32k	This program evaluates prime factors, cubic/quadratic equations, combinations & permutations, factorials, etc.
MATH SWIM.40	DG	PJI	202	16k	Drills 2 players on +, -, *, and /. Entertaining graphics of swimmers act as an incentive.
MATH TUTOR.40	D	PJ	202	16k	Drills student in +, -, * and /.
MATRIX MATH.40	DG	JI	000	16k	Provides factoring practice within the context of a game.

(E)MI - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
MATRIX.40 I	U	S	200	16k	Adds, subtracts, multiplies and determines matrices.
MEASURE.40	DS	PJ	000	16k	Student must read a ruler measuring various objects.
MET/STD CONV.40	U	JI	200	16k	Program performs metric/standard conversions for temperature, length, weight, area and volume.
METRIC (EC00).40	D	JI	202	16k	Student is required to convert between various metric units.
METRIC CONVER.40	D	J	000	16k	Drills student in metric conversions within metric.
METRIC DRILL.40	D	PJI	200	16k	Program drills conversion of all units within metric, including volume.
METRIC DRILLS.40	D	JI	202	32k	Drills the user in metric conversions within the metric system.
METRIC M.40	D	PJI	202	16k	This program drills conversion of units of distance within metric.
METRIC TEST.40	D	JI	202	16k	Drills students in converting between various units of length within metric.
METRIC VOLUME.40	D	JI	422	16k	Practice in converting between units of volume within metric.
MICRO MATH.40	DT	I	401	32k	A drill/lesson on finding the coordinates of a point on a cartesian graph.
MICROMATH +-.40	DT	I	200	16k	Teaches and drills the addition and subtraction of integers.
MISSING NUM.40	D	EPT	200	32k	Student must identify the missing number in a series from 1-10. A happy/sad face indicates right/wrong answers.
MISSING NUMB.40	D	P	200	16k	Given a list from 1-20, student must type in the missing number.
MON.PRODUCT.40	DT	I	400	16k	Provides instruction and practice in multiplying 2 or 3 monomial factors.
MONOMIAL MULT.40	D	I	202	16k	Program allows practice in multiplying two or three monomial factors with exponents.

(E)MJ - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
MONSTER MULT.40	DG	PJ	701	32k	User builds wall to repel monster by correctly answering multiplication questions. Variable time, difficulty.
MULT BINOMIAL.40	D	IS	202	16k	Student must provide 3 coefficients corresponding to each binomial equation presented.
MULT DRILL.40	D	PJ	202	16k	A timed multiplication drill involving 2 numbers. Their values are set by student, as is no. of questions.
MULTIPLY.GS.40	DU	J	202	16k	For drilling in multiplication; saves results of test to disk or tape.
MUNCHKIN MULT.40	D	PJ	202	32k	A drill on times tables using a student-selected number from 1-99 and the numbers 1-10.
NUMBER GUESS.40	DG	EP	200	16k	Student is asked to guess a number from a number line.
NUMBER SEQ.40	D	J		16k	Student must supply the correct number sequence of 2-digit numbers.

CONT. (E)MJ - MATHEMATICS....

NUMBER TRAIN.40	D	P	502	16k Student gives the number coming before and after a given one. A graphics 'train' arrives for a correct answer.
NUMBER-TOE 2.40	G	JI	202	16k Similar to 'TIC-TAC-TOE', except that player must make the first two numbers in a row add up to the third one.
NUMBER.40	U	S	200	16k Finds the mean, standard deviation, maximum and minimum of a user-input set of numbers.
OPERATIONS.40	D	JI	222	16k Explains order of operations. Questions become harder if user gets 3 corrects in a row; 5 levels of difficulty.
ORDERED PAIRS.40	U	JI	000	16k Generates ordered pairs once given function and starting point.
PARABOLA.40	U	S	700	32k Plots and re-plots parabolas according to student-input parameters.
PARALLEL LINE.40	DT	I	202	16k A drill in the 8 angles formed by a line intersecting 2 parallel lines.
PERIMETER.40	D	J	200	16k Student must find perimeter of displayed rectangle. Dimensions are printed on 2 or 4 sides at user's request.
PERM AND COMB.40	U	IS	000	16k Program computes permutations and combinations given size of set and subset.
PERMS & COMBS.40	U	S	200	16k User inputs the variables n and r, the program computes the number ofcombinations (nCr) and permutations (nPr).
PI CALCULATOR.40	U	ISC	200	16k Calculates pi to as many decimal places as requested. Slow - calculation to 40 places takes 7 minutes.

(E)MK - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
PIZZA.40	D	I	202	16k	Student must deliver pizza to houses located on the 1st quadrant of a cartesian coordinate plane.
PLACE VALUE#3.40	G	P	200	16k	Using random digits, player and computer compete to make 2 numbers with the largest difference between them.
PLACE VALUE.40	DT	J	202	16k	Student is taught the significance of a digit's position in a decimal number and is drilled on same.
PLANES.40	D	S	402	16k	A drill in evaluating equations dealing with planes.
PLANET INTEGR.40	DGT	I	712	16k	A graphing game whose object is to reach a certain point on a grid. Teaches graphing using X-Y axis.
POLAR I.40	U	I	200	16k	Student may enter the parameters for equation R=ACOS(BX) and see function plotted.
POLICE SUBTR.40	DG	J	202	16k	Student must correctly answer subtraction problems in order to save a town from robbers.
POLY PLOT.40	TU	IS	200	16k	Plots polynomials up to degree 5.
POLYFIT.40	U	S	200	16k	Finds the polynomial of best fit for a series of data.
POLYGON SECT.40	U	IS	200	16k	Calculates centroids and moments of inertia of polygons.
POWER-FACT.40	U	S	100	16k	Computes factors or powers up to 250 digits in length.
POWRS & ROOTS.40	D	I	202	16k	A drill in squares, cubes, square roots and cube roots of small and large numbers.
PRIME # SIEVE.40	T	I	200	16k	A tutorial on finding primes using the sieve of Eratosthenes.
PRIME FACT.40	U	JIS	000	16k	This program resolves any number into its prime factors.
PRIME FACTOR.40	U	JI	200	16k	Finds the prime factors of numbers entered by the user.
PRIME NUMBERS.40	U	JI	200	16k	Finds all prime numbers up to that entered by the student.
PROBABILITY.40	S	PJ	000	16k	Illustrates the random distribution of balls cascading between obstacles on the screen.
QUAD. EQ'N.40	T	S	200	16k	Student inputs the coefficients of an equation, solves it and compares the result with the computer's answer.

(E)ML - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
QUAD.40	DT	IS	202	16k	Asks student to determine the number and form of the roots in a given quadratic equation.
QUEUE.40	S	I	000	16k	Simulates queuing at a bank with 5 tellers' windows.
QUIZ ADD.40	D	EP	202	16k	Presents the student with a series of addition problems.
QUIZ DIVISION.40	D	J	200	16k	A simple division drill.
QUIZ MULT.40	D	PJ	202	16k	A straightforward drill on a series of multiplication problems.
QUIZ SUBTRACT.40	D	J	200	16k	Provides practice in simple subtraction.
R-PLOT.40	U	IS	200	16k	Takes sets of points and picks line of best fit; also gives statistics on each point.
RANDOM.40	S	PJ	000	16k	A poor simulation of a random generator, giving frequency of numbers.
RECIPES.40	U	J	200	16k	Converts kitchen measurements from Standard to S.I. or vice versa.
REDUCE FRACT.40	D	J	202	16k	A drill in reducing fractions.
RESULTANTS.40	U	S	200	16k	Resolves user-input vectors on a cartesian or polar grid.
REVERSE 1.40	G	JI	200	16k	Student must rearrange numbers or letters so that they are in the proper order.
RND GENERATOR.40	S	JI	200	16k	Program allows user to experiment with random number generator statement.
ROLLS TIL ONE.40	S	I	200	16k	Simulates the roll of a die, keeping track of the number of rolls needed to roll a one.

CONT. (E)ML - MATHEMATICS....

ROMAN.40	D	JI	402	16k Practice in conversion of Roman numerals to decimals and vice versa.
ROOTFINDER.40	U	IS	200	16k Finds the roots of a polynomial up to the 20th degree.
ROOTS QUIZ.40	D	IS	202	16k Drills students in finding roots to trinomial equations.
S.B.MATH.40	D	J	200	16k Story-book math problems using data (animals, people and food) input by the student.
S.N.B'KETBALL.40	DG	IS	002	16k A drill in the conversion of numbers to scientific notation.

(E)MM - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
SCIENTIFIC.40	D	I	202	16k	Provides practice in scientific notation.
SHAPES.40	DT	E	200	16k	Teaches the student to distinguish between various objects.
SI.CONVERSION.40	D	JI	202	32k	Drills students in metric conversion within metric.
SIEVE.40	T	JI	200	16k	Demonstrates method of determining prime numbers by eliminating multiples of integers.
SIGNIF DIGIT.40	D	IS	002	16k	A drill on the number of significant digits in various numbers.
SIMEQ. SOLVER.40	U	IS	200	16k	Solves up to 5 unknowns with 5 equations.
SIMP.SUBST.40	D	I	220	16k	A quiz in evaluating monomial expressions.
SINE GRAPH.40	TU	IS	200	16k	Student sets the parameters for a sine curve which is then plotted by the computer.
SLOPE AND INT.40	D	I	000	16k	Drills students on line equation forms and intercepts.
SLOPE/INTRCPT.40	T	I	200	16k	A tutorial on slope, x-intercept and y-intercept of linear equations.
SMALL MATH.40	D	P	220	16k	Drills students on addition and subtraction.
SNOOPY.40	DG	PJ	202	16k	Player keys in a number corresponding to Snoopy's relative position along a line in order to hit the Red Baron.
SPLASHDOWN.40	DG	J	200	16k	Student sums divers' scores. Each correct sum gives a letter of a mystery place-name which student must guess.
STATISTICS 1.40	U	IS	200	16k	Compares sets of numbers according to PMI correlation.
STATISTICS 2.40	U	IS	200	16k	Compares sets of numbers through the coefficient of determination.
STATISTICS 3.40	U	IS	200	16k	Compares sets of numbers through the correlation coefficient.
STATISTICS.40	U	IS	200	16k	Calculates median average, frequency and standard deviation.

(E)MN - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
STORY PROB.40	D	JI	202	16k	Student enters favorite friends, foods and animals and story problems are created using this data.
SUBTRACTION.40	D	J	402	16k	Drills subtraction with 4 digits.
SURVEY.40	U	IS	200	32k	A utility for taking surveys.
SYMMETRIC.40	O	J	200	16k	Draws a symmetrical pattern on the screen.
TABLES.40	D	J	422	16k	Drills multiplication of positive and negative numbers from -100 to 100.
TIC TAC ARITH.40	DG	PJ	500	32k	Math version of 'TIC-TAC-TOE' for 2. Player must answer arithmetic problem (+, -, *, /) to occupy a space.
TIC TOC CLOCK.40	GT	P	123	16k	A game designed to teach the student how to tell time.
TIME OF DAY.40	DT	P	200	16k	Instructs and tests the student in clock reading.
TIMES TIMER.40	D	PJ	220	16k	A multiplication drill in which the student tries to answer as many questions as possible in 60 seconds.
TRANSLATION.40	S	S	200	16k	Shifts Y=X squared according to user-chosen shifts in the 'X' and 'Y' directions. Shift is animated.
TREASURE ADD.40	D	P	202	16k	Student must add numbers in order to cross a stream; too many mistakes bring a 'dunking'.
TRIGONOMETRY.40	D	IS	202	16k	Drill on sine, cosine and tangent at 30, 45 and 60 degree angles.
TRINOMIAL FAC.40	DT	I	442	16k	This program gives practice in trinomial factoring, with excellent tutorial hints if required.
TWELVE BLOCKS.40	G	IS	600	16k	Student has 3 weighings to discover which one of 12 blocks is heavier/lighter than the others. Good graphics.
UP THE LADDER.40	DG	PJ	202	16k	Problems in addition of numbers from 0 to 99. Student goes up one rung of a ladder for each correct answer.
VECTOR ALGEB.40	D	S	400	32k	Drill in cross, dot product, addition and subtraction of vectors up to 12 dimensions.

(E)MO - Mathematics

Name of Program	Cat	Grade	PST	Mem	Description
VECTOR.40	U	IS	000	32k	A good utility package for manipulating vectors.
VELOCITY PROB.40	D	IS	202	16k	A drill in problems on velocity, time and distance.
Y EQUALS MX+B.40	U	I	200	16k	This program graphs lines, given slope 'M' and Y-intercept 'B'.
ZERO IN.40	G	J	200	16k	The computer picks a number and the student attempts to guess it.
ZONE X.40	DG	P	202	16k	A plotting game. The computer draws two invisible lines on a grid; student uses clues to find intersection.

(E)NA - Music

Name of Program	Cat	Grade	PST	Mem	Description
MUSIC FILE.40	U	JIS	400	32k	This program is a music file management system.
MUSIC MACHINE.40	U	JIS	800	32k	Program enables user to play/write music on a staff, and load/save compositions. Good features and graphics.
MUSIC THEORY.40	T	J	202	16k	A basic introduction to musical notation.
PUNK ROCK.40	D	IS	402	16k	A quiz on facts concerning punk rock.
SERIALISM.40	T	S	100	32k	Demonstrates the 12 tone row, including inversion, retrograde and retrograde inversion. A good tutorial.
SOUNDS.40	T		000	16k	A demonstration of PET sound effects.

(E)PA - Physical and Health Education

Name of Program	Cat	Grade	PST	Mem	Description
CHILD ABUSE.40	D	IS	202	16k	Program asks a series of questions to test the user's awareness of child abuse, teenage pregnancy and adoption.
DRIVER ED.40	D	IS	222	32k	A drill very similar to a beginner's permit test, based on the Driver's Handbook, Ministry of Transportation.
LIFE STYLES.40	GU	IS	200	16k	Offers an assessment of lifestyle, based on user-input data regarding health, excercise, personal habits, etc.
LIFESPAN.40	GU	IS	200	16k	Given user-input data on schooling, exercise, mental state, stress, etc., program estimates life expectancy.
METEOR.40	O	JIS	000	16k	Gauges reaction time and hand/eye co-ordination. User presses a key as soon as a 'star' on the screen 'falls'.
REACT.40	O	JIS	000	16k	Tests user's reaction time (reflexes) by timing how long it takes her/him to hit the space bar after a signal.
REACTION TEST.40	D	JI	202	16k	A test of student's reaction time to a stimulus.
REFLEX TIMER.40	O	JIS	002	16k	Tests user's reflexes by measuring reaction time.
RUNNING QUIZ.40	D	JIS	202	16k	A quiz on various aspects of running and jogging.
YELLOW LIGHT.40	GS	JIS		16k	Simulates a car approaching an intersection. When light turns yellow, player must decide whether to stop or go.

(E)SA - Science

Name of Program	Cat	Grade	PST	Mem	Description
ACCELERATION.40	GS	ISC	200	16k	Player estimates what the initial velocity of a ball must be for it to fall into a cup.
ALT.&AZIMUTH.40	U	ISC	200	16k	Calculates the positions of several stars.
BALANCE CHEM.40	DT	IS	200	32k	A tutorial/drill on balancing equations.
BIG OHM'S LAW.40	D	JI	302	16k	A drill on Ohm's law, using large numbers.
BROWNIAN.40	ST	JI	300	16k	A good simulation of Brownian motion.
BUOYANCY.40	DT	ISC	250	32k	A tutorial/drill on the concepts of mass, weight and buoyancy.
CASCADE.40	S	J	110	16k	A simulation of a waterfall.
CHARGED PART.40	ST	ISC	250	32k	A simulation and tutorial focusing on electron mass measurement.
CHEM 12.40	D	JIS	202	16k	A drill on nomenclature and the ratio of atoms from different elements in a compound.
CHEM CALC.40	U	JIS	200	16k	Calculates various chemical ratios and quantities given other known quantities.

CONT. (E)SA - SCIENCE....

CHEM QUIZ.40	D	SC	203	16k	A drill on symbols, valences and names of elements.
CHEM. PROB.40	DT	IS	202	32k	A drill on the 'mole' concept, and on conversion from and to particles, mass and volume.
CHEMIST.40	S	JISC	101	16k	Student attempts to dilute a dangerous acid to the correct chemical ratio.
CHEMISTRY.40	D	IS	202	16k	A drill on various aspects of chemistry. The questions are randomly chosen.

(E)SB - Science

Name of Program	Cat	Grade	PST	Mem	Description
CIRCUIT 3.40	U	ISC	200	16k	Calculates current through a resistor given its resistance and the voltage.
CIRCUIT 4.40	T	ISC	200	16k	Tutors the student on capacitors.
COMPOUNDS.40	D	ISC	202	16k	A drill on chemical nomenclature.
CONPRESS.40	U	I	200	16k	Converts measurements in one unit of pressure to another.
CYLINDER.40	DT	JI	322	32k	A drill/tutorial on reading graduated cylinders.
DECAY.40	U	I	200	16k	Calculates and graphs half-life and mass for decay.
DEFECT.40	TU	ISC	200	16k	Calculates and reviews concepts dealing with mass defect of isotopes.
DENSITY.40	U	ISC	200	16k	Calculates density given mass and volume.
DRILL SI.40	DT	IS	342	16k	A drill on metric conversion within metric.
E- CONFIGUR'N.40	T	ISC	300	32k	A tutorial on Schrodinger's model of the atom and placement of electrons in orbitals.
E.M.T.40	GS	ISC	210	32k	Student assumes the role of a doctor asked to diagnose various cases.
EARTHQUAKE.40	DT	ISC	210	32k	Tutors student on finding epicenters of earthquakes. Requires a handout.
ELECT. QUIZ.40	D	IS	201	16k	A drill on Ohm's law.
ELECTRICITY.40	TD	ISC	212	32k	A tutorial and drill on Ohm's law, energy, power, and energy cost problems.
ELECTRO MAG 2.40	ST	ISC	100	32k	A tutorial, with good graphics, on the applications of electromagnetism.

(E)SC - Science

Name of Program	Cat	Grade	PST	Mem	Description
ELEMENT QUIZ.40	D	ISC	201	16k	A quiz on chemical elements and symbols.
ELEMENTS.40	D	ISC	210	16k	A drill on the chemical symbols.
ENZYMES.40	ST	ISC	200	16k	Demonstrates the effect of various factors on the functioning of enzymes.
EQUIVALENTS.40	DT	ISC	243	32k	A tutorial/drill on chemical equivalents, molarity and normality.
FAST FOURIER.40	U	CS	200	16k	Fourier transformations and analysis of curves. The computer decomposes complex waves into components.
FISHING.40	GS	IS	202	16k	Player tries to estimate the number of trout in each of several lakes by catching, marking and returning fish.
FOOD CHAIN.40	D	JIS	200	32k	A drill having to do with the placement of organisms in a food chain.
FORCE CONV.40	U	JISC	100	16k	Performs conversions from one unit of force/mass to another.
FREQ & TIME.40	ST	ISC	300	16k	The student must estimate the period of a revolving square. Reviews the basics of frequency and period.
FUSE.40	D	JI	201	16k	A drill on choosing the appropriate fuses to handle a given current.
GAS EQUATIONS.40	D	ISC	200	16k	A drill on gas volumes, temperatures and pressures.
GEIGER.40	ST	IS	200	16k	An accurate simulation of a Geiger counter sensing radioactive samples.
GRAD CYLINDER.40	TD	ISC	210	16k	A tutorial and drill on reading graduated cylinders.
GRAVITY QUIZ.40	D	ISC	201	16k	This program is a quiz on planetary orbits and gravity.

(E)SD - Science

Name of Program	Cat	Grade	PST	Mem	Description
HARMONICDSPLY.4	U	ISC	200	16k	Graphs the effect of harmonics on the fundamental wavelength.
IDEAL GAS LAW.40	S	ISC	890	16k	A simulation of an experiment involving Boyle's and Charles' law (PV=nrT).
INTERFERENCE.40	U	ISC	200	16k	Program graphs waves separately along their interposed image.
INTERMODUL'N.40	T	ISC	200	16k	Calculates the intermodulation distortion products for every combination of frequencies entered by the user.

CONT. (E)SD - SCIENCE....

IONS.40	D	ISC	203	16k	A drill on the formulae and valences of ions and radicals.
KINEMATICS.40	D	ISC	203	16k	A drill on kinematic problems concerning the motion of a ball thrown vertically upwards.
LEVER.40	DS	JISCT	302	16k	Student learns to balance a simulated lever by altering the distance between the weight and the fulcrum.
LOCK-KEY.40	ST	ISC	200	16k	Shows the effects of inhibitors on the enzyme acetycholinesterase.
MAGIC POWDER.40	G	JIS	202	16k	Student deduces the identity of a mystery powder by the process of elimination.
MALARIA.40	GS	ISC	203	16k	Player administrates funds to build hospitals and provides medical supplies to combat an outbreak of malaria.
MASS.40	U	ISC	200	16k	Calculates the gram molecular mass of any compound, given the number and type of elements it contains.
MATCHING QU.40	D	JI		16k	Student matches questions on dispersion, solute, suspensions, etc. to correct answers. Responses are timed.
MATCHING SOL.40	D	ISC	201	16k	A drill in which the student is required to match questions to answers.
METRIC CONV.40	U	JISC	300	16k	A program which performs interactive metric conversions.
MILLIKANS EXP.40	S	ICS	800	16k	Simulates Millikan's oil drop experiment.
MINI EDISON.40	GS	ISC	202	16k	Student is put in charge of operating a simulated power station.
MITOSIS.40	ST	ISC	010	32k	Tutors the student on mitosis, using good graphic presentations.

(E)SE - Science

Name of Program	Cat	Grade	PST	Mem	Description
MOLARITY.40	U	IS	200	16k	A program which converts mass to moles to molarity.
MOLE CONCEPT.40	D	ISC	412	32k	A drill on converting from moles to gram molecular mass. The program has a built in calculator mode.
MOLECULAR LES.40	DT	ISC	230	32k	Tutorial and drill on VSEPR (Valence Shell Electron Pair Repulsion) method of determining shapes of molecules.
MOLECULE RACE.40	ST	ISC	300	16k	Simulates diffusion of molecules across space.
MOMENTUM CAI.40	DT	ICS	252	16k	Computer assisted instruction on momentum problems.
MOMENTUM TEST.40	D	ISC	202	16k	A quiz on momentum. Student should be familiar with 'MOMENTUM CAI.40' beforehand.
MOTION PROB.40	DT	ISC	200	32k	A tutorial and drill on problems in kinetics.
MUTANT.40	S	ISC	200	16k	Simulates the mutation of peppered moths to black moths within a population.
NICHE.40	S	ISC	201	32k	Simulates the effects of placing organisms in different habitats, with user controlling several variables.
NUC REACTOR.40	GS	ISC	311	32k	A simulation game in which the student controls the operation of a nuclear reactor.
OHM2.40	D	ISC	302	16k	Asks random questions on Ohm's law, with a time limit and scorekeeping provided.
ORBIT PLOT.40	S	JIS	200	16k	Plots the orbit of a satellite around a mass.
ORBIT.40	G	JISC	200	16k	User must locate and destroy an invisible spaceship by guessing its distance and angle in degrees.

(E)SF - Science

Name of Program	Cat	Grade	PST	Mem	Description
PALKO'S AUDIT.40	U	ISC	200	16k	Calculates average energy consumption given the frequency of use of several different appliances.
PERCENT COMP.40	U	ISC	200	16k	A chemistry utility program which calculates percent composition of each element in a compound.
PERCENT.40	U	ISC	200	16k	Calculates the percent composition of an element in a compound.
PERIODIC PROP.40	T	ISC	600	16k	Generates bar graphs of periodic properties vs atomic number.
PH PROBLEMS.40	DT	ISC	241	32k	A tutorial and drill on pH concepts.
PHOTEL.40	DS	ISC	200	16k	Given frequency of x-rays, user must find the voltage setting which causes a collector current to reduce to 0.
PHOTOSYNTH.40	S	ISC	210	16k	Simulation of an experiment varying the factors affecting the rate of photosynthesis.
POLLUTION.40	S	IS	200	16k	Simulates the depletion of oxygen in water systems by factors such as waste, temperature, treatment, etc.
PROJ.MOTION.40	S	JIS	200	16k	Plots trajectory of a projectile given initial height, angle of elevation and velocity.
PROJECTILE.40	DT	ISC	542	32k	Computer-assisted instruction on projectile problems.
RADIO DECAY.40	T	ISC	200	16k	Calculates one of the unknowns in the formula for radioactive decay.
RATE 1.40	T	JIS	200	16k	Demonstrates the effect of different factors on the rate of a reaction.
REG.POW.SUP.40	U	SC	100	32k	Prints out circuit diagrams for specified power supplies.
RESIST TEST V.40	D	IS	202	16k	A timed quiz on resistors.

(E)SG - Science

Name of Program	Cat	Grade	PST	Mem	Description
RESISTANCE.40	DT	ISC	250	32k	A tutorial and drill on series and parallel circuits.
RESISTORS.40	DT	IS	211	16k	A tutorial/drill on parallel and series circuits.
RMDL CHEMIST.40	DT	ISC	413	32k	A tutorial/drill on elements, radicals and acids.
SOLAR SYSTEM.40	ST	PJIS	200	32k	Displays a diagram of the solar system and provides information about it.
SPECIFIC HEAT.40	U	C	200	16k	Helps teacher to calculate and mark specific heat problems.
STOICHIOMETRY.40	U	ISC	200	16k	Calculates stoichiometric unknowns given information input by the user.
TEMP. CONV.40	DT	JIS	210	32k	A tutorial/drill on Kelvin and Celsius temperature scales.
TITRATE.40	ST	ISC	302	16k	A simulation of a titration experiment designed to give the student practice in that procedure.
VELOCITY.40	S	ISC	200	16k	Simulation of an experiment involving a cart. Requires a stop-watch, graph paper and a calculator.
VERNIER SCALE.40	DT	JISC	202	16k	A tutorial/drill on reading a vernier scale.
WAVES 3.40	ST	JIS	300	16k	A tutorial on, and simulation of, interference patterns.
WEATHER MAN.40	U	IS	200	16k	Converts temperatures and calculates humidex, wind chill factor, etc.
YOUNG'S.40	S	IS	200	16k	A simulation of double slit diffraction.

(E)TA - Technology

Name of Program	Cat	Grade	PST	Mem	Description
BRAILLE.40	DT	PJISC	200	32k	A tutorial and drill on Braille.
CIRCUITS.40	DS	JISC	722	32k	Drill on current flow through circuits. From a circuit diagram, student determines whether a lamp is on or off.
HOME.40	U	JISC	201	32k	Generates a graph of energy consumption in a typical home in Peterborough.
METER READING.40	D	IS	302	32k	Review and drill on the reading of a multimeter voltmeter scale.
MORSE CODE.40	D	JISC	202	16k	Program presents a letter in Morse code and gives the student 3 chances to identify it.
MULTIMETER.40	DT	ISC	312	32k	Tutorial and drill on reading voltmeters and micrometers.

(E)UA - Utilities

Name of Program	Cat	Grade	PST	Mem	Description
ALPHA SORT.40	U		410	16k	Sorts a list of names in alphabetical order.
BASE CHANGE.40	U	JIS	200	16k	A utility program which changes numbers from base 10 to bases 2-16. Input number range is 1 to 16,775,215.
COPY-ALL (HD)	U	JISC		16k	A utility for copying programs from one disk to another.
DISK DISPLAY.40	U		000	16k	A demonstration of PET disk drive commands.
DISK LISTER.40	U		000	32k	Stores directories of several disks on one disk.
GRAPH SUBRT.40	U	IS	000	16k	A subroutine which permits plotting in quarter character graphics; can be merged into a user's program.
HEXADECIMAL.40	U	JI	100	16k	A utility which enables conversion between hex and decimal numbers.
KEYBOARD.40	U		110	32k	Provides instruction in the use of the keyboard - cursor controls, graphic characters, etc.
MEMSEE.40	TU	IS	200	16k	Demonstrates how basic is stored in a microcomputer. Displays basic text and numeric storage; printed output.
UNCOMPACTOR	U		200	16k	Uncompacts programs from multi-statement lines to single statement lines.

Best Programs

PAGE

Byteds (C-64 and VIC) 262
Bill Yee, Winnipeg, Man.
This programmer's tool is very useful for managing hexadecimal data entry and storage.

Wedge-64 264
Darcy Mason & Alan Wunsche, Oshawa, Ont.
How to use all those wedge commands and what they will do.

VIC Micromon 267
Bill Yee, Winnipeg, Man.
Every programmer needs a monitor program and needs to know how to use it. So here is one of the best for the VIC with the instructions. Similar versions are available for the 64.

Best Programs — Diskette 272
Darrin McGugan, Shelburne, Ont.
What it is and where to get it.

Byteds (C-64 and VIC)

By Bill Yee, Winnipeg, Man.

This program is on The Best Programs Disk

BYTEDS is a compact little utility, written in BASIC, that allows VIC-20 and C-64 users to freely access and save any part of memory. As a tool for managing hexadecimal data entry and storage, its data saving capability has the possibility for misuse. My intentions for this utility is to allow users to achieve a better understanding of their VIC-20 or C-64 systems and also to allow them to explore the world of hexadecimal data bytes and machine language coding. I believe its value in these areas far outweighs any possible misuse. In the VIC-20, the system routines deter misuse by preventing tape saves of memory areas at $8000 and higher.

The utility is loaded and run like any other BASIC program. It has two main sections — the hexadecimal data editor and the memory image saver. The editor is always entered first to allow the user to view and modify data bytes in memory. Upon completion of editing, the editor is exited to invoke the saver section. The saver section allows the user to save memory to either disk or tape. Addresses for memory locations are inputted in either decimal or, if prefixed with $ sign, are interpreted as hexadecimal.

The editor displays the address of the current memory location being edited as a 4 digit hexadecimal number. The contents of the location are shown on the same line as a 2 digit hexadecimal number immediately after the address. To enter a new data byte into the displayed location, key in any 2 digit hexadecimal number. Only digits from 0 through 9 and alpha characters A through F are accepted. If a mistake is made, continue input as only the last two characters are used in writing the new data byte to memory.

After keying in a new data byte, memory write is done by hitting either RETURN, SPACE bar, or the UP/DOWN CRSR key. If memory is successfully written, the current location address is incremented except for the UP CRSR which causes the address to decrement after write. If the location cannot be written, the address remains constant and the contents are redisplayed followed with the message "R/O".

If no new data is keyed in, the location address is incremented or decremented by respectively keying the DOWN or UP CRSR. The SPACE bar redisplays the same location and data on the same line. This provides a monitoring capability for hardware registers such as timers and ports.

Hitting RETURN results in a new request for an edit location address. If no address is inputted, another RETURN ends the editor and starts the memory image save section of BYTEDs.

BYTEDS prompts the user if memory save is desired. A reply of "N" for No ends BYTEDS. A reply of "Y" for Yes causes prompts for start and end addresses for the memory area to be saved. Address values are decimal unless prefixed with the $ sign to indicate hexadecimal. Prompts are then made for device (Disk or Tape) and a mandatory file name. The save is then done.

The data on disk or tape can be reloaded into memory from where it was originally saved by using the non-relocatable form of the load command in the BASIC environment.

For disk: LOAD "file name", 8 , 1
For tape: LOAD "file name", 1 , 1

The load will disturb the BASIC memory pointers so a NEW is needed to reset the pointers. If the reloaded data is to be modified and re-saved, BYTEDS would be reloaded after reset of the pointers.

BYTEDS is a utility that I suspect most users will want to keep handy after they have tried it a few times. It should prove useful even after more sophisticated tools such as a monitor or an assembler package is acquired.

OK...Now what does it pick for the daily double?

BYTEDS BY B. YEE

```
0 PRINT"BYTE EDIT/SAVE FOR VIC & CBM-64 BY B YEE"
1 GOSUB7:L=D:C=99:REM BYTE EDIT
2 GOSUB6:L=D
3 GOSUB14:L$=H$:D=PEEK(L):GOSUB15:PRINTL$": "H$" ";:GOSUB20:IFH$=""AND
  C=32THENPRINT"□":GOTO3
4 IFH$<>""THENH$=RIGHT$(H$,2):GOSUB10:POKEL,D:IFPEEK(L)<>DTHENPRINT"R/O"
  :GOTO3
5 GOTO2
6 PRINT:IFC<>13THEND=L+SGN(99-C):RETURN
7 H$="":INPUT"LOC";H$:IFH$=""THEN25
8 IFLEFT$(H$,1)<>"$"THEND=VAL(H$):RETURN
9 H$=MID$(H$,2,LEN(H$)-1):H$=RIGHT$(H$,4)
10 N=LEN(H$):D=0:FORM=0TON-1:C$=MID$(H$,N-M,1):H=ASC(C$)-48:IFH>9THENH=H-7
11 D=D+H*16↑M:NEXT:RETURN
14 D=L
15 IFD<0ORD>65535THENPRINTD"OOR":END
16 H$="":M=4096:N=3:IFD<256THENM=16:N=1
17 FORH=0TON:C=INT(D/M):D=D-C*M:M=M/16:C=C+48:IFC>57THENC=C+7
18 H$=H$+CHR$(C):NEXT:RETURN
20 H$=""
21 GETC$:IFC$=""THEN21
22 C=ASC(C$):IFC=13ORC=17ORC=32ORC=145THENRETURN
23 IFC<48ORC>57ANDC<65ORC>70THEN21
24 PRINTC$;:H$=H$+C$:GOTO21
25 PRINT"SAVE DATA? ■Y■ES OR ■N■O"
26 GETC$:IFC$="N"THENEND
27 IFC$<>"Y"THEN26
28 PRINT"START ";:GOSUB7:SL=D:PRINT"END ";:GOSUB7:EL=D+1
29 AR=780:XR=781:YR=782:SR=783
30 N$="":INPUT"SAVE FILENAME";N$
31 IFN$=""THEN30
32 POKE183,LEN(N$):POKE187,PEEK(51):POKE188,PEEK(52)
33 PRINT"■D■ISK OR ■T■APE?"
34 GETC$:IFC$="D"THENDN=8:GOTO37
35 IFC$<>"T"THEN34
36 DN=1
37 POKE185,0:POKE186,DN
38 POKE251,SL-256*INT(SL/256):POKE252,INT(SL/256):POKEAR,251
39 POKEXR,EL-256*INT(EL/256):POKEYR,INT(EL/256)
40 POKE157,128:SYS(65496):REM SAVE TO DEVICE
41 PRINT:PRINT"**** TO RETRIEVE ****"
42 PRINT"LOAD"CHR$(34);N$;CHR$(34)","DN",1"
43 PRINT"NEW"
44 PRINT"LOAD"CHR$(34)"BYTEDS"CHR$(34)","DN
45 END
READY.
```

WEDGE-64

By Darcy Mason & Alan Wunsche, Oshawa, Ont.

This program is on The Best Programs Disk

HISTORY OF WEDGE-64

The **Wedge-64** was developed in the summer of 1983 during the Summer Canada '83 project sponsored by the Durham Board of Education. The "**Wedge**" actually started out as a machine language merge utility for Commodore 64 users. After the merge was completed, we realized that the C-64 was lacking in good catalog routines. At about the time that the catalog and disk status routine was done, we decided to add a few more commands. **Hex, look**, and **hunt** followed and we soon began thinking that our program could evolve into an aid that would help the C-64 programmer create BASIC programs more easily and efficiently. Below is the result of our efforts. As with most programs, there may be some bugs that weren't spotted in the debugging process. We would be interested in hearing about any questions, problems, or comments about the **Wedge**, as we would like to make this program as good as possible.

SUMMARY OF COMMANDS

Wedge = ►

►adjust	displays all colours for adjustment of monitor.
►auto	prints out next line number in given increment.
►cold	resets computer.
►colour	sets border, screen, and cursor colours.
►del	deletes a given range of lines.
►ds	displays disk status.
►help	lists all commands of the **Wedge** summarized here.
►hex	gives decimal of hex number or vice versa.
►hunt	searches through program for given string.
►look	displays values in variables currently in use.
►mem	memory dump in hex with ASCII on right hand side.
►merge	merges program from disk with one in memory.
►n	displays last filename used in **Wedge**.
►off	disables **Wedge**.
►renum	renumbers program or line range.
►save	saves program to disk.
►start	displays load address of file on disk in hex and decimal.
►send	sends command string to disk via command channel.
►$	displays directory of programs on disk.
►/	loads program from disk.

INDIVIDUAL DESCRIPTIONS

►adjust — This command takes no parameters. Bars of each of the sixteen colours are displayed, allowing for easy colour adjustment of the video monitor being used.

►auto (inc)► — If AUTO is called with no increment, then AUTO mode is cancelled. When AUTO mode is in effect and the user has just typed in a BASIC program line, the **Wedge** will automatically print the next line number to be entered. The next line number is given by the line just entered plus the increment. The increment must range from one to 255.
Example:

►auto 10 — sets auto mode with increment of ten.

►cold — This command takes no parameters. The computer is reset, similar to when it is first turned on. Any BASIC program will be erased. If the **Wedge** is in memory at $C000, then it may be re-enabled with SYS 12*4096. A **Wedge** located at $9000 is re-enabled with SYS 9*4096.

►colour (border, background, cursor) — The colour command sets the colour of the screen border, background, and cursor, as shown above.

The colour numbers may range from zero to 255, anything not within this range will result in an ILLEGAL QUANTITY error. Example: **►colour 0,2,1** — sets black border, red background, and white cursor.

►del (line range) — This function will erase all BASIC program lines within the given line range. Line ranges are the same as that of the LIST command in BASIC. If no line range is given, then no lines will be deleted.

Examples:

►**del 10—** — deletes from line 10 on.
►**del 10—100** — deletes lines 10 to 100 inclusive.
►**del —100** — deletes lines up to and including line 100.

►**ds** — This command will display the disk status. This routine is called by all other disk-based routines to report the status after every disk access.

►**help** — Upon calling this routine, a command summary will be printed as a help screen to remind the user of all the commands of the **Wedge**.

►**hex ($) number** — This function will give the decimal equivalent of a hex number and vice versa. A hex symbol ($) must preceed the hex number to be converted. The hex number must have four digits (e.g. $003E). The decimal number may be an expression (e.g. 9*4096) but must be less than 65535.

Examples:

►**hex $fd16** — gives decimal equivalent (64790).
►**hex 59468** — gives hex equivalent ($E84C).

►**hunt "search string" (line range)** — This function will search through a BASIC program for a string and list all lines which contain the string. The delimiter may be any character although it should be noted that any delimiters other than quotes (") will tokenize the string. e.g. ►**hunt** "end" will find the lines with the word **end** within quotation marks. To find lines with the tokenized **END** use ►**hunt "end"**. The line range (same format as **LIST**) is optional and, if not found, the search will take place through the entire program. Note that **hunt** changes the string displayed by ►**n**.
Examples:

►**hunt "print"** — searches entire program for token **PRINT**.
►**hunt "the",10–** — displays all occurrences of **"the"** from line 10 on.

►**look** — This command will display all variables currently in use and their values. Dimensioned variables are not displayed. The program must already have been **RUN** for this command to work.

►**mem start hex address, end hex address** — This function will display the hex value of the specified memory locations in groups of eight bytes. The ASCII equivalent of the bytes displayed is also given in reverse field at the right hand side of the screen.

Example:

►**mem 1000,2000** — displays the contents of memory locations $1000-$2000.
►**mem 0300** — displays memory from $0300 on, until stop key pressed.

►**merge "filename"** — This command will merge the specified BASIC program from the disk with the one in memory. Any lines in memory that coincide with ones from the disk will be replaced by the lines from disk.
Example:

►**merge "file2merge"** — merges program called "file2merge" from disk with program in memory.

►**n ("filename")** — When no filename is specified, the last **Wedge** filename or hunt string used will be displayed. If a filename is given then the filename is set. In the first case, the cursor will be kept on the line of the last file displayed.
Example:

►**n** — displays last filename used.
►**n "newfile"** — sets filename to "new file". Note that this command may be used when a file on disk is accessed repeatedly. This command eliminates the need to retype the filename. Also note that ►**hunt** changes the string displayed by this command.

►**off** — This command disables the **Wedge**. To reenable the **Wedge** at $C000, type SYS 12*4096.

►**renum (start line #, inc, (line range))** — This function will renumber a BASIC program beginning with the start line #, increasing the line numbers by the increment. If a line range is given then the renumber takes place only within the line range. If no parameters are entered then the whole program is renumbered starting from line 100, in increments of 10. The increment must range from one to 255.
Examples:

►**renum** — renumbers entire program 100,10. The new starting line will be 100.
►**renum 200,5** — renumbers entire program 200,5. The new starting line will be 200.
►**renum 100,10,100–** — renumbers lines 100 on, by 100,10.

►**save "filename" (,start hex address, end hex address + 1)** — This save command will default to disk. If no parameters are given the BASIC program in memory is saved. If parameters are supplied then memory is saved within the given range. This command is useful in saving machine code to disk.

Examples:

►**save "filename"** — saves BASIC program to disk.
►**save "filename",2000,3001** — saves memory from $2000-$3000.

►**send "disk command"** — This function sends a disk command to the disk drive through the command channel. The disk command is in BASIC 2.0 form.
Example:

►**send "i0"** — sends an initialized command to drive 0.

►**start "filename"** — This function will fetch the load address of a program on disk and display it in hex and decimal.

►**$** — This command will display a directory of programs on the disk. Pattern matching is supported. The program in memory is **not** destroyed by a directory.
Examples:

►**$0** — displays directory of drive 0.
►**$:st*** — displays directory of all files beginning with the letters "st".
►**$:*** = **seq** — displays all sequential files.

►**/"filename"** — This command will load the specified file from disk without relocating it. This is identical to the format : load "filename",8,1.

ADDITIONAL INFORMATION

— None of the **Wedge** commands have abbreviations as BASIC commands do, i.e., the whole word must be entered.

— Any disk commands will cause the computer to "hang up" if the disk drive is not connected. If any problem with disk access occurs, then press RUN/STOP and RESTORE simultaneously to regain control.

— All disk commands operate on disk device 8.

— The commands ►**hunt,** ►**look** and ►**mem** may be frozen by pressing the shift or shift lock keys. Release the shift key to continue.

— The **Wedge** is CHRGET-driven and may therefore co-exist with an interrupt-driven machine language routine at another location.

HER RECIPE PROGRAM CRASHED AND SHE SAYS NO DINNER TILL SHE GETS IT ON LINE!

VIC Micromon

By Bill Yee, Winnipeg, Man.

This program is on The Best Programs Disk

Attention VIC users! Have you been trying to do more with your VIC than just BASIC programming and games playing? Is the relationship with your VIC getting a little stale? Well, here is a powerful new utility that will open another dimension in your VIC. It provides all the BASIC support that is needed for assembler language development. Assembly language is the language of the "pros" and is the only language used for fast action-packed game programs.

The utility is a machine language monitor called VIC Micromon V1.3 which was developed over the last year. It has a single line 6502 assembler, disassembler, machine language debug, data editor, data convertor, data storage and retrieval, and EPROM programmer I/O. A total of 37 commands are provided by a monitor that is exactly 4K bytes in size. This allows the monitor to be used in a VIC with just 3K bytes of RAM expansion. I have supplied a club with two versions of the monitor. The first named "MICROMON @ $0E00" loads into $0E00-$1DFF for a VIC with only the 3K memory expansion. The second called "MICROMON @ $3000" loads into $3000-$3FFF for a VIC with the 8K memory expansion.

The monitor is loaded with the non-relocating form of the LOAD command. Be sure to specify the name of the version required for your particular (3K or 8K) memory expansion. For disk the command is LOAD "file name" ,8,1 and for tape is LOAD "file name" ,1,1. Access the monitor at $0E00 with a SYS3584 and at $3000 with a SYS12288. Response will be a title, user image, and period prompt.

Due to the fact that BASIC and some kernal routines use workspace at the top of memory, the first command upon accessing the monitor should be a reconfiguration of memory. This is done with the I (Initialize memory and screen) command. If you have the monitor at $0E00, use I 0438 0E00 IE. The reason 0438 is used instead of 0400 is because the monitor defines the tape buffer as being from $0375 to $0434. If you have the monitor at $3000, use I 1200 3000 10.

If you have 16K of memory expansion, you may wish for a version located at $7000 to $7FFF. Well, with a little bit of work you can have your wish. You can use the monitor commands to generate a new copy at $7000 to $7FFF with all of the addresses relocated. The following example starts with a copy at $3000-$3FFF and ends with a copy at $7000-$7FFF. The copy at $3000 is not changed and can be executed to do the relocating commands.

```
T 3000 3FFF 7000
N 7000 7003 4000 3000 3FFF
N 7015 7E6D 4000 3000 3FFF
N 7FB5 7FFE 4000 3000 3FFF W
```

Once you have done these commands, there are 6 locations which must be individually changed. Use the Memory display and colon commands to make the changes shown in Table 1.

TABLE 1: Individual changes to relocate from $3000 to $7000

Location	Old Value	New Value
7018	35	75
102A	33	73
7392	3C	7C
7650	35	75
76E7	35	75
7897	33	73

The last location in the monitor is only used to make the 4K checksum be evenly divisible by 256. This makes it easy to verify the integrity of the program. The copy at $7000 to $7FFF has a last byte value of $4E at $7FFF to give a checksum of $1500.

After completing the last change, exit with the E command to BASIC and use a SYS28672 to access the new copy at $7000 for check out. If OK, save it to disk with S 7000 8000 "MICROMON @ $7000" 08 or to tape with S 7000 8000 "MICROMON @ $7000".

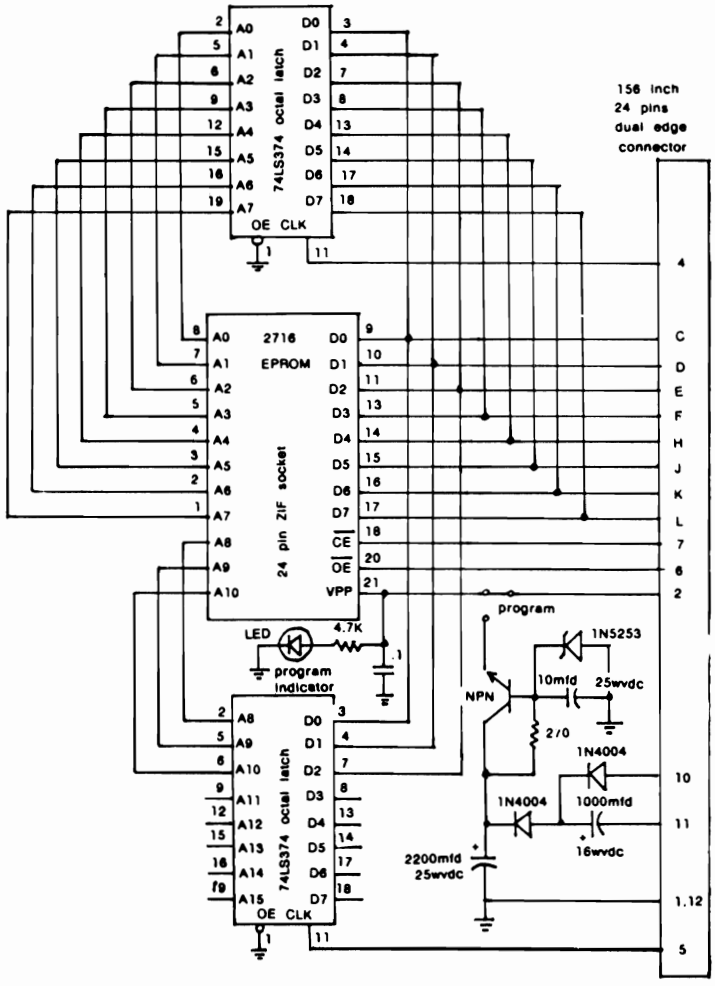
What next? I'll bet that some of you are now wishing that it could be put on EPROM. Well, if you build the EPROM programmer whose schematic is shown on figures 1 and 2, you can use the EPROM commands in the monitor to "burn" your own copies. There is no self-modifying code so the monitor will run just as well in EPROM.

For the more advanced VIC enthusiast, it is only a minor step to getting an EPROM version located at $A000 in the games cartridge area. I've already included the required calls to kernal routines for initialization on power-up. First, you must relocate your copy of the monitor to $A000-$AFFF then make the additional changes shown in Table 2.

TABLE 2

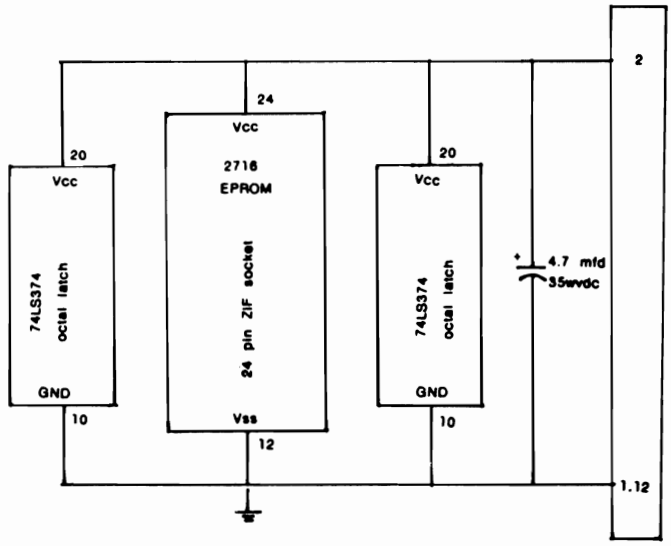
Location	Value
A000	09
A001	A0
A002	C7
A003	FE

FIGURE 1



BUILD YOUR OWN EPROM PROGRAMMER

FIGURE 2



Note: As Vcc is 5 volts, the capacitor wvdc is not critical.

Connector for VIC 20 USER I/O port

TABLE 2: Additional individual changes for cartridge version

After making a copy on EPROM and installing as a cartridge, the monitor will be entered immediately after power-up of the VIC-20. Use the E command to exit to BASIC. Once in BASIC, always use SYS40981 to re-access the monitor at $A015.

Following is a list of the VIC Micromon V1.3 commands shown with examples. It should be sufficient to get you started on assembly language development on your VIC-20. I hope that many of you will enjoy the new dimensions opened to you by this powerful monitor.

COMMANDS

A period prompts the user for a command. All the commands consist of a single character. One or more operands may follow depending on the command. Here are examples of each available command.

Assembler

.A 2100 LDX #$12 : COMMENT
Assemble and display address, machine code and instruction. (use colon only if comment)

.A 2100 A0 12 LDA #$12

.A 2102 (hit RETURN to exit)

Break Set

.B 2102 0010 (Walk on 17th pass of $2102)

Compare Memory

.C 05F0 05FF 0600 ($5F0-$5FF compare to $600-$60F)

Disassembler

.D 2100 2102 (second operand optional)
Disassemble and display address, machine code and instruction.

., 2100 A0 12 LDX #$12

., 2102 CA DEX

Exit Micromon

.E (clear linkages & exit to BASIC)

Fill Memory

.F 2000 27FF 00 (zero $2000 to $27FF)

Go Run User Code At Full Speed

.G (execute user image PC location)

.G 2100 (start execution at $2100)

Hunt Memory For Up To 32 Byte String

.H C000 DFFF 'CBM
(look for character string CBM)
.H C000 DFFF 43 42 4D
(look for byte string 43 42 4D)

Initialize Memory and Screen Pointers

.I 1000 1E00 1E
(initialize as unexpanded VIC 20)

Jump To Micromon Subroutine

.J 2100 (call subroutine at $2100)

Load

.L 4000 "DATA FILE" 08
(load from disk into $4000)
.L 4000 "DATA FILE" 01
(load from tape into $4000)

Memory Display

.M C23B C243 (second operand optional)
Display address, 8 bytes in hexadecimal, and ASCII translation.

.: C23B 52 45 54 55 52 4E 20 57 RETURN W
.: C243 49 54 48 4F 55 54 20 47 ITHOUT G

New Locater

.N 7015 7E6D 4000 3000 3FFF
(relocate instruction addresses)
.N 7FB5 7FFE 4000 3000 3FFF W
(relocate word addresses)

Offset Branch Calculate

.0 2103 2102 FD

Print Switcher

.P CCBB (set command = CC & control = BB)

.P (switch output to screen or port)

.P 0000 (clear port and output to screen)

Note: For VIC printer, use OPEN4,4:CMD4 then access Micromon.

Quick Trace

.Q (execute user image PC location)

. 2100 (start execution at $2100)

Register Display

.R

Display user image with title strip.

PC IRQ SR AC XR YR SP

.;OE4E 1191 32 33 00 00 F7
(user image on entry at $0E00)

Save

.S 2100 21FF "file name" 08
(save $2100-$21FF to disk)
.S 2100 21FF "file name" 01
(save $2100-$21FF to tape)

Transfer Memory

.T 05F0 05FF 0600
($5F0-$5FF copied to $600-$60F)

Verify

.V 2100 "file name" 08
(verify against file on disk)
.V 2100 "file name" 01
(verify against file on tape)

Walk Code

.W (execute user image PC location)

.W 2100 (start execution at $2100)

Do single instruction step, then display user image as SR, AC, XR, YR, SP, then next instruction address, code and disassembly. Hit RUN/STOP to stop walk. Hit J to execute subroutine at full speed with walk on exit from subroutine. Any other key does single step.

Exit to BASIC

.X (linkages exist on exit)

ASCII Conversion

."A

."A 41 65 0100 0001
(ASCII, hex, decimal & binary)

Binary Conversion

.%0100001001011001

Display binary value in hexadecimal, decimal & ASCII characters.

.%0100001001011001 4259 16985 B Y

Decimal Conversion

.#16985

Display decimal value in hexadecimal, ASCII characters & binary.

.#16985 4259 B Y 0100 0010 0101 1001

Hexadecimal Conversion

.$4259
Display hexadecimal value in decimal, ASCII characters & binary.

.$4259 16985 B Y 0100 0010 0101 1001

Addition

+ 6000 7000 D000

Subtraction

— FFFF 7000 8FFF

Checksum

.& C000 CFFF 2DFC

Command End Tone

.((tone at end of next command)

Hit return to shut off tone between commands.

.) (tone disabled)

User Image Modify

.; 2102 1191 32 12 00 F7
(output from R command)

Machine Code Modify

., 2100 A0 12 xxx (output from D command)

Use cursor to modify code in line. RETURN writes memory, reads memory, disassembles and displays as follows for the example.

., 2100 A0 12 LDX #$12

Memory Data Modify

.: 2110 40 41 (output 8 bytes from M command)

If 8 bytes input, ASCII conversion and next address is displayed.

Screen scrolling will occur when output from D, M or $ command is displayed and cursor is moved eithr to top or bottom of screen.

Program 2716 EPROM

WARNING: To avoid destroying EPROM, do EPROM insertion or removal:

*Always with EPROM programming voltage OFF (switch at +5)

*Always read EPROM before turning on programming voltage.

.(pi) 5000 57FF 00
($5000-$57FF programmed into EPROM starting at page 00 in EPROM)

Read 2716 EPROM

.(lb.) 0600 09FF 04
(last 1K of EPROM into $600-$9FF)

Compare 2716 EPROM

.=0500 05FF 00
(1st 256 bytes of EPROM compared)

by Chaput

SOMETHING TELLS ME WE SHOULD WAIT A FEW DAYS BEFORE WE ASK ABOUT GETTING A COMPUTER.

Best Programs — Diskette

By Darrin McGugan, Shelburne, Ont.

The following is a catalogue of a diskette that contains all the programs of over approximately twenty lines that are listed in this book. It also contains all the major programs discussed in this chapter such as MICROMON (two versions), BYTEDS, the 64 WEDGES (four of them), plus redefinitions of the keyboards to DVORAK (two related programs) that are discussed on page 36, etc.

Since we had quite a bit of space left over we also put on this disk some of the BEST of Series listed on the following pages. The catalogue lists everything that is on the disk and tells you where to find more information about the programs.

Since these programs are all public domain programs you are welcome to either copy them from or to your friends. You may be able to get a copy from the dealer from whom you bought your copy of this book. Most dealers will make you a copy of a public domain disk for around $10 which is quite reasonable when you consider the disk material, the effort involved and their overhead. The programs themselves are **free** and the cost is simply for the copying. Still, when you figure that there are usually about 20 programs on a diskette and if you get the diskette for $10 then that works out to about 50 cents per program, and **that is a real bargain.**

If you are unable to get this diskette elsewhere, you may order it directly from:

TORPET DISKETTES
Horning's Mills, Ont.
L0N 1J0
Canada

TORPET DISKETTES
1 Brinkman Ave.
Buffalo, N.Y. 14211
U.S.A.

Enclose $10 with your order.

THE BEST PROGRAMS DISKETTE — OVER 40 PROGRAMS

SIZE	TITLE	COMPUTER	PAGE #
3	load and run — This program loads and runs any the programs on this disk for you.		
4	disk to tape		
20	list-me		
5	painting	C64	28
2	menu select.1	C64	38
7	menu select.2	C64	38,39
2	detecting format	ALL	142
3	relocate m-code	ALL	97,98
7	friendly menu	C64/PET	74,75
3	conv pet-64	C64	19,20
23	graphic routines	C64	25,26,27
1	between lines.1	ALL	80,81
1	between lines.2	ALL	80,81
1	between lines.3	ALL	80,81
9	g.i.r.-instr.	VIC	48,49,50
4	g.i.r.	VIC	50
2	print hint	C64/PET	83
1	blinking prompt	VIC	77
3	file transfer	C64	125
8	cursor control	VIC	47
8	underline cursor	VIC	76,77
41	keydef	C64	36
9	dvorak	C64	37
10	file convert	C64	—
15	keydef.inst	C64	36
2	convert	C64	104
15	list formater	C64	63-69
5	byteds	C64/VIC	263
7	program func.src	C64	32,33,34
6	program function	C64	34,35
17	micromon@$0e003k	VIC	267
17	micromon@$30008k	VIC	267
17	wedge-64-$9000.c	C64	264
17	wedge-64-$c000.c	C64	264
17	wedge-64-$7000.c	C64	264
17	wedge-64-$8000.c	C64	264
98	monopole	C64	Disk 3
62	graphic tutor	C64	Disk 5
9	jstick doodle.c	C64	Disk 6
23	supermon-instr.	C64	Disk 7
10	supermon loader	C64	Disk 7
11	scopy.64	C64	Disk 7
22	audio teach 64	C64	228
1	boot tiny aid	C64	Disk 7
5	tiny aid $c066	C64	Disk 7
7	vic aid4.rel	VIC	Disk 10
52	piano.c	C64	Disk 6
15	audio teach vic	VIC	228
3	string thing.c	C64	82
28	1541 backup.c	C64	Disk 7
7	file convert.i	C64	—

4 blocks free

Best Disks

PAGE

Introduction to The 10 Best Disks 274
Darrin McGugan, Shelburne, Ont.
What the Best Disks are and where to get them.

Directories of The 10 Best Disks 274
Darrin McGugan, Shelburne, Ont.
These are printouts of the 10 Disk Directories.

Descriptions of The Programs on The 10 Best Disks 278
Darrin McGugan, Shelburne, Ont.
With user instructions.

Disk Description Summary and Checklist 289
Bruce Beach, Horning's Mills, Ont.
A quick reference to the free programs described in this book.

Introduction to the 10 Best Disks

By Darrin McGugan, Shelburne, Ont.

The following diskettes are the **cream of the crop** (or the crème de la crème as the French say) of all the public domain programs that we have in our library. These represent the best programs from five major collections which between themselves probably represent over 6,000 unique public domain programs. We have gone through all the programs listed to test them and document how to use them. Our user documentation follows these catalog listings.

Since these programs are all public domain programs you are welcome to either copy them from or to your friends. You may be able to get a copy from the dealer from whom you bought your copy of this book. Most dealers will make you a copy of a public domain disk for around $10 which is quite reasonable when you consider the disk material, the effort involved and their overhead. The programs themselves are **free** and the cost is simply for the copying. Still, when you figure that there are usually about 20 programs on a diskette and if you get the diskette for $10 then that works out to about 50 cents per program, and **that is a real bargain**.

If you don't happen to be able to get the diskettes from someone else you can get them by writing to:

TORPET DISKETTES
Horning's Mills
Ontario L0N 1J0
Canada

or

TORPET DISKETTES
1 Brinkman Ave.
Buffalo, N.Y.
14211 U.S.A.

Enclose $10 per diskette requested or $70 for all ten.

Best of The TORPET Best Disks

Directories of The 10 Best Disks

By Darrin McGugan, Shelburne, Ont.

GAMES#1.C — Disk 01

3	"load and run"	prg
4	"disk to tape"	prg
8	"list-me"	prg
25	"eliza"	prg
14	"battleships"	prg
52	"labyrinth"	prg
25	"solitaire"	prg
25	"othello"	prg
17	"flight simulator"	prg
27	"pro football"	prg
12	"checkers"	prg
62	"stock market"	prg
22	"card snap"	prg
22	"yahtzee"	prg
23	"bridge bid train"	prg
18	"wet paint.c"	prg
20	"golf"	prg
21	"viper.c"	prg
20	"grandprix.c"	prg
69	"pirate adventure"	prg
25	"bowling"	prg
71	"dog.star.adven"	prg
28	"querk.c"	prg
18	"pet emulator.c"	prg

33 blocks free.

GAMES#2.C — Disk 02

3	"load and run"	prg
4	"disk to tape"	prg
9	"list-me"	prg
10	"motorcycle"	prg
27	"hangman 1"	prg
17	"3d tic-tac-toe"	prg
24	"concentration"	prg
16	"basketball"	prg
26	"blockade"	prg
15	"star wars!"	prg
78	"super star trek"	prg
24	"deepspace"	prg
22	"c.c.starwar.ins"	prg
4	"c.c.starwar"	prg
23	"star trek"	prg
27	"toker"	prg
25	"billiards"	prg
60	"pet nuc pwr plnt"	prg
55	"sorcerers castle"	prg

60	"wizard's castle"	prg
27	"ratrun"	prg
25	"1000 miles"	prg
22	"states & capitals"	prg
18	"power plant"	prg
21	"power instr."	prg
18	"pet emulator.c"	prg

4 blocks free.

GAMES#3.C Disk 03

3	"load and run"	prg
4	"disk to tape"	prg
9	"list-me"	prg
20	"grundy towers"	prg
23	"clue"	prg
27	"new water"	prg
25	"easy dungeon"	prg
25	"niche"	prg
10	"lemonade.c"	prg
21	"pink panther"	prg
50	"lost dutch gold"	prg
39	"quest 3.0"	prg
49	"adventure"	prg
98	"monopole"	prg
22	"spaceshooter"	prg
51	"supertrek/16knr"	prg
27	"osc lander"	prg
26	"keno"	prg
19	"horse race"	prg
23	"draw poker"	prg
18	"black jack 1"	prg
6	"roulette"	prg
20	"slots jackpot"	prg
18	"pet emulator.c"	prg

30 blocks free

TORPET MANUAL.C Disk 04

3	"load and run"	prg
4	"disk to tape"	prg
4	"list-me"	prg
3	"ref.page 20.1"	prg
2	"ref.page 45.1"	prg
1	"ref.page 85.1"	prg
2	"ref.page 110.1"	prg
1	"ref.page 111.1"	prg
5	"ref.page 114.1"	prg
2	"ref.page 117.1"	prg
6	"ref.page 118.1"	prg
7	"ref.page 119.1"	prg
1	"ref.page 123.1"	prg
1	"ref.page 123.2"	prg
2	"ref.page 126.1"	prg
2	"ref.page 127.1"	prg
2	"ref.page 130.1"	prg
2	"ref.page 139.1"	prg
1	"ref.page 142.1"	prg
1	"ref.page 142.2"	prg
7	"ref.page 146.1"	prg
12	"ref.page 147.1"	prg
8	"ref.page 148.1"	prg
1	"ref.page 153.1"	prg
1	"ref.page 159.1"	prg
1	"ref.page 160.1"	prg
1	"ref.page 161.1"	prg
2	"ref.page 163.1"	prg
2	"ref.page 165.1"	prg
5	"ref.page 167.1"	prg
4	"ref.page 181.1"	prg
2	"ref.page 185.1"	prg
7	"ref.page 187.1"	prg
2	"ref.page 193.1"	prg
2	"ref.page 197.1"	prg
2	"ref.page 200.1"	prg
2	"ref.page 203.1"	prg
1	"ref.page 205.1"	prg
1	"ref.page 206.1"	prg
1	"ref.page 207.1"	prg
1	"ref.page 208.1"	prg
2	"ref.page 344.1"	prg
4	"ref.page 347.1"	prg
4	"ref.page 356.1"	prg
3	"ref.page 357.1"	prg
4	"ref.page 371.1"	prg
1	"user.page 43.1"	prg
1	"user.page 44.1"	prg
2	"user.page 46.1"	prg
1	"user.page 47.1"	prg
1	"user.page 49.1"	prg
2	"user.page 51.1"	prg
2	"user.page 58.1"	prg
1	"user.page 65.1"	prg
2	"user.page 71.1"	prg
2	"user.page 76.1"	prg
1	"user.page 78.1"	prg
1	"user.page 80.1"	prg
2	"user.page 81.1"	prg
1	"user.page 85.1"	prg
1	"user.page 86.1"	prg
1	"user.page 87.1"	prg
2	"user.page 88.1"	prg
1	"user.page 90.1"	prg
1	"user.page 90.2"	prg
1	"user.page 99.1"	prg
1	"user.page 110.1"	prg
1	"user.page 111.1"	prg
7	"user.page 145.1"	prg
4	"user.page 146.1"	prg
6	"user.page 147.1"	prg

483 blocks free.

TORPET EDUC.C Disk 05

3	"load and run"	prg
4	"disk to tape"	prg

```
  6    "list-me"              prg
 70    "ponzo tutor-1"        prg
 67    "ponzo tutor-2"        prg
 74    "ponzo tutor-3"        prg
  1    "insert new tape1"     prg
 64    "ponzo tutor-4"        prg
 67    "ponzo tutor-5"        prg
 61    "ponzo tutor-6"        prg
 60    "ponzo tutor-7"        prg
  1    "insert new tape2"     prg
 76    "sprites tut-1"        prg
 31    "sprites tut-2"        prg
 62    "graphic tutor"        prg
  2    "smooth scroll"        prg
  2    "example sprite"       prg
  1    "move sprite"          prg
  1    "expanded sprite"      prg
  4    "making sprites"       prg
  5    "program chars"        prg
  2    "multicolor chars"     prg

  0 blocks free.
```

MISCELLANEOUS.C Disk 06

```
  3    "load and run"         prg
  4    "disk to tape"         prg
  9    "list-me"              prg
  3    "colour test"          prg
  8    "c-64 grapher.c"       prg
  2    "64 h-r plot m/l"      prg
 20    "tv satellites"        prg
 13    "billboard"            prg
 12    "bio-compat."          prg
 17    "bio-printer"          prg
 23    "turtle"               prg
 23    "base conv.alt"        prg
 25    "english grammar"      prg
 28    "solar system"         prg
 26    "math iq"              prg
 46    "balancing equ"        prg
 95    "the kanon.c"          prg
109    "bach fugue"           prg
 25    "entertainer.c"        prg
 23    "yesterday.c"          prg
 19    "bach duet.c"          prg
 14    "organ.c"              prg
 52    "piano.c"              prg
  9    "jstick doodle.c"      prg
 56 blocks free.
```

UTILITIES-64.C Disk 07

```
  3    "load and run"         prg
  4    "disk to tape"         prg
 11    "list-me"              prg
  3    "sprite boot.c"        prg
  2    "scroll.data.d"        prg
 42    "sprite editor.d"      prg
 29    "sprite-instr."        prg
 30    "char-instr."          prg
  4    "char boot.c"          prg
  1    "rotate.data.d"        prg
  9    "standard.set.d"       prg
 32    "char editor.d"        prg
 23    "supermon-instr."      prg
 10    "supermon loader"      prg
 10    "copy-all.c"           prg
 13    "disk cat. 64"         prg
  1    "boot tiny aid"        prg
  5    "tiny aid $c066"       prg
  3    "base.c"               prg
 15    "disk check"           prg
  4    "change disk"          prg
  7    "disk name (r)"        prg
 11    "scopy.64"             prg
 15    "disassembler.c"       prg
 12    "lister 2.c"           prg
  9    "lockdisk64"           prg
  6    "prg function.c"       prg
 12    "superkey.c"           prg
 14    "block modifier.c"     prg
  7    "disk tidier.z"        prg
 18    "terminal.64.2.c"      prg
  7    "term.64.c"            prg
 28    "1541 backup.c"        prg
 17    "wedge-64-$9000.c"     prg
 17    "wedge-64-$c000.c"     prg
 17    "wedge-64-$7000.c"     prg
 17    "wedge-64-$8000.c"     prg

196 blocks free.
```

GAMES.V Disk 08

```
  3    "load and run"         prg
  4    "disk to tape"         prg
 12    "list-me"              prg
 10    "dr dementia in.v"     prg
 11    "dr dementia.v"        prg
 16    "car race(t)3k.v"      prg
 11    "shooter joy"          prg
  4    "steal money-inst"     prg
  9    "steal money"          prg
 13    "slo vicman keyb"      prg
 12    "the helicopter"       prg
 11    "heli. part 2."        prg
  6    "fort. hunt.inst."     prg
 11    "fortune hunter"       prg
 14    "mineslide"            prg
 12    "zarzon base"          prg
 14    "marston city"         prg
 14    "othello"              prg
  2    "artill-inst"          prg
 13    "artillery"            prg
  2    "rugby-inst"           prg
  9    "rugby"                prg
 12    "dragon maze"          prg
  3    "memory-inst"          prg
```

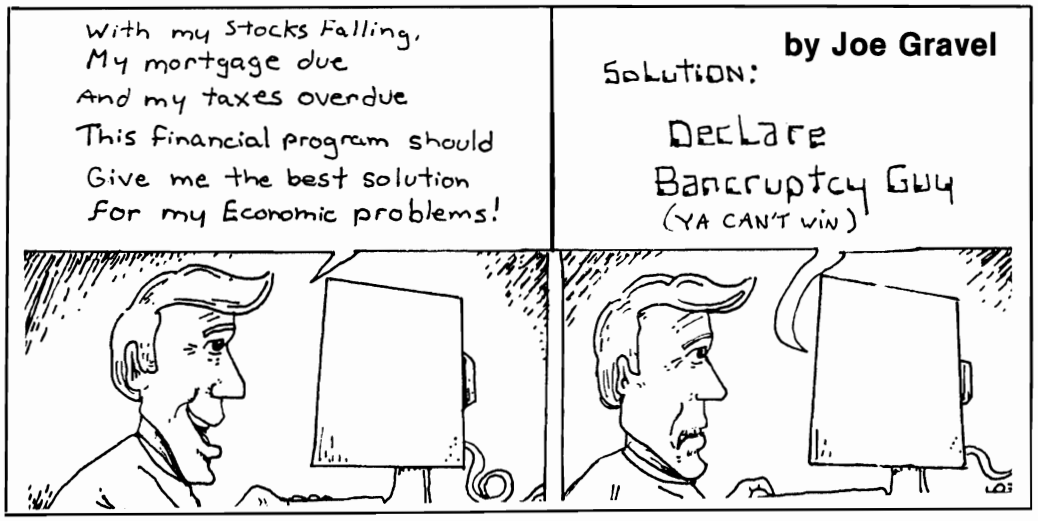
```
 9      "memory"                 prg
 2      "schuifspel-inst"        prg
 8      "schuifspel"             prg
 2      "firing tank-inst"       prg
11      "firing tank"            prg
 2      "ping pong-inst"         prg
10      "ping pong"              prg
 2      "zapem-inst"             prg
11      "zapem"                  prg
11      "maze-chase"             prg
12      "frogrun.v"              prg
12      "tron.v"                 prg
 6      "astroglad inst.v"       prg
 8      "astrogladiator.v"       prg
330 blocks free.
```

EDUCA/SIMULA.V Disk 09

```
 3      "load and run"           prg
 4      "disk to tape"           prg
 8      "list-me"                prg
 8      "speed reading.v"        prg
 6      "long division.v"        prg
 9      "globe"                  prg
19      "bilingual spell"        prg
19      "cryptograms"            prg
13      "factors"                prg
13      "french sentences"       prg
14      "v chinese c'book"       prg
13      "reading"                prg
31      "vicab1 8k.v"            prg
22      "vicab2 8k.v"            prg
19      "vicab3 8k.v"            prg
17      "vicab4 8k.v"            prg
19      "vicab5 8k.v"            prg
 4      "v drum mania"           prg
 6      "vic vic"                prg
12      "usa song"               prg
11      "graphics + sound"       prg
11      "sounds"                 prg
11      "demo"                   prg
 8      "piano"                  prg
 8      "over the r'bow.v"       prg
 6      "merry vic-mas"          prg
11      "bumblebee"              prg

339 blocks free.
```

MIS/UTILITIES.V Disk 10

```
 3      "load and run"           prg
 4      "disk to tape"           prg
 7      "list-me"                prg
11      "vic graph plot"         prg
10      "calculate base.v"       prg
 5      "metric convert"         prg
17      "vic checkbook"          prg
12      "esp test"               prg
11      "finance"                prg
19      "vic triangulator"       prg
 8      "dates"                  prg
12      "vic mail"               prg
19      "wizzacalc"              prg
 9      "term 5k inst.v"         prg
 3      "v-term 5k"              prg
10      "super vicmon2"          prg
 6      "function key.v"         prg
 1      "vic aid4.rel.v"         prg
 8      "vicword"                prg
 4      "joystick test"          prg
 7      "vic tape index"         prg
 9      "vic g.i.r.-inst"        prg
 4      "vic g.i.r."             prg
 7      "programmable char"      prg
14      "vic pilot 3k.v"         prg
17      "micromon@$0e003k"       prg
17      "micromon@$30008k"       prg

410 blocks free.
```



Descriptions of The Programs on The 10 Best Disks

By Darrin McGugan, Shelburne, Ont.

GENERAL

The following programs are located on all ten disks, except for the PET EMULATOR, which is on the C64 disks only.

'LOAD AND RUN' — This program gets a directory of the disk or disks in the disk drive. It then displays the list of the first 9 programs and asks you to enter the number of the program or to enter 9 to see more of the directory. When you enter the program number, it is automatically loaded for you.

'DISK TO TAPE' — This program loads programs one by one from the disk in the drive, then saves them one by one onto a tape. Just run the program, making sure that the disk you wish to be copied is in the drive and the blank tape is in the cassette recorder, and make sure it's re-wound. A message will come on the screen requesting you to press play and record. Now just sit back and relax until the tape is finished. If you haven't gotten all of the programs on it, then put in a new tape or flip it over.

'LIST-ME' — This is a list-me file for this disk. Just load the program and then list it. The title of the program wll be in CAPS or in quotes, and it will be followed by a short description of the program.

'PET EMULATOR' — This program makes it so that certain PET programs can work on the Commodore 64. All you have to do is load the program, then run it. After a moment, the screen will go grey. Now you can run certain PET programs on it. (NOTE: The run/stop-restore will not work when emulator is in.)

GAMES #1.C Disk 01

'ELIZA' — This is an interactive computer game with no graphics, just text. The computer (Eliza) asks you and answers your questions with short phrases or sentences. No matter what you ask her, she will come up with an answer. A change of pace from the arcade-type games. This is a PET emulator game.

'BATTLESHIPS' — This is a game of strategy and luck. Play against the computer. You must try guessing the locations of the opponent's five ships on a grid. You get one shot at the computer's ships, then the computer gets one shot at yours. It takes 3, 4, or 5 hits to sink each of the five ships. The one who sinks all five ships first wins. This game is very similar to the game by Milton Bradley. NOTE: This is a PET emulator game.

'LABYRINTH' — This is a maze game in which you must find your way from the entrance to the exit. You enter the labyrinth size (up to 20 by 20) and a labyrinth is designed for you. It is then displayed for you for a moment (length of time it is displayed depends on the difficulty level you have chosen). You must get to the far end of this 3-dimensional maze in as few moves as possible. Use the 'F' key to move forward one block, the 'C' key to move 90 degrees left one block, and the 'R' key to move right 90 degrees one block. Good use of graphics characters.

'SOLITAIRE' — Play solitaire on the computer. You select the version of solitaire that you would like to play and then see if you can win.

'OTHELLO' — Play Othello against the computer. At the start of the game, both the player and the computer have two pieces each located in the centre of the 8x8 grid. The object is to end up with the majority of the pieces on the board being your color. When it is your turn, you place your next piece on the board by entering co-ordinates (B7, A2, etc.). You place your pieces adjacent to the computer and then all its pieces between yours are turned to your colors. Good strategy game.

'CHECKERS' — A game of checkers using co-ordinates to make your moves. Not bad if you like checkers, but the computer always moves first, so I won't even play it because I'm a real suck.

'FLIGHT SIMULATOR' — This program is a flight simulator which displays a simplified view of a cockpit, including heading, altitude, speed, etc. You control the per cent of thrust (0-100%), the elevators and ailerons per cent (-100 to 100%), to make the plane do a 360-degree turn, then land on the runway without crashing. Takes a bit of practice. Plane can stall if you're not careful.

'PRO FOOTBALL' — A game of football using all text and no graphics. Entering 99 will result in the playbook being displayed. Just enter the number that is beside the move which you would

like to do. It uses American football rules. It could be almost close to being pretty near, just about slightly thrilling to a football enthusiast.

'STOCK MARKET' — Play the stock market with up to four players and/or the PET. Buy and sell stock in oil, gold, silver, industry and computers, trying to beat your opponents. The first one to make a million dollars is the winner. All stocks start at $1.00 each and then randomly go up or down. When a stock reaches $2.00, there is a stock split. This is a good game and may take a while to finish.

'CARD SNAP' — A game of snap on the computer. Two cards are displayed on the screen simultaneously, and if they match you must hit the space bar as quickly as possible. The one who gets the most snaps in wins. Game has several levels. The higher the level, the faster the computer is. Can you beat the computer at the higher levels? Fat chance! It has nice-looking cards.

'YAHTZEE' — This game is very similar to the real game of Yahtzee. It's a dice game played against the computer, in which you roll five dice (actually the computer does it for you) and enter the score in the appropriate spot. The one with the highest score wins. This is a game of chance. You'll like it if you like playing Yahtzee. You don't have to worry about losing the dice, and the computer doesn't cheat, but then, neither can you!

'BRIDGE BID TRAIN' — This program is supposed to train you to win at bridge. It assumes you already know something about bridge before you start. I don't know how to play bridge, so it didn't do anything for me. It is based on a book called 'Winning Contract Bridge Complete', by Edgar Kaplan.

'WET PAINT.C' — The object of this game is to paint more (and therefore get paid more) than your computer-controlled opponents. You will receive one cent per inch of painting you do. If you hit any of the wet paint that the others have painted your brush will blow up. The better you do the harder it gets because the boss will hire an increasing number of other painters.

'GOLF' — A game of golf using only text and no graphics. It's played on an 18-hole course at the Commodore country club. The user is even allowed a handicap, but he also has one of five different difficulties (putting, poor distance, etc.). You get to choose the club you want and the % of full swing, trying to get below par.

'VIPER.C' — Use the joystick to make a snake move around the screen running over the asterisks. Before you start playing the game you have the choice of many different speeds and of several different screens. Running into a wall or your own tail will result in destruction.

'GRANDPRIX.C' — Control your car in this race avoiding all barriers and blockades and see how fast you can do the race.

'PIRATE ADVEN' — This is another adventure game using text instead of graphics. In this adventure you get to build a ship and sail to other islands, among other things.

'BOWLING' — Use the keyboard to release the bowling ball at the correct time in order to knock down the bowling pins. See how good a score you can get.

'DOG.STAR.ADVEN' — In this space age adventure game you have landed on the Death Star and must search for and rescue Princess Leia from the clutches of Darth Vader.

'QUERK.C' — Use the keyboard to move your man around the maze avoiding all the aliens that are following you. Eat all the dots that you can before all three of your men are killed.

GAMES #2.C Disk 02

'MOTORCYCLE' — You set the speed of your motorcycle and the angle of the ramp to try to jump the buses. You choose how many buses you want to jump and, for each successful jump you make, another bus is added. When you crash, a list of your injuries is posted. You're likely to break a few ribs, an arm, a leg or two and both your noses. not to mention wrecking your head. Evil Knievel obviously used this program to calculate his jumps.

'HANGMAN 1' — A hangman game where you must pick a topic from the list, then try to guess the secret word the computer has picked by guessing one letter at a time. Each time you guess a letter, your friend (who has committed a nasty thing) moves up a step further towards the noose. Guessing the secret word makes the Judge happy, and he lets your friend off. A lot like real justice!

'3D TIC TAC TOE' — If you're a tic tac toe fan, this is for you. A very complex version of tic tac toe played against the computer on three different levels. Can be temporarily amusing.

'CONCENTRATION' — Enter coordinates of cards on the grid. The cards will flip over and each will have one of a number of symbols. If they match, then those two are taken off the screen. Keep trying different combinations until all are gone. The fewer moves it is done in the better your score. Can be nerve-racking. This game was the cause of my first of seven nervous breakdowns.

'BASKETBALL' — You are the captain and playmaker of this imaginary basketball team. This is a game of basketball against the computer, using text and no graphics. Kinda sorta boring at times but, if you're a basketball freak or just a plain ordinary freak, then you might like it a little.

'BLOCKADE' — You're a really naughty person who is trying to escape from a maximum security prison guarded by incredibly idiotic robots. Try not bumping into electric fences because they hurt. The object of the game is to get as many points as you can (by collecting food and supply bags) without getting caught by the robots. This game also has a bonus round for anyone who makes it that far.

'STAR WARS' — You are the pilot of an X-wing fighter, and are leading the attack against the death star in the name of the Alliance. The object is to destroy as many of the enemy TIE fighters as you can before you run out of fuel. There are three levels of play in this game. I don't know why everyone is always picking on poor Darth Vadar.

'SUPER STARTREK' — Move around the galaxy zapping aliens, using any of the nine commands. The galaxy is divided into an 8x8 quadrant grid, and each quadrant is further divided into an 8x8 sector grid. The object is to seek out and destroy all the Klingons using long and short-range sensor scans, photon torpedos, phasors, shields, etc. A good space game that even Spock likes.

'DEEP SPACE' — This space game is a tactical simulation of ship-to-ship combat in deep space. You are one of a group of captains assigned to patrol a section of border against hostile aliens. You get to select the type of vessel you use. Another okay space game.

'C.C. STARWARS INST.' — This program gives detailed instructions for the program 'C.C. STARWARS'.

'C.C. STARWARS' — You are the captain of the Millenium Falcon. You and your passengers (Luke, Obi-wan and the boys) have already rescued Princess Leia, but must now get back to the Rebel Forces Base. You must kill Darth Vader in order to make it back alive. This game is all text and no graphics, but is interesting to play.

'STARTREK' — This space game is very, very, very, very, very similar to 'SUPER STARTREK', but is in a simpler version. This one is done by Jim Butterfield.

'TOKER' — You are in control of a drug-crazed maniac.Use the keyboard to control the druggy's intake of the smoke from the pipe. Toke too hard and he coughs. Toke too soft and the pipe goes out, requiring another match to be lit. The object is to finish the contents of the pipe using the least number of matches you can.

'BILLIARDS' — The object of this game is to direct the cue ball at such an angle that it hits a combination of the three cushions before hitting both balls. Various points are given for various combinations. Angles of shots can be anywhere from 0 degrees to 359 degrees. This is a one- or two-player game.

'PET NUC PWR PLNT' — This program simulates the operation of a nuclear power reactor. The object is to operate the plant at as high a power output as possible, without causing a reactor meltdown.

'SORCERERS CASTLE' — An adventure game in which you use one or two word commands to move around. When you get to the castle door you'll probably find that it's locked. Now you'll have to go back and find the door. Chances are, you will likely spend most of your time lost in the forest.

'WIZARD'S CASTLE' — Another adventure game but this one is dungeon style. You have the choice of being a hobbit, dwarf, elf, etc., and you also get characteristics such as intelligence and strength. Before starting your quest you get to choose from a variety of weapons.

'RATRUN' — This is a three-dimensional maze game in which you use the joystick to move around through the maze. The object is to search for the cheese and eat it before it gets mouldy. Try to find it as quickly as possible.

'1000 MILES' — This game is a computer simulated version of a card game (Mille Bournes). You play against the computer picking random cards as you go. These cards include things like fuel and spare tires. The one who makes it one thousand miles first is the winner.

'STATES & CAPITALS' — This game tests your knowledge of the states and their capitals. You have the choice between fill-in-the-blanks

questions or multiple-choice ones.

'POWER INSTR.' — This program gives instructions for the game 'POWER PLANT'

'POWER PLANT' — This game is a simulation of a powerplant. See 'POWER INSTR.' for instructions.

GAMES #3.C Disk 03

'GRUNDY TOWERS' — A murder has been committed at the Grundy Towers and your job is to find the murderer. Check everyone's alibis closely: the killer is sure to give himself/herself away. Hint: nobody ever suspects the real murderer.

'CLUE' — You have been assigned to solve the murder of Lord Croker who was murdered at Blectchly Hall between one and nine P.M. You must solve the murder by questioning the five suspects as to where they were at the time of the murder. The suspects are not totally honest; they may lie to you. The case is solved when you successfully accuse the killer stating when and where he/she did it. False accusations make your career suffer.

'PINK PANTHER' — You are the assistant to the Great Inspector Clouzot. You must figure out who the murderer is. There are six suspects altogether but only one is the killer. The Inspector will ask all the questions. When you think that you have figured out everything then you must challenge the Inspector. You have the choice of five different difficulty levels before you start.

'QUEST3.0'—This is an adventure game. You were walking through the woods, and you came across the entrance to a cave that was covered with brush. People say that, many years ago, a Pirate hid his treasure in the woods, but no one has ever found it. It might still be there and if you're lucky then you might find it.

'LOST DUTCH GOLD' — You are searching for the Dutchman's Gold. You will be guided through this adventure by the ghost of Backpack Sam. He understands two word commands and you can ask for help if you get stuck. You travel around the countryside picking up items along your way to the gold.

'NICHE' — This is an ecological game. Niche refers to all of the ecological variables which relate to a given organism — its habitat, living space, and role in the ecosystem. It is your job to fit the selected organism into its niche. The goal is to maximize the size of the population by providing ideal conditions for growth.

'NEW WATER' — This is a water resource management game. You must make the correct decisions to insure that the village survives. Your starting population is two thousand and each acre of land under irrigation will support ten people (unirrigated land will only support about two).

'EASY DUNGEON' — This is a dungeon-style adventure game. Use two word commands to move around collecting treasures.

'ADVENTURE' — This is another adventure game in which you must find the treasures and place them in their correct location.

'MONOPOLE' — This is a game of monoply on the computer. Up to four players can play. You can roll the dice or make a play. When you land on a property a title deed of that property is displayed and gives you the option to buy if it is not already owned. The winner is the one who makes everyone else go broke.

'SPACESHOOTER' — This is a space game in which you use the joystick to move the sights onto the enemy, then fire. Try to get as high a score as you can before your time is up.

'SUPERTREK/16KNR' — You're in charge of the USS Enterprise and must zip all over the galaxy and kill all the Klingons you can find because they have invaded the galaxy and will soon invade the Federation H.Q. on Stardate 3128. You have 28 days to do it in. This game is a mixture of text and graphics.

'OSC LANDER' — In this game you must land a spacecraft on a lunar surface. You must do this by using the gauges provided. The readings on the gauges are; velocity in metres/sec, height in metres, remaining fuel in cubic metres, and elapsed time in seconds. If the velocity is in reversed field then you are going up, otherwise you're going down. Don't run out of fuel, though, because you will definitely crash.

'KENO' — This is a computer-simulated game of Keno. In the game there is a board with the numbers one through eighty on it. You select from one to fifteen numbers to play, then the computer picks twenty numbers at random and prints them on the board. It then sees how many numbers match your numbers and prints the results. You start with whatever amount of money you wish then try to win more.

'HORSE RACE' — In this game of chance the computer runs a horse race for you. Any number of players can bet on the results. You can choose the horse of your choice and whether you think it

will win, place, or show. The odds and names of the horses are posted for you to see.

'DRAW POKER' — This is a simulated game of Draw Poker. The ante is five dollars and there is no limit on bets. Try to see if you can out-play the computer at poker and win all his money.

'BLACK JACK1' — Play blackjack against the computer and see how much money you can win (or lose).

'ROULETTE' — This is a simulated game of roulette. Place your bet on the number that you think will win. The computer will then spin the wheel and if your number comes up then you're a winner.

'SLOTS JACKPOT' — This is a slot machine game which is pure chance. You pay a dollar every time you play and could win various amounts depending on the combination of symbols that appear on the screen.

'LEMONADE.C' — You are the manager of a lemonade stand. You must correctly spend your money on supplies, advertising, etc. in order to make a profit.

TORPET MANUAL.C Disk 04

'REF.PAGE 20.1' through to **'REF.PAGE 371.1'** — These programs are all of the sample programs that are in the Commodore 64 Reference Guide. Instead of typing out these sample programs all you have to do is load the program off this disk which has the corresponding page number. It's a whole lot quicker and easier than typing it out yourself.

'USER.PAGE 43.1' through to **'USER.PAGE 147.1'** — same as the above programs only it's for the Commodore 64 User's Guide.

TORPET EDUCATION.C Disk 05

'PONZO TUTOR-1' through to **'PONZO TUTOR-7'** — These programs teach the user, in a step-by-step format, all about computer programming. They let the user experiment with short programs, then continue on teaching. They progress from print statements and for-next loops to machine language. If he wants to go back a page or two, he can. When he gets done one set of lessons, he can then load the next set and then the next, until he has completed all seven.

'SPRITES TUT-1' and **'SPRITES TUT-2'** — These programs are similar to the 'BASIC TUTOR'S' but they are all about sprites (how to make them and how to use them).

'GRAPHIC TUTOR' — This program is similar to the other tutor programs but this one is about hi-resolution graphics and how to use them.

'SMOOTH SCROLL' — This program puts the Commodore 64 into a twenty-four line screen instead of twenty-five. This makes it so that a very smooth scroll can be done.

'EXAMPLE SPRITE' — This program gives an example of a sprite so the user can incorporate some of these ideas.

'MOVE SPRITE' — This program gives a demonstration of how to move the sprite you have created.

'EXPANDED SPRITE' — This program shows you how you can expand your sprite's height and/or length.

'MAKING SPRITES' — This program generates the data for the sprite you want.

'PROGRAM CHARS' — This program allows you to design your own characters. You can define a single key or the whole keyboard. You can save your newly-designed characters for later use.

'MULTICOLOR CHARS' — This program gives an example of how the computer can use multiple colored characters.

MISCELLANEOUS.C Disk 06

'COLOR TEST' — This program displays all sixteen of the computer's colors and their names so that you can adjust the tint, color, brightness and the contrast knobs to get the color to suit you.

'C-64 GRAPHER C' — This program plots hi-resolution graphs for the formula you enter.

'64 H-R PLOT M/L' — This program is machine language loaded by 'C-64 GRAPHER C'. Do not attempt to load it yourself; it won't do you any good.

'TV SATELLITES' — This program calculates the position (expressed as the compass bearing and altitude) of a satellite which has geostationary orbit. Most television and radio satellites are geostationary. You can choose from the list of satellites given or you can input the coordinates of other satellites for which you know the com-

pass bearing and altitude.

'BILLBOARD' — This program displays short messages on the computer screen. Up to three lines of up to six characters can be displayed. Only the letters between A and Z, and the numbers from 0 to 9 can be used.

'BIO-COMPAT' — You enter the birthdates of two people and the computer gives a compatibility rating expressed as a percent. It breaks it down into three categories: physical compatibility, emotional compatibility, and intellectual compatibility.

'BIO-PRINTER' — This program plots biorhythym charts. You must enter you date of birth, the starting date of the biorhythym chart, and the number of days you want it to run. The chart will then be charted on the screen for you to see.

'TURTLE' — This program allows you to draw pictures on the screen using simple commands. A line like this: U4-L5-D4-R5: would draw a 4X5 square on the screen.

'BASE CONV.ALT' — This program converts numbers from one base of numbers to another base. It converts in the following bases: binary, decimal, hexadecimal and Roman numerals.

'ENGLISH GRAMMAR' — Teaches and tests your English grammar. You must identify the part of speech of each of the given words.

'SOLAR SYSTEM' — This program informs you of the various planets in our solar system and their characteristics. It also gives a graphic demo of the planets' orbits.

'MATH IQ' — You are given fifteen mathematical questions to answer in twenty minutes. When you have done all the questions (or all that you can do), the computer will give you a mark and a mathematical IQ rating depending on how well you've done.

'BALANCING EQU' — Assists you in solving chemical equations. It also tests you on them.

'THE KANON.C' — This program simulates music.

'BACH FUGUE' — This program simulates some music of Bach.

'ENTERTAINER' — This program is also a musical simulation.

'YESTERDAY' — This program plays a song of the Beatles.

'BACH DEUT.C' — This program plays another one of Bach's pieces.

'ORGAN.C' — This program displays the keyboard of a organ. Just hit the key that corresponds to the organ note that you want.

'PIANO.C' — This program is similar to the program 'ORGAN.C' but this one plays piano music.

'JSTICK DOODLE.C' — Put your joystick in port #2 and draw pictures on the hi-resolution screen. Holding down the firebutton will lift the drawing pen. To clear the screen just press the F1 button. Pressing the F3, F5, F7 keys will change the color (border color, background color and line color).

UTILITIES.C Disk 07

'SPRITE-INSTR.' — This program gives instructions (on the screen or on paper) for the programs 'SCROLL.DATA.D' and 'SPRITE EDITOR.D'.

'SPRITE BOOT.C' — Load and run this program and it will load the following two programs for you.

'SCROLL.DATA.D' — Instructions for this program are found in 'SPRITE-INSTR.'.

'SPRITE EDITOR.D' — Instructions for this program are found in 'SPRITE-INSTR.'.

'CHAR INSTR.' — This program gives instructions for the programs 'ROTATE.DATA.D', 'STANDARD.SET.D' and 'CHAR EDITOR.D'.

'CHAR BOOT.C' — Load and run this program and it will load the following three programs for you.

'ROTATE.DATA.D' — Instructions for this program are found in 'CHAR INSTR.'.

'STANDARD.SET.D' — Instructions for this program are found in 'CHAR INSTR.'.

'CHAR EDITOR.D' — Instructions for this program are found in 'CHAR INSTR.'.

'SUPERMON-INSTR.' — This program gives you detailed instructions on how to use SUPERMON.

'SUPERMON LOADER' — This program loads the SUPERMON for you to use. See 'SUPERMON-INSTR.' to find out how to use this program correctly.

'TERMINAL.64.2.C' — This program is for use with a modem. A menu is displayed for you to choose from. You can have it in terminal mode, you can receive programs, transmit programs (be careful when doing this), open disk files, print disk files, change the screen colors, or quit if you wish.

'TERM.64.C' — This program is loaded by the above program and you should not attempt to load this program yourself.

'1541 BACKUP.C' — This program allows you to make backups of your disks using a 1541 disk drive. Just load this program and run it and you will be directed as to what to do. You must keep on switching the disks that are in the disk drive. This program is slow but is a lot cheaper than buying a dual-disk drive.

'COPY-ALL.C' — This is a copying utility for copying all types of files and requires two 1541 disk drives. It is easy to use (just follow the screen prompts) but is slow.

'BOOT TINY AID' — This is a small programmer's utility that adds five commands to the BASIC language:

FIND.search string. — prints all occurrences of 'search string' (you may use quotes as the delimiters, or anything else that you want to).

CHANGE.search string.replacement string. — changes all occurrences of 'search string' to 'replacement string' — see FIND command description.

DELETE first line number — last line number — used for multi-line deletions (eg. DELETE 10-100 will delete all lines between 10 and 100).

NUMBER first line number, increment — renumbers BASIC programs.

KILL — disables the TINY AID.

'TINY AID $C066' — Machine language loaded by 'BOOT TINY AID'.

'DISSEMBLER.C' — This program allows you to disassemble any section of the Commodore 64's memory to your printer or the screen.

'LISTER 2.C' — This program allows you to list a program from your disk to your printer or the screen.

'LOCKDISK 64' — This program allows you to protect your programs. Now your program will automatically run and you cannot get out of the program unless you turn the computer off and on again.

'PRG FUNCTION.C' — This program allows you to assign words and/or values to each of the eight function keys. They will remain like that until power down or until the RUN/STOP RESTORE is pressed.

'WEDGE-64-$C0000'** — This is a wedge program for the Commodore 64. Some of the commands are: ►adjust; ►auto; ►cold; ►color; ►del; ►ds; ►help; ►hex; ►hunt; ►look; ►mem; ►merge; ►n; ►off; ►renum; ►save; ►start; ►send; ►$ and ►/. To load this program : LOAD **'WEDGE-64-$c000',8,1 then enter SYS 12*4096 to activate. Article on page 264 gives complete instructions.

'SUPERKEY.C' — This program gives your keyboard BASIC keywords on each letter. To get a keyword, Press 'F1' followed by the character of your choice. 'F3' will delete a keyword. The 'F5' key will list for you and the 'F7' key will run for you.

'BLOCK MODIFIER.C' — This program allows you to load a block into the memory, examine it, change it if you wish and save it back to the disk. BE CAREFUL; if you're not careful you may ruin your disk.

'DISK TIDIER.Z' — This program allows you to go through a disk and scratch any unwanted files or programs. This program can save you time.

'DISK CHECK' — This is a disk utility program. It will display the BAM map of your disk and then it will give you three options:

1) check all files
2) check for bad spots
3) recover scratched file

Just choose your option and sit back while your computer goes to work.

'SCOPY.64' — This is a program copier. It uses a single disk drive and is a utility that any person who wants to copy programs should not be without. It will only copy program and sequential files. The instructions are on the screen and very simple to follow.

'DISK NAME (R)' — This program will rename your disk. You run the program, tell it what the new name for your disk is, and it will rename the disk. Just follow the prompts on the screen.

'CHANGE DISK' — This utility is used to change the disk device number, i.e., you have two drives

and you want one to become a '9'. Turn one drive on, load and run the program and voila! you have a disk with a device number of 9.

'DISK CAT. 64' — This program is used to print a catalog of your disk's contents to your printer.

'BASE.C' — This program converts numbers from one base to another base. Hexadecimal to decimal, etc.

GAMES.V Disk 08

'DR DEMENTIA IN.V' — This program gives the instructions for the game 'DR DEMENTIA.V'.

'DR DEMENTIA.V' — You have the choice of three different levels of play. You use the joystick to move your hic-dewey from one side of the screen to the other, shooting at the spikes to make them shorter and shorter until finally they are gone. Don't let them reach the bottom or you will die. Hint: leave the middle ones until the last.

'CAR RACE(T)3K.V' — Use the joystick to move your car around the race track trying to out-race the computer-controlled car. You even get to choose the name of your opponent from a list. The first one to make five laps of the track wins. Pushing down the fire button puts your car in high gear, making it faster than before. NOTE: This program requires a 3K expander.

'SHOOTER JOY' — This is a space game in which you must use the joystick to aim the sights on the various alien spacecraft, each of which are worth different points. You only have one minute to get as many points as you possibly can.

'STEAL MONEY-INST' — This program gives instructions for the game 'STEAL MONEY'.

'STEAL MONEY' — Use the keyboard to move the little man from one side of the screen to the other. When you get to the other side, pick up the money and go back. Sounds easy, but you must avoid being hit by the acid snow that is continuously falling and destroying your shelters.

'SLO VICMAN KEYB' — This is a version of the famous arcade game of PACMAN. You use the keyboard to move your man around, eating all the dots and avoiding all the ghosts. As soon as you eat one of the power dots, the ghosts change color and you can now eat them.

'THE HELICOPTER' — This program gives you instructions for the game 'HELI. PART .2'.

'HELI.PART.2' — Use the joystick to manoeuvre your helicopter over to the building blocks. Then use the fire button to pick it up. You must avoid all the bullets that are constantly coming while you try placing the blocks at various levels. The longer you stay alive the greater your score.

'FORT. HUNT.INST.' — This program gives instructions for the game 'FORTUNE HUNTER'.

'FORTUNE HUNTER' — Use the joystick to move the man across the screen to retrieve the money. You must get the money and return without tripping over the creature. Keep going across and getting the money until you get killed. The object of the game is to get as much money as you can.

'MINESLIDE' — Use the joystick to manoeuvre your man down the mineshaft collecting all of the money without falling down a mineshaft and getting yourself killed. When you've cleared the mine of all the cash, then take the elevator to the surface and drop it off. Now go back down and get some more. NOTE: This program requires a 3K expander.

'ZARZON BASE' — Use the A and D keys to move your missiles into the descending bombs before they land on your city, causing it to be obliterated. Use the S key to launch your next missile.

'MARSTON CITY' — Use the joystick to move the shooter at the bottom of the screen, shooting away the many shrubs, cacti and tumbleweeds, all of which have certain point values. You must also destroy the descending thing-a-mabob before it rams into you.

'OTHELLO' — Play othello against the computer and see if you can beat it. You are given the choice of color of your pieces, the option to go first and you can tell the computer not to try its best. The object of the game is to end up with the majority of pieces on the board. NOTE: This program requires a 3K expander.

'ARTILL-INST' — This program gives instructions for the game 'ARTILLERY'

'ARTILLERY' — You set the charge and the angle of a mortar and try to make it land on the enemy. You've got to make it go over the mountain. Use the F1 key to decrease the charge, the F3 key to increase the charge, the F5 key to decrease the angle and the F7 key to increase the angle.

'RUGBY-INST' — This program gives instructions

on how to play the game 'RUGBY'.

'RUGBY' - Use your joystick to move your man (the red dot) to the goal line at the top without getting tackled by the four opposing team members. Really fair — ha!

'DRAGON MAZE' — In this maze game you must try to make it to the far exit without getting killed by the monster. You have the decision on how wide the path will be. Good game but it is practically impossible to exit the maze without getting smucked by the meany monster.

'MEMORY-INST' — This program gives instructions for the game 'MEMORY'.

'MEMORY' — This is a game of memorization for one to four players. Enter the coordinates for two of the sixty-four squares on the screen. The computer will then show you the symbols that are under the squares and if they match you get the points for them. The object is to try and remember what symbols are under each of the coordinates. Points are given every time you get a match. The one with the most matches wins. NOTE: This program requires a 3K expander.

'SCHUIFSPEL-INST' — This program gives instructions for the game 'SCHUIFSPEL'.

'SCHUIFSPEL' — Try to get the numbers from one to fifteen arranged correctly on the sixteen-square grid. See 'SCHUIFSPEL-INST' for instructions.

'FIRING TANK-INST' — This program gives instructions for the game 'FIRING TANK'.

'FIRING TANK' — Move your tank around the screen shooting all the asterisks for maximum points. Use the '6' key to turn right, the '4' key to turn left, and the '5' key to fire.

'PING PONG-INST' — This program gives you instructions for the game 'PING PONG'.

'PING PONG' — See how high a score you can get by knocking away all the blocks on the screen. When you have finished one screen, another will appear. The paddle is moved using the two cursor keys. The higher the blocks are on the screen, the more points they are worth.

'ZAPEM-INST' — This program gives instructions for the game 'ZAPEM'.

'ZAPEM' — You choose a level of difficulty depending on your skill. The object of the game is to get as many points as you can by moving your man around the screen in eight directions, blasting all in sight.

'MAZE CHASE' — This is a maze game. You must move around the maze without getting yourself killed.

'TRON' — Use your joystick to move your motorcycle around the screen, filling up as much of it as you can without colliding with the three other motorcycles (or their trails) and without colliding with your own trail.

'ASTROGLAD INST.V' — Instructions for the game 'ASTROGLADIATOR.V'.

'ASTROGLADIATOR.V' — Use the keyboard to move your ship around the galaxy, shooting the enemy.

'FROGRUN.V' — Public domain version of the popular game FROGGER. Try to get the frog to safety.

EDUCATION — SIMULATION.V
Disk 09

'SPEED READING.V' — This program is designed to test and improve your reading speed. A short phrase or sentence is displayed for second or a fraction of a second and then you must correctly type in what you saw. If you are correct then it will go on to the next one but this one is displayed for an even shorter time. If you're wrong it will try the same phrase over and over until you get it right.

'LONG DIVISION.V' — This program is to test your long division. You get to choose the level of difficulty before any questions are asked. You just enter the numbers that you think are right; then the computer will confirm it. If it is right, then the next question will appear. If you are wrong you get to try again. You may stop at any time you wish and you will get a summary of how well you've done.

'GLOBE' — This program is a geography quiz. You must pick the correct answer out of the four answers given. You will be asked a total of twenty multiple choice questions on continents and oceans. At the end of the quiz you will be given a mark depending on how well you've done.

'BILINGUAL SPELL' — This is a Spanish-English program. You choose between English words or Spanish words. You will then be given a list of words to study and spell. The word is flashed on the screen for a moment and then removed. You must now correctly spell the word in order to move on to the next question. At the end of the

quiz you will be told how many words you got wrong and what they were. This way you can write down the words and study them.

'CRYPTOGRAMS' — In this game you can solve or make cryptograms. Crytograms are coded puzzles. NOTE: This program requires a 3K expander.

'FACTORS' — This program tests you on factors. There are ten different levels to try depending on how good you are at solving factors. When you answer a certain number of questions you will be given a mark and shown which ones you have done wrong.

'FRENCH SENTENCES' — This program is a French quiz. Before each set of questions that you will be asked there is a new set of rules. This program tests and supposedly improves your French.

'V CHINESE C'BOOK' — This program is on Chinese cooking. Choose a recipe (from the menu) that you think you might like and the computer will proceed to instruct you on how to cook that particular dish. It will tell you what ingredients you'll need, what equipment, and the number of people it will serve.

'READING' — This program is to test your reading comprehension. A short story on alcohol and its effects will be displayed for you to read; then, at the end of the story, you will be asked a number of questions referring to the story. You will then be given a mark.

'VICAB1 8K.V' — This program assists in building vocabulary for young children using word association illustrations. This has selective input, i.e., for a truck, only the letter 'T' will be accepted followed by R...U...C...K. It shows children about colors, about numbers and about different objects such as rockets and houses. NOTE: These five programs require an 8K expander.

'VICAB2 8K.V' — Same as above

'VICAB3 8K.V' — Same as above

'VICAB4 8K.V' — Same as above

'VICAB5 8K.V' — Same as above

'V DRUM MANIA' — This program simulates the sounds from a set of drums. Use the space bar for the bass, the F key for the snare, and the T, Y, J, M keys for the tom-toms. The only thing that you don't have is cymbals.

'VIC VIC' — This program is a demo of some of the VIC's graphics. It makes good use of the VIC's color and window capabilities.

'USA SONG' — This program is a demo. It displays a graphic picture of the United States flag and it plays their national anthem.

'GRAPHICS—SOUND' — This is a demo of the VIC's graphics and sound. It gives examples of animation, mazes, 3D effects, patterns, special effects and sound.

'SOUNDS' — This program will show you some of the sound capabilities of your VIC. Just enter the number of the sound effect (from the menu) you would like and then the computer will play it for you. Examples of these are: UFO's landing; red alerts; running feet; and computer mania.

'DEMO' — This demo tells the user about the VIC's specifications, abilities, periferals, and applications.

'PIANO' — This program lets you make music via the keyboard. Just hit any of the keys and see what sound comes out. This program would be better if a diagram of the keys and their notes was displayed on the screen.

'OVER THE R'BOW' — This program plays a song using the VIC's sound capabilities.

'MERRY VIC-MAS' — This program also plays a song plus it shows a Christmas tree complete with lights.

'BUMBLEBEE' — This one plays the fast-paced Bumblebee tune and displays the notes as they are played.

MISCELLANEOUS — UTILITIES
Disk 10

'CALCULATE BASEC.V' — This program will calculate binary, hex and decimal numbers for you. If you wish, you can have examples (as many as you want) before you start. Enter the base of the number and then the number itself. Now enter the base of the other number and the computer will calculate it for you.

'METRIC CONVERT' — This program converts measurements from imperial to metric. It will convert inches to centimeters, pounds to kilograms, quarts to litres, and miles to kilometres.

'VIC TRIANGULATOR' — You enter data about a triangle and the computer will find the angles of

the triangle and the length of its sides. This program solves only rectangular triangles. NOTE: This program requires a 3K expander.

'VIC MAIL' — This program lets you save and retrieve mailing information on tape. You can store the person's name, house number, street number, city, province, and miscellaneous information. When you like, you can update any of the information.

'WIZZACALC' — This program calculates grade averages. You determine the weight or hours for each grade. NOTE: This program requires a 3K expander.

'VIC GRAPHPLOT' — This does a medium resolution data analysis. You enter the names of the x and y axis then the data for each pair. When all data is entered the computer will draw a graph of it for you.

'VIC CHECKBOOK' — This program acts as a personal chequebook. It lets you save data on a cassette tape referring to the cheques you have written. You can save the whole year's data on a tape ready for easy retrieval. NOTE: This program requires a 3K expander.

'ESP TEST' — This program tests one variety of ESP — the ability to predict future events. The computer will select one of five symbols randomly but will not disclose it until you have recorded your prediction as to what it is. At the end of your multiple trails you will be told whether the amount of correct predictions is a result of chance or ESP.

'FINANCE' — This program does financial calculations. If you want to find one of the values (present value, number of periods, interest rate, payments made, value of investment, or unpaid balance) then just enter the numbers of the other five values. The computer will make the necessary calculations for you.

'DATES' — This program calculates days between dates and also gives the date of any given number of days after base date. Just enter the base date and then you'll have the choice of either calculating how many days another date is ahead or back of that date, or calculate the date of a number of days after the base date.

'VICTERM 5K-INST.V' — This program gives instructions on how to use the program 'VICTERM 5K.V'

'VICTERM 5K.V' — This program is designed to be used with RS 232 'cheap' modems. It may work on others though. This program will automatically switch to lower case when run. Use the F1 key to clear screen without affecting data. Use the F7 as a control key. You have about two seconds to hit any key from A to Z before the program goes back to receiving data from the other computer. Hit return a few times to sign on to the host computer. It's for use with the 5K unexpanded VIC only. The 'bell' from the host computer is picked up by this program so turn up your volume.

'VICWORD' — This program assigns keywords to a single key. Just load and run this program; then type SYS 6912 and give it a try. Just use the shift key with any key between A and Z and see what comes up. Examples of the key words are: shifted P equals POKE, shifted A equals PRINT, shifted E equals GET, etc.

'VIC TAPE INDEX' — This program lets you put a directory on your cassette tapes. It allows you to automatically load the desired program by entering the corresponding number of the program you wish from the directory menu. NOTE: When saving programs on the tape make sure the longest ones are at the end of the tape.

'VIC AID 4.REL.V' — This program is similar to TINY AID but is relocatable.

'VIC G.I.R.-INST' — This program gives instructions for the program 'VIC G.I.R.'.

'VIC G.I.R.' — See 'VIC G.I.R.-INST' for instructions.

'PROGRAMMABLE CHAR' — This program is a character editor. It allows you to redesign any or all of the characters on the keyboard. You can save your newly designed character set for later use. It also lets you see the numeric data for any of the old or new characters.

'FUNCTION KEY.V' — This program allows you to define the eight function keys on the VIC. You can make these a single character or as many as 256 characters in length. For example: F1 equals A; F2 equals B; F3 equals RUN; F4 equals LIST; F5 equals SUPERCALIFRAGILISTIKEXPIALIDOSHOUS; F6 equals TORPET; F7 equals IS and F8 could be left blank.

'VIC PILOT 3K.V' — This program allows you to use the Pilot language on the 3K VIC. If you enter an illegal command, the computer will inform you. A prior knowledge of Pilot is necessary to utilize this program. NOTE: This program requires a 3K expander.

'JOYSTICK TEST' — This program is designed to let you check and see if your joystick if func-

tioning properly. Run the program; then plug in your joystick and move it around. On the screen you will see the four letters (n,s,e,w) and an FB (firebutton). When you move your joystick to the top or north you should see a 1 under the n and when you move it left you should see a 1 under the letter w, etc. When the firebutton is pressed a 1 should appear under the FB. If any of the results are not correct, then your joystick is not functioning right.

'MICROMON@0E003K' — Refer to article on page 267 for instructions. NOTE: Use this program with a 3K expander.

'MICROMON@30008K' — Refer to article on page 267 for instructions. NOTE: Use this program with an 8K expander.

Disk Description Summary and Checklist

By Bruce Beach, Horning's Mills, Ont.

The information presented here is all in this book in various places in other forms but, because the amount of information available is so much that it is sometimes confusing to a new reader, it will be summarized here and presented in a different manner, so that you will have a checklist of what is described in this book and where it is available.

PUBLIC DOMAIN PROGRAMS

First and foremost, it is important to remember that all the programs on this checklist are in the public domain, which means you are welcome to copy them to or from friends free of charge. If you want to make copies of them and sell them, there is no law saying that you can't and, on the other hand, if you are willing to pay someone for them, they have done nothing wrong by selling them to you. Most people who copy public domain diskettes for others simply charge for the disk material, effort of copying, mailing and handling expenses, etc. About ten dollars per diskette for that service is usually considered a fair price. If you get 20 programs on a diskette for $10, you have paid 50 cents a program, which is a real bargain.

TAPE USERS

If you don't have a disk drive, you can get the education programs on tape from TPUG (see page 239 for further information), and they have many other programs available also. The education series is also available from Aurora Software (see page 239). All of the Diskettes from TORPET have a disk-to-tape file transfer program, so tape users who have a friend who has a disk drive can get them to conveniently make the transfer for them.

THE BEST PROGRAMS DISK

The Best Programs disk is a list of the programs whose long listings (of 20 lines or over) appear in this book, plus some others that were described but not listed because of lack of space. Since these programs did not completely fill the diskette, we added some of the best programs from the 10 best disks series. If you were going to get only one diskette, this is probably the best one to get, but it doesn't contain many games.

Some of these programs work on the C64, VIC-20 and PET (these are listed ALL in the directory on page 272), but some work only on the C64 or VIC-20.

MAKING PROGRAMS WORK ON THE PET

Because the diskette was developed on a C64, it is necessary when loading the programs in a PET to use the instructions on page 23 of this book.

THE 10 BEST DISKETTES

The 10 Best Diskettes are made up of public domain programs accumulated from several sources that have over 6,000 programs in total. These programs are described on each diskette in a 'List Me' file on that diskette. The programs are also described in more detail, along with directions on how to use the programs, in the article preceding this one.

Seven of the 10 Best Diskettes are for the Commodore 64, and three are for the VIC-20, but many of the VIC-20 programs may work (or work somewhat) on the Commodore 64. Sometimes, the VIC-20 programs contain different screen formats or special pokes that need to be modified for them to work on the 64.

The 64 programs are generally too long to work on

an unexpanded VIC. For that matter, some of the VIC programs will not work on an unexpanded VIC, and these are identified both in the "List Me Files" and in the preceding chapter.

THE EDUCATION SERIES

The Education Series of 58 diskettes will all work on a C64 or 32K PET, but not on a VIC because of space limitations. Most would need modifications to work even on an expanded VIC, because of differences in screen formatting.

The TORPET Best Programs

1. Diskette for the C64/VIC (Includes all the lengthy programs in this book plus many more). **The one diskette to get if you get only one diskette.**

10 Best TORPET Diskettes

1. Games#1.C	C64
2. Games#2.C	C64
3. Games#3.C	C64
4. Torpet Manual.C (The Programs in the Commodore Manual & User's Guide)	C64
*5. Torpet Educ.C (Teaches you how to program)	C64
6. Miscellaneous.C	C64
* 7. Utilities-64.C	C64
8. Games.V	VIC
9. Educa/Simula.V	VIC
*10. Misc/Utilities.V	VIC

***The Second Most Important Diskettes to Get!**

Education (E) Series C64/PET

1. AA — Administration
2. BA — Business
3. BB — Business
4. BC — Business
5. CA — Computer Science
6. CB — Computer Science
7. D1 — Commodore 64
8. D2 — Commodore 64
9. D3 — Commodore 64
10. EA — English
11. EB — English
12. EC — English
13. ED — English
14. EE — English
15. EF — English
16. EG — English
17. EH — English
18. EI — English
19. FA — French
20. GA — Games
21. GB — Games
22. GC — Games
23. GD — Games
24. GE — Games
25. RA — Geography
26. RB — Geography
27. RC — Geography
28. JA — Language
29. LA — Language and Problem Solving
30. LB — Logic and Problem Solving
31. LC — Logic and Problem Solving
32. LD — Logic and Problem Solving
33. MA — Mathematics
34. MB — Mathematics
35. MC — Mathematics
36. MD — Mathematics
37. ME — Mathematics
38. MF — Mathematics
39. MG — Mathematics
40. MH — Mathematics
41. MI — Mathematics
42. MJ — Mathematics
43. MK — Mathematics
44. ML — Mathematics
45. MM — Mathematics
46. MN — Mathematics
47. MO — Mathematics
48. NA — Music
49. PA — Physical and Health Education
50. SA — Science
51. SB — Science
52. SC — Science
53. SD — Science
54. SE — Science
55. SF — Science
56. SG — Science
57. TA — Technology
58. UA — Utilities

Any of these Diskettes are available from:

TORPET DISKETTES
Horning's Mills
Ontario. L0N 1J0
Canada

or

TORPET DISKETTES
1 Brinkman Ave.
Buffalo, N.Y.
14211, U.S.A.

Single Diskettes are $10 each (send cash, cheque of money order), or $350 for all 69 diskettes.

Maps

PAGE

The Commodore 64 Maps 292
Jim Butterfield, Toronto, Ont.
For the advanced programmer, these tell where all those locations are in the Commodore ROM.

Commodore 64 Maps 300
Don White, Ottawa, Ont.
More information about the 64 ROMs, with special emphasis on manipulating the Lo-Res and Hi-Res Screens and also on the 6581 SID synthesizer chip.

PET/VIC/64 Cross Reference Map 307
Mark Niggemann, Ames, Iowa
These maps can be a very major aid in converting old PET programs to the VIC or 64 (or any other combination).

The Commodore 64 Maps

By Jim Butterfield, Toronto, Ont.

HEX	DECIMAL	DESCRIPTION
0000	0	Chip directional register
0001	1	Chip I/O; memory & tape control
0003-0004	3-4	Float-Fixed vector
0005-0006	5-6	Fixed-Float vector
0007	7	Search character
0008	8	Scan-quotes flag
0009	9	TAB column save
000A	10	0 = LOAD, 1 = VERIFY
000B	11	Input buffer pointer/# subscript
000C	12	Default DIM flag
000D	13	Type: FF = string, 00 = numeric
000E	14	Type: 80 = integer, 00 = floating point
000F	15	DATA scan/LIST quote/memry flag
0010	16	Subscript/FNx flag
0011	17	0 = INPUT; $40 = GET; $98 = READ
0012	18	ATN sign/Comparison eval flag
0013	19	Current I/O prompt flag
0014-0015	20-21	Integer value
0016	22	Pointer: temporary strg stack
0017-0018	23-24	Last temp string vector
0019-0021	25-33	Stack for temporary strings
0022-0025	34-37	Utility pointer area
0026-002A	38-42	Product area for multiplication
002B-002C	43-44	Pointer: Start-of-Basic
002D-002E	45-46	Pointer: Start-of-Variables
002F-0030	47-48	Pointer: Start-of-Arrays
0031-0032	49-50	Pointer: End-of-Arrays
0033-0034	51-52	Pointer: String-storage (moving down)
0035-0036	53-54	Utility string pointer
0037-0038	55-56	Pointer: Limit-of-memory
0039-003A	57-58	Current Basic line number
003B-003C	59-60	Previous Basic line number
003D-003E	61-62	Pointer: Basic statement for CONT
003F-0040	63-64	Current DATA line number
0041-0042	65-66	Current DATA address
0043-0044	67-68	Input vector
0045-0046	69-70	Current variable name
0047-0048	71-72	Current variable address
0049-004A	73-74	Variable pointer for FOR/NEXT
004B-004C	75-76	Y-save; op-save; BASIC pointer save
004D	77	Comparison symbol accumulator
004E-0053	78-83	Misc work area, pointers, etc
0054-0056	84-86	Jump vector for functions
0057-0060	87-96	Misc numeric work area
0061	97	Accum#1: Exponent
0062-0065	98-101	Accum#1: Mantissa
0066	102	Accum#1: Sign
0067	103	Series evaluation constant pointer
0068	104	Accum#1 hi-order (overflow)
0069	105	Accum#2: Exponent, etc.
006A-006D	106-109	Accum #2: Mantissa
006E	110	Accum #2: Exponent, etc.
006F	111	Sign comparison, Acc#1 vs #2
0070	112	Accum#1 lo-order (rounding)
0071-0072	113-114	Cassette buff len/Series pointer
0073-008A	115-138	CHRGET subroutine; get Basic char
007A-007B	122-123	Basic pointer (within subrtn)
008B-008F	139-143	RND seed value
0090	144	Status word ST
0091	145	Keyswitch PIA: STOP and RVS flags
0092	146	Timing constant for tape
0093	147	Load = 0, Verify = 1
0094	148	Serial output: deferred char flag
0095	149	Serial deferred character
0096	150	Tape EOT received
0097	151	Register save
0098	152	How many open files
0099	153	Input device, normally 0
009A	154	Output CMD device, normally 3
009B	155	Tape character parity

HEX	DECIMAL	DESCRIPTION
009C	156	Byte-received flag
009D	157	Direct = $80/RUN = 0 output control
009E	158	Tp Pass 1 error log/char buffer
009F	159	Tp Pass 2 err log corrected
00A0-00A2	160-162	Jiffy Clock HML
00A3	163	Serial bit count/EOI flag
00A4	164	Cycle count
00A5	165	Countdown,tape write/bit count
00A6	166	Tape buffer pointer
00A7	167	Tp Wrt ldr count/Rd pass/inbit
00A8	168	Tp Wrt new byte/Rd error/inbit cnt
00A9	169	Wrt start bit/Rd bit err/stbit
00AA	170	Tp Scan;Cnt;Ld;End/byte assy
00AB	171	Wr lead length/Rd checksum/parity
00AC-00AD	172-173	Pointer: tape bufr, scrolling
00AE-00AF	174-175	Tape end adds/End of program
00B0-00B1	176-177	Tape timing constants
00B2-00B3	178-179	Pntr: start of tape buffer
00B4	180	1 = Tp timer enabled; bit count
00B5	181	Tp EOT/RS232 next bit to send
00B6	182	Read character error/outbyte buf
00B7	183	# characters in file name
00B8	184	Current logical file
00B9	185	Current secndy address
00BA	186	Current device
00BB-00BC	187-188	Pointer to file name
00BD	189	Wr shift word/Rd input char
00BE	190	# blocks remaining to Wr/Rd
00BF	191	Serial word buffer
00C0	192	Tape motor interlock
00C1-00C2	193-194	I/O start address
00C3-00C4	195-196	Kernel setup pointer
00C5	197	Last key pressed
00C6	198	# chars in keybd buffer
00C7	199	Screen reverse flag
00C8	200	End-of-line for input pointer
00C9-00CA	201-202	Input cursor log (row, column)
00CB	203	Which key: 64 if no key
00CC	204	0 = flash cursor
00CD	205	Cursor timing countdown
00CE	206	Character under cursor
00CF	207	Cursor in blink phase
00D0	208	Input from screen/from keyboard
00D1-00D2	209-210	Pointer to screen line
00D3	211	Position of cursor on above line
00D4	212	0 = direct cursor, else programmed
00D5	213	Current screen line length
00D6	214	Row where cursor lives
00D7	215	Last inkey/checksum/buffer
00D8	216	# of INSERTs outstanding
00D9-00F2	217-242	Screen line link table
00F3-00F4	243-244	Screen color pointer
00F5-00F6	245-246	Keyboard pointer
00F7-00F8	247-248	RS-232 Rcv pntr
00F9-00FA	249-250	RS-232 Tx pntr
00FF-010A	256-266	Floating to ASCII work area
0100-103E	256-318	Tape error log
0100-01FF	256-511	Processor stack area
0200-0258	512-600	Basic input buffer
0259-0262	601-610	Logical file table
0263-026C	611-620	Device # table
026D-0276	621-630	Sec Adds table
0277-0280	631-640	Keybd buffer
0281-0282	641-642	Start of Basic Memory
0283-0284	643-644	Top of Basic Memory
0285	645	Serial bus timeout flag
0286	646	Current color code
0287	647	Color under cursor
0288	648	Screen memory page
0289	649	Max size of keybd buffer
028A	650	Repeat all keys
028B	651	Repeat speed counter
028C	652	Repeat delay counter
028D	653	Keyboard Shift/Control flag
028E	654	Last shift pattern
028F-0290	655-656	Keyboard table setup pointer
0291	657	keyboard shift mode
0292	658	0 = scroll enable

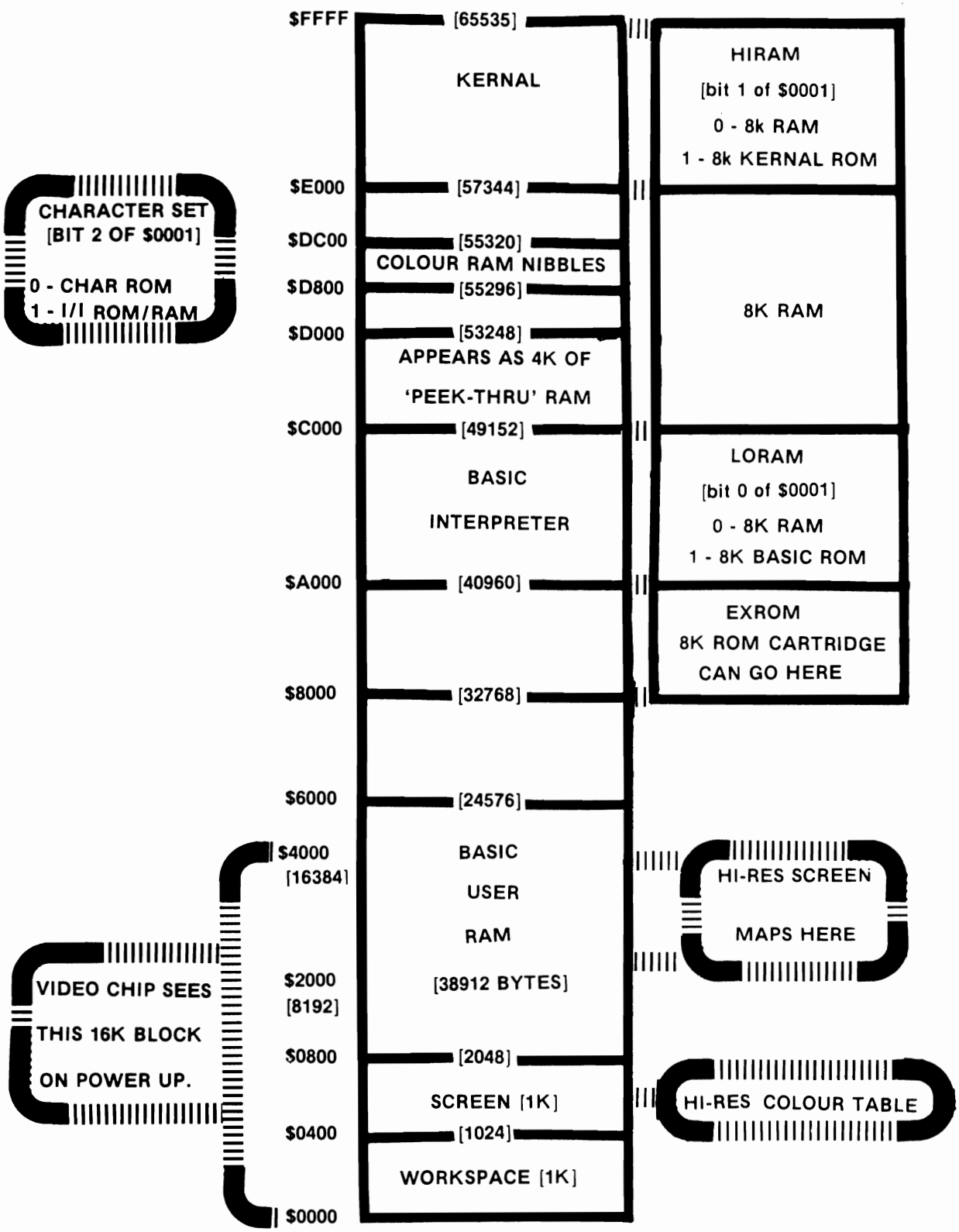
HEX	DECIMAL	DESCRIPTION	
0293	659	RS-232 control reg	
0294	660	RS-232 command reg	
0295-0296	661-662	Bit timing	
0297	663	RS-232 status	
0298	664	# bits to send	
0299-029A	665	RS-232 speed/code	
029B	667	RS232 receive pointer	
029C	668	RS232 input pointer	
029D	669	RS232 transmit pointer	
029E	670	RS232 output pointer	
029F-02A0	671-672	IRQ save during tape I/O	
02A1	673	CIA 2 (NMI) Interrupt Control	
02A2	674	CIA 1 Timer A control og	
02A3	675	CIA 1 Interrupt Log	
02A4	676	CIA 1 Timer A enabled flag	
02A5	677	Screen row marker	
02C0-02FE	704-766	(Sprite 11)	
0300-0301	768-769	Error message link	
0302-0303	770-771	Basic warm start link	
0304-0305	772-733	Crunch Basic tokens link	
0306-0307	774-775	Print tokens link	
0308-0309	776-777	Start new Basic code link	
030A-030B	778-779	Get arithmetic element link	
030C	780	SYS A-reg save	
030D	781	SYS X-reg save	
030E	782	SYS Y-reg save	
030F	783	SYS status reg save	
0310-0312	784-785	USR function jump	(B248)
0314-0315	788-789	Hardware interrupt vector	(EA31)
0316-0317	790-791	Break interrupt vector	(FE66)
0318-0319	792-793	NMI interrupt vector	(FE47)
031A-031B	794-795	OPEN vector	(F34A)
031C-031D	796-797	CLOSE vector	(F291)
031E-031F	798-799	Set-input vector	(F20E)
0320-0321	800-801	Set-output vector	(F250)
0322-0323	802-803	Restore I/O vector	(F333)
0324-0325	804-805	INPUT vector	(F157)
0326-0327	806-807	Output vector	(F1CA)

HEX	DECIMAL	DESCRIPTION	
0328-0329	808-809	Test-STOP vector	(F6ED)
032A-032B	810-811	GET vector	(F13E)
032C-032D	812-813	Abort I/O vector	(F32F)
032E-032F	814-815	Warm start vector	(FE66)
0330-0331	816-817	LOAD link	(F4A5)
0332-0333	818-819	SAVE link	(F5ED)
033C-03FB	828-1019	Cassette buffer	
0340-037E	832-894	(Sprite 13)	
0380-03BE	896-958	(Sprite 14)	
03C0-03FE	960-1022	(Sprite 15)	
0400-07F7	1024-2039	Screen memory (default)	
07F8-07FF	2040-2047	Sprite Pointers(default)	
0800-9FFF	2048-40959	BASIC ROM memory	
8000-9FFF	32768-40959	Alternate: ROM plug-in area	
A000-BFFF	40960-49151	ROM: Basic	
A000-BFFF	49060-59151	Alternate: RAM	
C000-CFFF	49152-53247	RAM memory, including alternate	
D000-D02E	53248-53294	Video Chip (6566)	
D400-D41C	54272-54300	Sound Chip (6581 SID)	
D800-DBFF	55296-56319	Color nybble memory	
DC00-DC0F	56320-56335	Interface chip 1, IRQ (6526 CIA)	
DD00-DD0F	56576-56591	Interface chip 2, NMI (6526 CIA)	
D000-DFFF	53248-53294	Alternate: Character set	
E000-FFFF	57344-65535	ROM: Operating System	
E000-FFFF	57344-65535	Alternate: RAM	
FF81-FFF5	65409-65525	Jump Table, Including:	
	FFC6	Set Input Channel	
	FFC9	Set Output channel	
	FFCC	Restore default I/O channels	
	FFCF	INPUT	
	FFD2	PRINT	
	FFE1	Test Stop Key	
	FFE4	GET	

A000 ROM control vectors
A00C Keyword actión vectors
A052 Function vectors
A080 Operator vectors
A09E Keywords
A19E Error messages
A328 Error message vectors
A365 Misc messages
A38A Scan stack for FOR/GOSUB
A3B8 Move memory
A3FB Check stack depth
A408 Check memory space
A435 'out of memory'
A437 Error routine
A469 BREAK entry
A474 'ready.'
A480 Ready for BASIC
A49C Handle new line
A533 Re-chain lines
A560 Receive input line
A579 Crunch tokens
A613 Find BASIC line
A642 Perform [NEW]
A65E Perform [CLR]
A68E Backup text pointer
A69C Perform [LIST]
A742 Perform [FOR]
A7ED Execute statement
A81D Perform [RESTORE]
A82C Break
A82F Perform [STOP]
A831 Perform [END]
A857 Perform [CONT]
A871 Perform [RUN]
A883 Perform [GOSUB]
A8A0 Perform [GOTO]
A8D2 Perform [RETURN]
A8F8 Perform [DATA]
A906 Scan for next statement
A928 Perform [IF]
A93B Perform [REM]
A94B Perform [ON]
A96B Get fixed point number
A9A5 Perform [LET]
AA80 Perform [PRINT#]
AA86 Perform [CMD]
AAA0 Perform [PRINT]
AB1E Print string from (y.a)
AB3B Print format character
AB4D Bad input routine
AB7B Perform [GET]
ABA5 Perform [INPUT#]
ABBF Perform [INPUT]
ABF9 Prompt & input
AC06 Perform [READ]
ACFC Input error messages
AD1E Perform [NEXT]
AD78 Type match check
AD9E Evaluate expression
AEA8 Constant - PI
AEF1 Evaluate within brackets
AEF7 ')'
AEFF comma..
AF08 Syntax error
AF14 Check range
AF28 Search for variable
AFA7 Setup FN reference
AFE6 Perform [OR]
AFE9 Perform [AND]
B016 Compare
B081 Perform [DIM]
B08B Locate variable
B113 Check alphabetic
B11D Create variable
B194 Array pointer subroutine
B1A5 Value 32768
B1B2 Float-fixed conversion
B1D1 Set up array
B245 'BAD SUBSCRIPT'
B248 'ILLEGAL QUANTITY'
B34C Compute array size
B37D Perform [FRE]
B391 Fix-float conversion
B39E Perform [POS]
B3A6 Check direct
B3B3 Perform [DEF]
B3E1 Check FN syntax
B3F4 Perform [FN]
B465 Perform [STR$]
B475 Calculate string vector
B487 Set up string
B4F4 Make room for string
B526 Garbage collection
B5BD Check salvageability
B606 Collect string
B63D Concatenate
B67A Build string to memory
B6A3 Discard unwanted string
B6DB Clean descriptor stack
B6EC Perform [CHR$]
B700 Perform [LEFT$]
B72C Perform [RIGHT$]
B737 Perform [MID$]
B761 Pull string parameters
B77C Perform [LEN]
B782 Exit string-mode
B78B Perform [ASC]
B79B Input Byte parameter
B7AD Perform [VAL]
B7EB Parameters for POKE/WAIT
B7F7 Float-fixed
B80D Perform [PEEK]
B824 Perform [POKE]
B82D Perform [WAIT]
B849 Add 0.5
B850 Subtract-from
B853 Perform [subtract]
B86A Perform [add]
B947 Complement FAC#1
B97E 'overflow'
B983 Multiply by zero byte
B9EA Perform [LOG]
BA2B Perform [multiply]
BA59 Multiply-a-bit
BA8C Memory to FAC#2
BAB7 Adjust FAC#1/#2
BAD4 Underflow/overflow
BAE2 Multiply by 10
BAF9 +10 in floating pt
BAFE Divide by 10
BB12 Perform [divide]
BBA2 Memory to FAC#1
BBC7 FAC#1 to memory
BBFC FAC#2 to FAC#1
BC0C FAC#1 to FAC#2
BC1B Round FAC#1
BC2B Get sign
BC39 Perform [SGN]
BC58 Perform [ABS]
BC5B Compare FAC#1 to mem
BC9B Float-fixed
BCCC Perform [INT]
BCF3 String to FAC
BD7E Get ASCII digit
BDC2 Print 'IN..'
BDCD Print line number
BDDD Float to ASCII
BF16 Decimal constants
BF3A TI constants
BF71 Perform [SQR]
BF7B Perform [power]
BFB4 Perform [negative]
BFED Perform [EXP]
E043 Series eval 1
E059 Series eval 2
E097 Perform [RND]
E019 ?? breakpoints ??
E12A Perform [SYS]
E156 Perform [SAVE]
E165 Perform [VERIFY]
E168 Perform [LOAD]
E1BE Perform [OPEN]
E1C7 Perform [CLOSE]
E1D4 Parameters for LOAD/SAVE
E206 Check default parameters
E20E Check for comma
E219 Parameters for open/close
E264 Perform [COS]
E26B Perform [SIN]
E2B4 Perform [TAN]
E30E Perform [ATN]

E37B Warm restart
E394 Initialize
E3A2 CHRGET for zero page
E3BF Initialize BASIC
E447 Vectors for $300
E453 Initialize vectors
E45F Power-up message
E500 Get I/O address
E505 Get screen size
E50A Put/get row/column
E518 Initialize I/O
E544 Clear screen
E566 Home cursor
E56C Set screen pointers
E5A0 Set I/O defaults
E5B4 Input from keyboard
E632 Input from screen
E684 Quote test
E691 Set up screen print
E6B6 Advance cursor
E6ED Retreat cursor
E701 Back into previous line
E716 Output to screen
E87C Go to next line
E891 Perform < return >
E8A1 Check line decrement
E8B3 Check line increment
E8CB Set color code
E8DA Color code table
E8EA Scroll screen
E965 Open space on screen
E9C8 Move a screen line
E9E0 Synchronize color transfer
E9F0 Set start-of-line
E9FF Clear screen line
EA13 Print to screen
EA24 Synchronize color pointer
EA31 Interrupt — clock etc
EA87 Read keyboard
EB79 Keyboard select vectors
EB81 Keyboard 1 — unshifted
EBC2 Keyboard 2 — shifted
EC03 Keyboard 3 — 'comm'
EC44 Graphics/text contrl
EC4F St graphics/text mode
EC78 Keyboard 4
ECB9 Video chip setup
ECE7 Shift/run equivalent
ECF0 Screen ln address low
ED09 Send 'talk'
ED0C Send 'listen'
ED40 Send to serial bus
EDB2 Serial timeout
EDB9 Send listen SA
EDBE Clear ATN
EDC7 Send talk SA
EDCC Wait for clock
EDDD Send serial deferred
EDEF Send 'untalk'
EDFE Send 'unlisten'
EE13 Receive from serial bus
EE85 Serial clock on
EE8E Serial clock off
EE97 Serial output '1'
EEAO Serial output '0'
EEA9 Get serial in & clock
EEB3 Delay 1 ms
EEBB RS-232 send
EF06 Send new RS-232 byte
EF2E No-DSR error
EF31 No-CTS error
EF3B Disable timer
EF4A Compute bit count
EF59 RS232 receive
EF7E Setup to receive
EFC5 Receive parity error
EFCA Receive overflow
EFCD Receive break
EFD0 Framing error
EFE1 Submit to RS232
F00D No-DSR error
F017 Send to RS232 buffer
F04D Input from RS232
F086 Get from RS232
F0A4 Check serial bus idle
F0BD Messages
F12B Print if direct
F13E Get..
F14E ..from RS232
F157 Input
F199 Get..tape/serial/RS232
F1CA Output..
F1DD ..to tape
F20E Set input device
F250 Set output device
F291 Close file
F30F Find file
F31F Set file values
F32F Abort all files
F333 Restore default I/O
F34A Do file open
F3D5 Send SA
F409 Open RS232
F49E LOAD program
F5AF 'searching'
F5C1 Print filename
F5D2 'loading/verifying'
F5DD SAVE program
F68F Print 'SAVING'
F69B Bump clock
F6BC Log PIA key reading
F6DD Get time
F6E4 Set time
F6ED Check stop key
F6FB Output error messages
F72D Find any tape header
F76A Write tape header
F7D0 Get buffer address
F7D7 Set buffer start/end pointers
F817 Find specific header
F7EA Bump tape pointer
F80D 'press play..'
F82E Check tape status
F838 'press record..'
F841 Initiate tape read
F864 Initiate tape write
F875 Common tape code
F8D0 Check tape stop
F8E2 Set read timing
F92C Read tape bits
FA60 Store tape chars
FB8E Reset pointer
FB97 New character setup
FBA6 Send transition to tape
FBC8 Write data to tape
FBCD IRQ entry point
FC57 Write tape leader
FC93 Restore normal IRQ
FCB8 Set IRQ vector
FCCA Kill tape motor
FCD1 Check r/w pointer
FCDB Bump r/w pointer
FCE2 Power reset entry
FD02 Check 8-rom
FD10 8 ROM mask
FD15 Kernal reset
FD1A Kernal move
FD30 Vectors
FD50 Initialize system constnts
FD9B IRQ vectors
FDA3 Initialize I/O
FDDD Enable timer
FDF9 Save filename data
FE00 Save file details
FE07 Get status
FE18 Flag status
FE1C Set status
FE21 Set timeout
FE25 Read/set top of memory
FE27 Read top of memory
FE2D Set top of memory
FE34 Read/set bottom of memory
FE43 NMI entry
FE66 Warm start
FEB6 Reset IRQ & exit
FEBC Interrupt exit
FEC2 RS-232 timing table
FED6 NMI RS-232 in
FF07 NMI RS-232 out
FF43 Fake IRQ
FF48 IRQ entry
FF81 Jumbo jump table
FFFA Hardware vectors

COMMODORE 64 MEMORY MAP



SID (6581) COMMODORE 64

Voice V1	Voice V2	Voice V3		Voice V1	Voice V2	Voice V3
$D400	$D407	$D40E	FREQUENCY L	54272	54279	54286
$D401	$D408	$D40F	FREQUENCY H	54273	54280	54287
$D402	$D409	$D410	PULSE WIDTH L	54274	54281	54288
$D403	$D40A	$D411	PULSE WIDTH H (0 0 0 0)	54275	54282	54289
$D404	$D40B	$D412	VOICE TYPE NSE PUL SAW TRI KEY	54276	54283	54290
$D405	$D40C	$D413	ATTACK TIME 2 ms - 8 secs / DECAY TIME 6 ms - 24 secs	54277	54284	54291
$D406	$D40D	$D414	SUSTAIN LEVEL / RELEASE TIME 6 ms - 24 secs	54278	54285	54292

voices
(Write only)

$D415	(0 0 0 0 0) FILTER FREQUENCY L	54293
$D416	FILTER FREQUENCY H	54294
$D417	RESONANCE / FILTER VOICES EXT V3 V2 V1	54295
$D418	V3 off PASSBAND: HI BD LO / MASTER VOLUME	54296

FILTER & VOLUME
(Write only)

$D419	PADDLE X (A/D#1)	54297
$D41A	PADDLE Y (A/D #2)	54298
$D41B	NOISE 3 (Random)	54299
$D41C	ENVELOPE 3	54300

SENSE
(Read only)

Special voice features (TEST, RING MOD, SYNC) are omitted from the above diagram.

PROCESSOR I/O PORT (6510) COMMODORE 64

	7	6	5	4	3	2	1	0	
$0000	IN	IN	OUT	IN	OUT	OUT	OUT	OUT	DDR 0
$0001			TAPE MOTOR	TAPE SENSE	TAPE WRITE	D-ROM SWITCH	EF-RAM SWITCH	AB-RAM SWITCH	PR 1

CIA 1 (IRQ) (6526) COMMODORE 64

$DC00	PADDLE SEL A	PADDLE SEL B		JOYSTICK 0 FIRE	Right	Left	Down	Up	PRA 56320
	KEYBOARD ROW SELECT (INVERTED)								
$DC01				JOYSTICK 1 FIRE	Right	Left	Down	Up	PRB 56321
	KEYBOARD COLUMN READ								
$DC02	$FF — ALL OUTPUT								DDRA 56322
$DC03	$00 — ALL INPUT								DDRB 56323
$DC04 $DC05	TIMER A								TAL 56324 TAH 56325
$DC06 $DC07	TIMER B								TBL 56326 TBH 56327
$DC0D			TAPE INPUT			TIMER B	INTERR. A		ICR 56333
$DC0E				ONE SHOT	OUT MODE	TIME PB6 OUT	TIMER A START		CRA 56334
$DC0F				ONE SHOT	OUT MODE	TIME PB7 OUT	TIMER B START		CRB 56335

CIA 2 (NMI) (6526) COMMODORE 64

$DD00	SERIAL IN	CLOCK IN	SERIAL OUT	CLOCK OUT	ATN OUT	RS-232 OUT	VIC II ADDR.15	VIC II ADDR.14	PRA 56576
$DD01	DSR IN	CTS IN		DCD# IN	RI# IN	DTR OUT	RTS OUT	RS-232 IN	PRB 56577
$DD02	$3F — SERIAL								DDRA 56578
$DD03	$00 — P.U.P. ALL INPUT or $06 — RS-232								DDRB 56579
$DD04 $DD05	TIMER A								TAL 56580 TAH 56581
$DD06 $DD07	TIMER B								TBL 56582 TBH 56583
$DD0D			RS-232 IN			TIMER B	INTERRUPT A		ICR 56589
$DD0E	TIMER A START								CRA 56590
$DD0F	TIMER B START								CRB 56591

*** CONNECTED but not used by O.S.**

Commodore 64 Maps

By Don White, Ottawa, Ont.

MANIPULATING THE LO-RES & HI-RES SCREENS ON THE COMMODORE 64

Register 24 in the video chip allows the user to have a number of low-resolution screens available. The configuration of this register is as follows:

VB13	VB12	VB11	VB10	CB13	CB12	CB11	NC
8192	4096	2048	1024	8192	4096	2048	

VIDEO MATRIX BASE

These 4 bits are bits 10-13 of the starting address of the low-resolution screen. Thus, the lo-res screen can theoretically start anywhere between 1024 (dec) and 15360 (dec) on 1K boundaries. However, the screens starting at the following addresses are not available: 4096 5120 6144 7168.

CHARACTER SPACE BASE

These 3 bits are bits 11-13 of the starting address of the programmable character memory. In theory, character memory can be located anywhere between 2048 (dec) and 14336 (dec) on 2K boundaries. However, the blocks starting at 4096 (dec) and 6144 (dec) cannot be used. When bit 2 is set, the character set at 53248 (dec) is accessed, i.e., graphics/upper case, and when bits 1 and 2 are set, the upper/lower case character set at 55296 (dec) is utilized. Thus, character sets can be stored starting at the following addresses: 2048 8192 10240 12288 14336.

To relocate the lo-res screen, bits 4-7 of register 24 must be set to point to the start address of the screen. Also, address 648 (dec) must be loaded with the actual physical memory page number where the screen is to start. For example, to relocate the lo-res screen to 2048 (dec), the following statements must appear in your program:

```
POKE 53272,37:REM POINT TO 2048(dec)
POKE 648,8 :REM MEMORY PAGE 8 (i.e., 8*256 = 2048)
```

The POKE to 53272 (register 24) sets bits 5, 2 and 1. This means that the video display will start at 2048 (dec) and the character set address is 53248 (dec). Any characters in PRINT statements will now appear on the lo-res screen in its new location. However, POKEs to the screen must be adjusted to point to the appropriate area of memory. This can be accomplished by storing the calculated start of screen memory in a variable and using offsets to determine the POKE locations. The start of the lo-res screen can be determined using one of the following formulae:

```
lr = 64 * (peek(53272) and 240)
or
lr = 256 * peek(648)
```

Alternatively, a small sub-routine can be used to poke the appropriate byte into register 24 once the start address of the lo-res screen has been selected.

```
10 sc   =   10240:rem start address of lo-res screen
20 gosub 10000


10000 rem set lo-res screen location
10001 rem does not change character set pointer
10002 rem
10010 poke 53272,(peek(53272) and 15) + (240 and 16*sc/1024)
10020 poke 648,sc/256
10030 return
```

The 6566 Video Chip is only capable of looking at 16K blocks of memory. On power-up, the video chip sees the bottom 16K. Which block is enabled is controlled by bits 0 and 1 at address 56576 (dec) in CIA 2 (6526). The values of the two bits and the block of RAM addressed are as follows:

Bit 1	Bit 0	16K Block of RAM	Starting Address
1	1	Block 0	0
1	0	Block 1	16384
0	1	Block 2	32768
0	0	Block 3	49152

With a small set-up program, the 64 can quickly be configured to look like a PET as far as screen handling is concerned. It must be stressed that this routine only deals with PEEKs and POKEs to screen RAM, and will not take care of any PEEKs and POKEs to Zero Page.

```
100 rem configure 64 to handle screen like pet
110 rem
120 poke 56,128:poke 55,0:          rem protect memory above 32768
130 poke 56576,peek(56576) and 253: rem enable bank 2
140 poke 53272,peek(53272) and 15:  rem lores screen starts at
150 poke 648,128:                   rem 32768 or memory page 128
160 poke 44,4:poke 43,1:poke 1024,0: rem basic starts at 1024
```

Once this program has been RUN, any PET programs that PEEK and POKE the PET screen will operate properly on the 64. However, I must again point out that this in no way will adjust the system for PEEKs and POKEs to other areas of the computer. Furthermore, it does not compensate for the differences in the coding of the keyboards.

The bit graphics mode on the 64 can be enabled by setting bit 5 in Register 17 (53265 (dec)) of the Video Chip. Also, bit 3 in Register 24 (53272 (dec)) must be set so that the character base starts at 8192 (dec). This can be accomplished as follows:

```
100 poke 53265,peek(53265) or 32:rem bit map mode
110 poke 53272.25:rem hires screen at 8192
```

This routine will leave the lo-res screen, which becomes the colour nybble map for the hi-res screen, at 1024 (dec). The start-of-BASIC will still be located at 2048 (dec). This is fine if the routine you plan to use is quite small. In this case, the hi-res screen can be protected with a POKE 56,32 and POKE 55,0, which will lower the top-of-memory pointer.

If you plan to use a large program, the start-of-BASIC should be moved to 16384 (dec). This is done by:

```
poke 44,64:poke 43,1:poke 16384,0:clr:new
```

Once the hi-res screen has been established, points can be plotted with the following routine which was extracted from the VIC Programmer's Manual.

```
1000 rem hires plotting routine
1010 rem
1020 ch = int(x/8) + int(y/8) * 40
1030 ro = int((y/8 - int(y/8)) * 8)
1040 by = 8192 + 8 * ch + ro
1050 bi = 7 - (x - (int(x/8) * 8)
1060 poke by,peek(by) or (2(uparrow)bi)
1070 return
```

COMMODORE 64 MEMORY MAP

HEX	DECIMAL	DESCRIPTION
0000-03FF	0000-1023	Zero page basic pointers/stack/etc.
0400-07FF	1024-2047	Screen RAM
0800-9FFF	2048-40959	Program text area
2000-3FFF	8192-16383	Hi-res screen (alternate)
8000-9FFF	32768-40959	8K cartridge ROM area
A000-BFFF	40960-49151	8K BASIC (V2)
C000-CFFF	49152-53247	4K RAM
D000-DFFF	53248-57343	'66-'81-'26(2)-10 exp1&2/4K char space
E000-FFFF	57344-65535	ROM - Operating system

EDITOR'S NOTE:

The following are maps released by Commodore to some dealers but not known to be generally available anywhere to users. As originally distributed, they had quite a number of errors, and these have been corrected by Don. — Ed.

6566 VIDEO CHIP (D000-D02E/53248-53294)

REGISTER # Dec.	Hex	ADDRESS Dec.	Hex	DESCRIPTION
0	0	53248	D000	SPRITE 0 X cmp
1	1	53249	D001	SPRITE 0 Y cmp
2	2	53250	D002	SPRITE 1 X cmp
3	3	53251	D003	SPRITE 1 Y cmp
4	4	53252	D004	SPRITE 2 X cmp
5	5	53253	D005	SPRITE 2 Y cmp
6	6	53254	D006	SPRITE 3 X cmp
7	7	53255	D007	SPRITE 3 Y cmp
8	8	53256	D008	SPRITE 4 X cmp
9	9	53257	D009	SPRITE 4 Y cmp
10	A	53258	D00A	SPRITE 5 X cmp
11	B	53259	D00B	SPRITE 5 Y cmp
12	C	53260	D00C	SPRITE 6 X cmp
13	D	53261	D00D	SPRITE 6 Y cmp
14	E	53262	D00E	SPRITE 7 X cmp
15	F	53263	D00F	SPRITE 7 Y cmp
16	10	53264	D010	SPRITE X cmp (msb of x coord.)
17	11	53265	D011	Bit 7 Raster compare Bit 6 Extended colour mode Bit 5 Bit map mode Bit 4 Blank/unblank screen Bit 3 24/25 row select (1 25 rows) Bit 2-0 Scroll in y position
18	12(R/0)	53266	D012	Raster read (raster cmp IRQ write)
19	13(R/0)	53267	D013	Light pen latch x
20	14(R/0)	53268	D014	Light pen latch y
21	15	53269	D015	SPRITE disable (1 SPRITE enabled)
22	16	53270	D016	Bit 7-5 Unused Bit 4 Multi-colour mode Bit 3 38/40 column select (1= 40 col.) Bit 2-0 Scroll in x position
23	17	53271	D017	SPRITE expand in IY
24	18	53272	D018	Bit 7-4 Video matrix base Bit 3-1 Character space base
25	19	53273	D019	Bit 7 Follows IRQ line Bit 2 IRQ for SPRITE to SPRITE collision Bit 1 IRQ for SPRITE to background Bit 0 Raster cmp IRQ collision
26	1A	53274	D01A	IRQ mask register (0= interrupt disabled)
27	1B	53275	D01B	Background to SPRITE priority
28	1C	53276	D01C	Multi-colour SPRITE select
29	1D	53277	D01D	SPRITE expand in IX
30	1E	53278	D01E	SPRITE to SPRITE collision detect
31	1F	53279	D01F	SPRITE to background collision detect

6566 VIDEO CHIP COLOR REGISTERS (Bit 3-0)

REGISTER # Dec.	Hex	ADDRESS Dec.	Hex	DESCRIPTION
32	20	53280	D020	Border colour
33	21	53281	D021	Background colour 0
34	22	53282	D022	Background colour 1
35	23	53283	D023	Background colour 2
36	24	53284	D024	Background colour 3
37	25	53285	D025	SPRITE multi-colour register 0
38	26	53286	D026	SPRITE multi-colour register 1
39	27	53287	D027	SPRITE 0 colour
40	28	53288	D028	SPRITE 1 colour
41	29	53289	D029	SPRITE 2 colour
42	2A	53290	D02A	SPRITE 3 colour
43	2B	53291	D02B	SPRITE 4 colour
44	2C	53292	D02C	SPRITE 5 colour
45	2D	53293	D02D	SPRITE 6 colour
46	2E	53294	D02E	SPRITE 7 colour

REGISTER DESCRIPTION FOR THE 6566 VIDEO CHIP AS USED IN THE COMMODORE 64

Registers 0-1 control the co-ordinate position of SPRITE 0; Register 0 controls the X-axis and Register 1 controls the Y-axis.

Registers 0-15 are paired like 0 and 1 for SPRITES 0-7.

Register 16 is the most significant bit of the X co-ordinate of the SPRITES. This way, we can move a SPRITE all the way across the 320 pixel X-axis. Bit 0 corresponds to SPRITE 0 and so on.

Register 17 is a multi-function register:
Bit 0-2 Scroll in Y position
Bit 3 24/25 row select (1 = 25 rows)
Bit 4 Blank/unblank screen (0 = turn off video display)
Bit 5 Bit map mode (Hi-res mode)
Bit 6 Extended colour mode
Bit 7 Raster compare

Register 18 is the raster compare IRQ write register, and is read only.

Register 19 and 20 are light pen latches for X and Y and are read only.

Register 21 is the SPRITE enable register. Each bit (0-7) set corresponds to a SPRITE (0-7).

Register 22 is another multi-function register:
Bit 0-2 Scroll in X position
Bit 3 38/40 column select (1 = 40 columns)
Bit 4 Multi-colour mode
Bit 5-7 Not used

Register 23 is the SPRITE expand in Y register. Setting the bit corresponding to the SPRITE doubles the vertical size (Y) of the SPRITE.

Register 24 is a dual-function register:
Bit 0 Unused
Bit 1-3 Character space base
Bit 4-7 Video matrix base

Register 25 is IRQ register:
Bit 0 Raster cmp IRQ
Bit 1 IRQ for SPRITE-background collision
Bit 2 IRQ for SPRITE-SPRITE collision
Bit 3 Light pen IRQ
Bit 4-7 Not used

Register 26 is IRQ mask register (0 = interrupt disabled).

Register 27 is background-SPRITE priority register (0 = SPRITE has priority).

Register 28 is the multi-colour SPRITE select.

Register 29 is the SPRITE expand in X register (see Register 23).

Register 30 is the SPRITE-SPRITE collision detection register.

Register 31 is the SPRITE-background collision detection register.

Registers 32-46 are 4-bit colour registers.

COLOUR

CODE	COLOUR
0	Black
1	White
2	Red
3	Cyan
4	Purple
5	Green
6	Blue
7	Yellow
8	Orange
9	Brown
10	Light Red
11	Gray 1
12	Gray 2
13	Light Green
14	Light Blue
15	Gray 3

REGISTER	FUNCTION
32	Exterior colour
33	Background 0
34	Background 1
35	Background 2
36	Background 3
37	SPRITE multi-colour reg 0
38	SPRITE multi-colour reg 1
39	SPRITE 0 colour
40	SPRITE 1 colour
41	SPRITE 2 colour
42	SPRITE 3 colour
43	SPRITE 4 colour
44	SPRITE 5 colour
45	SPRITE 6 colour
46	SPRITE 7 colour

NOTE: Only colours 0-7 may be used in multi-colour character mode.

6581 (SID) SYNTHESIZER CHIP (D400-D41C/54272-54300)

REGISTER # Dec.	Hex	ADDRESS Dec.	Hex	DESCRIPTION
0	0	54272	D400	Frequency lo
1	1	54273	D401	Frequency hi
2	2	54274	D402	Pulse width lo
3	3	54275	D403	Bit 7-4 Unused Bit 3-0 Pulse width hi
4	4	54276	D404	Control register voice 1 Bit 7 Noise Bit 6 Pulse Bit 5 Sawtooth Bit 4 Triangle Bit 3 Test bit Bit 2 Ring modulation Bit 1 Sync Bit 0 Gate
5	5	54277	D405	Attack/decay register
6	6	54278	D406	Sustain/release register
7-12	7-D	54279-54285	D407-D40D	Control register voice 2 (Functionally identical to D400-D406)
13-20	E-14	54286-54292	D40E-D414	Control register voice 3 (Functionally identical to D400-D406)
21	15	54293	D415	Cutoff frequency lo
22	16	54294	D416	Cutoff frequency hi
23	17	54295	D417	Bit 7-4 Resonance of filter (Bit 3-0 Select signals to be routed through filter. Bits set to zero appear directly at audio output. bits set to 1 will be processed through filter) Bit 3 External input Bit 2 Voice 3 Bit 1 Voice 2 Bit 0 Voice 1
24	18	54296	D418	(Bit 7-4 Select filter mode and output options) Bit 7 Off Bit 6 High pass Bit 5 Band pass Bit 4 Low pass Bit 3-0 Output volume
25	19	54297	D419	Pot X
26	1A	54298	D41A	Pot Y
27	1B	54299	D41B	Oscillator 3/random number generator
28	1C	54300	D41C	Envelope 3

REGISTER DESCRIPTION FOR THE 6581 (SID) SYNTHESIZER AS USED IN THE COMMODORE 64

The SID chip is a three-voice polyphonic synthesizer on single 24-pin IC. In addition to independent ADSR envelopes for each voice, the SID allows osc sync, ring mod, high, low and band pass filtering as well as two a/d converters.

Registers 0-6 control voice 1. Registers 7-13 and 14-20 are organized similarly for voices 2 and 3, respectively.

Registers 0 and 1 represent the 16-bit number which linearly controls the frequency of osc 1.

Register 2 and bits 0-3 of register 3 represent the 12-bit number which controls the pulse width of osc 1.

Register 4 is the control register for osc 1. Its bits select various output options.

Bit 0 Gate bit: when set the envelope is triggered and attack/decay/sustain cycle begins; when unset, the release cycle begins.
Bit 1 Sync bit: synchronizes osc 1 to osc 3 (2 to 1 for voice 2, 3 to 2 for voice 3).
Bit 2 Ring Mod bit: replaces osc 1 output with the ring modulated output of osc 1 and osc 3 (2 and 1 for voice 2, 3 and 2 for voice 3).
Bit 3 Test bit: when set osc 1 is reset and locked to zero until it is unset.
Bit 4 Triangle waveform: when set the triangle waveform output is selected. The triangle waveform is low in harmonics and has a mellow, flute-like quality.
Bit 5 Sawtooth waveform: when set the sawtooth waveform output is selected. The sawtooth waveform is rich in even and odd harmonics, and has a bright, brassy quality.
Bit 6 Pulse waveform: when set the pulse waveform output of osc 1 is selected. The harmonic quality of this waveform can be adjusted by the pulse width registers, producing a tone from a bright, hollow square wave to a nasal, reedy pulse. Sweeping the pulse produces a dynamic 'phasing' effect.
Bit 7 Noise waveform: when set, the noise output waveform of osc 1 is selected. This output is a random signal which changes at the frequency of osc ?. The sound quality can be varied from a low rumbling to a hissing white noise.

Register 5 is the attack/decay register.

Bit 4-7 Attack rate (value from 0 to 15; see Table 1).
Bit 0-3 Decay rate (value from 0 to 15; see Table 1).

Register 6 is the sustain/release register:

Bit 4-7 Sustain rate (amplitude value during sustain 0 = min 15 = max).
Bit 0-3 Release rate (value from 0 to 15; see Table 1)

Registers 21 and bits 0-2 of register 22 control the cutoff frequency for the programmable filter. FC(out) = (30 + FC(in) * 5.8) Hz.

Register 23 is the resonance/filter control.

Bit 4-7 Controls the resonance of the filter. There are 16 settings from 0 (no resonance) to 15 (maximum resonance).
Bit 0-3 Determine signals to be routed through the filter.
Bit 0 Voice 1: when set, signal will be processed; when unset, signal will appear directly at the output.

Bit 1 Same for voice 2
Bit 2 Same for voice 3
Bit 3 Same for external input

Register 24 is the filter mode/volume control.
Bit 4-7 Selects various filter modes.
Bit 4 When set, low pass filter is selected (components below cutoff are unaltered; components above cutoff are attenuated at 12 dB/octave).
Bit 5 When set, band pass filter is selected (all frequency components above and below cutoff are attenuated at 6 dB/octave).
Bit 6 When set, high pass filter is selected (components above cutoff are unaffected; components below cutoff are attenuated at 12 dB/octave).
Bit 7 When set, voice 3 is disconnected from audio path (this allows voice 3 to be used for modulation without any undesirable output).
Bit 0-3 Select 1 of 16 overall volume levels for the final composite audio output.

Register 25 is an A/D converter for POTX (pin 24).

Register 26 is an A/D converter for POTY (pin 23).

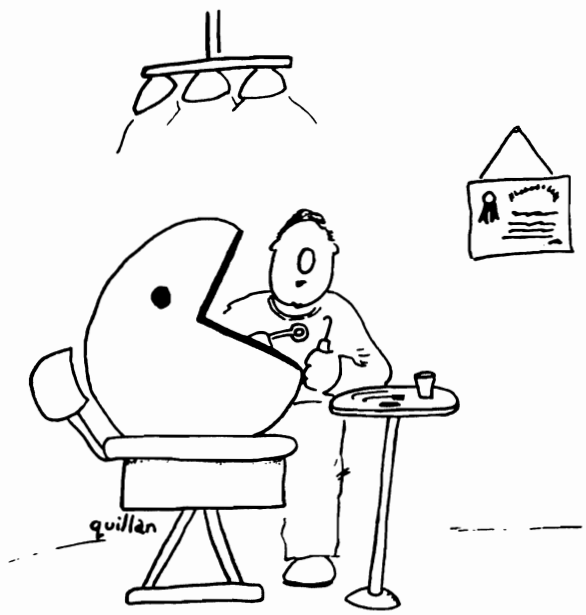
Register 27 is oscillator 3/random number generator register. It allows the microprocessor to read the upper 8 output bits of oscillator 3. The number generated is directly related to the waveform selected.
If the sawtooth waveform is selected, the register presents a series of numbers incrementing from 0 to 255 at a rate determined by oscillator 3.
If the triangle waveform is selected, the number will increment from 0 to 255 and then decrement back to 0.
If the pulse waveform is selected, the numbers will jump from 0 and 255.

If the noise waveform is used, a random number will be generated.

Register 28 is the same as oscillator 3, but it allows the microprocessor to read the output of the voice 3 envelope generator. This output can be added to the filter frequency to produce harmonic envelopes, wah and similar effects.

TABLE 1: ATTACK & DECAY/RELEASE RATES

VALUE	ATTACK RATE	DECAY/RELEASE RATE
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s



SAY "AH"

PET/VIC/64 Cross Reference Map

By Mark Niggemann, Ames, Iowa

These entry points for each of the Commodore ROM sets that are out represent some of the most-called routines in many machine language programs. The programmer is cautioned to check for proper set-up of registers, memory locations, etc., before calling any of these routines.

COMMON ENTRY POINTS
Original/Upgrade/4.0 Rom/VIC 20/CBM 64

Compiled by Mark Niggemann

Orig	Upgr	4.0	VIC	64	Description
C357	C355	B3CD	C435	A435	?OUT OF MEMORY
C359	C357	B3CF	C437	A437	Send Basic error message
C38B	C389	B3FF	C474	A474	Warm start, Basic
C3AC	C3AB	B41F	C49C	A49C	Crunch and insert line
C430	C439	B4AD	C52A	A52A	Fix chaing and READY
C433	C442	B4B6	C533	A533	Fix chaing
C48D	C495	B4FB	C579	A579	Crunch tokens
C522	C52C	B5A3	C613	A613	Find Basic line
C553	C55D	B5D4	C642	A642	Do NEW
C567	C572	B5E9	C659	A659	Reset Basic and do CLR
C56A	C575	B5EC	C65E	A65E	Do CLR
C59A	C5A7	B622	C68E	A68E	Reset Basic to start
C6B5	C6C4	B74A	C7AE	A7AE	Continue Basic execution
C863	C873	B8F6	C96B	A96B	Get fixed point number from Basic
C9CE	C9DE	BADB	CAD3	AAD3	Send RETURN, LF if in screen mode
C9D2	C9E2	BADF	CAD7	AAD7	Send RETURN, line feed
CA27	CA1C	BB1D	CB1E	AB1E	Print string
CA2D	CA22	BB23	CB24	AB24	Print precomputed string
CA47	CA43	BB44	CB45	AB45	Print '?'
CA49	CA45	BB46	CB47	AB47	Print character
CE11	CDF8	BEF5	CEFD	AEFD	Check for comma
CE13	CDFA	BEF7	CEFF	AEFF	Check for specific character
CE1C	CE03	BF00	CF08	AF08	'SYNTAX ERROR'
CFD7	CFC9	C187	D0E7	B0E7	Find fl-pt variable, given name
D079	D069	C2B9	D185	B185	Bump variable address by 2
D0A7	D09A	C2EA	D1BF	B1BF	Float to fixed conversion
D278	D26D	C4BC	D391	B391	Fixed to float conversion
D679	D67B	C8D7	D7A1	B7A1	Get byte to X reg
D68D	D68F	C8EB	D7B5	B7B5	Evaluate string
D6C4	D6C6	C921	D7EB	B7EB	Get two parameters
D73C	D773	C99D	D867	B867	Add (from memory)
D8FD	D934	CB5E	DA28	BA28	Multiply by memory location
D9B4	D9EE	CC18	DAE2	BAD2	Multiply by ten
DA74	DAAE	CCD8	DBA2	BBA2	Unpack memory variable to FAC#1
DAA9	DAE3	CD0D	DBA2	BBA2	Copy FAC#1 to (X,Y) location
DB1B	DB55	CD7F	DC49	BC49	Completion of fixed to float conversion
DC9F	DCD9	CF83	DDCD	BDCD	Print fixed-pt value conversion
DCA9	DCE3	CF8D	DDD7	BDD7	Print floating pt value
DCAF	DCE9	CF93	DDDD	BDDD	Convert number to ASCII string
E3EA	E3D8	E202	E742	E716	Print a character

NA	E775	D722	NA	NA	Output byte as 2 hex digits
NA	E7A7	D754	NA	NA	Input 2 hex digits to .A
NA	E7B6	D763	NA	NA	Input 1 hex digit to .A
E7DE	F156	F185	F1E6	F12F	Print system message
F0B6	F0B6	F0D2	FFB4	FFB4	Send 'TALK' to bus
F0BA	F0BA	F0D5	FFB1	FFB1	Send 'LISTEN' to bus
F12C	F128	F143	FF93	FF93	Send 'LISTEN' Secondary Address
F12C	F128	F143	FF96	FF96	Send 'TALK' Secondary Address
F167	F16F	F19E	FFA8	FFA8	Send character to bus
F17A	F17F	F1B6	FFAB	FFAB	Send 'UNTALK' to bus
F17E	F183	F1B9	FFAE	FFAE	Send 'UNLISTEN' to bus
F187	F18C	F1C0	FFA5	FFA5	Input from bus
F2CD	F2AE	F2E2	FFC3	FFC3	Close logical file in .A
F32A	F301	F335	FFE1	FFE1	Check stop key
F33F	F315	F349	F1E2	F12B	Send message if direct
NA	F322	F356	FFD5	FFD5	Load subroutine
F3FF	F40A	F449	F647	F5AF	Print 'SEARCHING'
F411	F41D	F45C	F659	F5C1	Print file name
F43F	F447	F486	NA	NA	Get load/save type parms
F462	F466	F4A5	FFC9	FFC9	Open channel for output
F495	F494	F4D3	F867	F7EA	Find specific tape header block
F52A	F521	F560	FFC0	FFC0	Open logical file
F579	F56E	F5AD	F784	F701	?FILE NOT FOUND
F57B	F570	F5AF	F798	F715	Send error message
F5AE	F5A6	F5E5	F7AF	F72C	Find any tape block header
F667	F656	F695	F84D	F7D0	Set tape buffer start address
F67D	F66C	F6AB	F854	F7D7	Set cassette buffer pointers
F6E6	F6F0	F72F	FFCC	FFCC	Close channel
F78B	F770	F7AF	FFC6	FFC6	Set input device from LFN
F7DC	F7BC	F7DF	FFC9	FFC9	Set output device from LFN
F83B	F812	F857	F894	F817	PRESS PLAY..; wait
F85E	F835	F87A	F8AB	F82E	Sense tape switch
F87F	F855	F89A	F8C0	F841	Read tape to buffer
F88A	F85E	F8A3	F8C9	F849	Read tape
F8B9	F886	F8CB	F8E3	F864	Write tape from buffer
F8C1	F88E	F8D3	F8E8	F869	Write tape, leader length in .A
F913	F8E6	F92B	F94B	F8D0	Wait for I/O complete or stop key
FBDC	FB76	FBBB	FBD2	FB8E	Reset tape I/O pointer
FD1B	FC9B	FCE0	FCF6	FCB8	Set interrupt vector
FFC6	FFC6	FFC6	FFC6	FFC6	Set input device
FFC9	FFC9	FFC9	FFC9	FFC9	Set output device
FFCC	FFCC	FFCC	FFCC	FFCC	Restore default I/O
FFCF	FFCF	FFCF	FFCF	FFCF	Input character
FFD2	FFD2	FFD2	FFD2	FFD2	Output character
FFD5	FFD5	FFD5	FFD5	FFD5	Load program to ram
FFD8	FFD8	FFD8	FFD8	FFD8	Save program from ram
FFE1	FFE1	FFE1	FFE1	FFE1	Check stop key
FFE4	FFE4	FFE4	FFE4	FFE4	Get character

Anyone can be a success at computer games. To win in FROGGER takes TOADAL concentration, while at HANGMAN you should try to get only the good news, not the bad noose. To win at WAVY NAVY, let your computer generate those 30-foot waves on its own for a while. It is easier than to beat a C-stick by four. — Ylimaki

Jim Butterfield's cats are now famous for their attraction to his computers and CAThode ray tubes. I also have a feline ON LINE, but mine has taken to sprawling on my printer. Did I mention that she was a TABby...a little spaced out at times? — Ylimaki

A

Accounting 121
Accumulator 89,92,98,99
Acronym 187
ADC (also see analog) 190
Addons 137
Address 18,265
ADSR 189
Advantage Computer
Accessories of Mississauga 170,171
Aim 65, 13,194
Airco Central Research
Laboratories 209
Algorithm 28,191
Alpha 169
Alphanumeric 187
Amplifier 193
Analog (see also D/A) 186
Analodigital 187,189
Analysis 105
Analytical Geometry 105
AND 38,100
APL 121
Apple 5,13,38,40,60,62,105,106,
107,139,188
Apple Avocation Alliance 141
Arrays 22,67,104
ASCII 13,78,124,139,213,223,265
Ashcraft, Cliff 194
ASM 118
Assembler 88,97,121,262,267
Assembly Language 75,89,90,98
Atari 13,14,40,188,191
Attack 112
Audio 189
Audio Waveform 190
Aurora Software 239
Autorun 171

B

Background Colours 109
BAM 134,135,139,144
Baron 187
BASIC 9,13,18,23, 24,28,40,47,
48,51,52, 63,68,75,76,78,80,81,89,
93,94,95,104,109,122,123,124,
162,191, 227,236,262,264,266,267
Basic Aid 22,23,68,104,162
Basic Listings 151
Basic 40 23
Bass 186,190
BBS 212,213,217
Bell Labs 193
Bennett, Barb 2
Bennett, Chris 3,138
BEQ 90
Bidirectional 53
Binary 31,38,124
Binary Notation 78,86,98
Bitmapped 24
Bits 28,30,38,42,47,92,95,140,192,202
Bits And Bytes 236
Bit Mapping 28,30
Block checksum 213
Blocks 23,24,30,61
Block Availability Map 134,135,139,144
BMB Compuscience 3
BNE 89,90
Bonnycastle, Michael 3
Boolean Functions 68
Booster 217
Border 261
Branching Loops 89
Branch Commands 92
Brother 151
Buffer 53, 144,180,205,222,230,236
Buffer Transmitter 54
Built-in Clock 228
Bulk Eraser 138
Bulletin Boards 212,214,215,218
Bulletin Board Systems 138
Bus Transceivers 53
Buti 162
Butterfield, Jim 2,3,12,21,24,98,
138,142,236
Bit 101
Bytes 28,31, 38,42,87,92 ,93,97,
98,100,107,109,162,191,194,198,
204,205,262,265,267
Byte Magazine 193

C

Calculate 190
Calculations 169,190
Calculator 174
Callapple Magazine 62
Canada 170,171
Canadian Educational
Microcomputer 227
Cardco 62,149
Carriage Return 47
Carrier 217
Cartridge 110,122,124,125,154
Cassette 23,134,137,178,190
Cassette Buffer 22
Cbasic 121
CBM 3,12,134,175,181
CBM 2001 137
Censorship 215
Centronics 149,150
Chamberlin, Hal 187,204
Characters 62,63,107,151
Character Set 30
Character Strings 109
Charsetup 75,76
Chip 143,181,188
Chords 186
Chroma Noise 25
CHUG 162
Circuit 60,188,189,217
Circuit Board 228,230
CLC 193
Cleaning Kit 141
CLI 76
Close 109
CLR 30,94
CLR/home 171,228
Club 2,105,267
CMOS 53
CMP 92
COBOL 121
Code 61,63,67,68,122,123,175
Codes 31,62,116
Code Routine 191
Coding 67,80,188
Colour 91,122,186,222,261,264
Comdex 11
Commander 2
Commands 175,178
Commodore2,24,36,60,106,120,121,139,
150,151,171,178,182,218,223,226,228
Commodore shift 63
Commodore Canada 107
Commodore Computers 120,189
Commodore Disk Format 123
Commodore International Ltd 5,8,46
Commodore Key 30
Commodore Logo 107
Commodore Magazine 2
Commodore Programmer's Aid 162
Commodore Programmer's
Reference Guide 25,107
Commodore Tractor Printer 154
Commodore-64 (see also C-64)
Commodore-64 5,10,18,28,30,36,38,40,
62, 24,143,144,171,175,188,175,105,
106,107,196,197,198,201,261,262
Commodore-64 User's Guide 24,62
Common Music Notation 186
Compatability 175
Compile 104
Compiler 104,189
Compuserve 123,218
Compute 12,21,24,86,100
Computer-assisted Music 186
Computer Literacy 9
Computer Software 170
Computer Software Associates 171
Compute's Gazette 2
Com File 119
Configurations 186
Connector 149
Constructional Mode 186
Converter 189
Coupler Access Port 219
Covitz, Dr. Frank 194,209
CPM 109,116,117,118,120,122,123,124
Crimando, Bill 75
CRT 121
CRT Mode 182
Ctrl 116
Cursor 22,180,236,261
Cursor Control 47
Cymbals 186
C Itoh 150
C-64 (see also Commodore-64)
C-64 24,25,28,62,83,88,89,90,95,99,
100, 104,120,121,122,123,124,149,
169,170, 182,187,188,210,219,222,
226, 228,230,236,262,264
C-64 Link 130
C-64 Programmer's Reference
Guide 24,62

D

D/A (see also digital and analog)
D/A 189,190,191,192
D/A Synthesis 194

Daisy Wheel 148,150,151,153
Data 61,63,78,79,80,81,93,104,123,213, 228,236,262
Data Base 121,145,158,159,162
Data Editor 267
Data Files 134,176
Data General 13,14
Data Monitor 181
Data Products 153
Data Storage 267
Data 20 16K Video Pak 181,182
Data20 131
Db 190
Dbms 162
DCP 101
Dealer Disk 236
Debugging 61,67,105,109,227,261, 264,267
Debugging Aids 109
DEC 60
Decay 112
Decimal 38,230,261,265
Decimal Code 62
Decoding 25
Defining 175
Definite Pitch 186
Delayed Playback 190
Delete 25,62,236
Delphi's Oracle 145
Del Key (see also insert) 116,236,264
Demagnetize 138
Demo 106
Design 189
DEX 90
DEY 90
Diablo 150
Digital 186,189
Digital Audio 189,190
Digital Computers 9
Digital IC Hex Inverter 230
Digital Research 119,122
Digital System 187
Diode 230
Direct Access 135
Direct Backup 144
Disassembler 267
Disctape 182
Disk 23,36,40,104,106,107,116,117, 118,119,120,122,123,124,134,139, 153,169,175,178,179,180,181,222, 223, 236,262,265,266
Diskette Format 123
Disk Commands 23,109
Disk Drive 123,137,138,143,152,178,190
Disk Errors 138
Disk Error Channel 143
Disk Id Mismatch Errors 142
Disk Initialization 109
Disk Master 145
Disk Operations 109
Disk Rules 138
Disk Status 261,262,264
Display Manager 131
DOS 222
DOSO-wedge 143
Dosprotected 180
Dot Matrix 148,150
Doubledensity Disks 138,140
Doublesiding 138
Download 217,222
Download Buffer 223
Dual Cursor 171
Dwell 191

E

Ear Drum 186
Easton, John 3
Easyscript 36
Editor 51,222
Edit Mode 179,180
Education 105,226,261,262,264
Educational 236
Electrical Audio Signal 189
Electronics 2001 3
Electronic Mail 212
Electronic Matrix 169
Emulator 24
Enabler 75,76
Enable Control 54
Entry 171
Entry Point Address 93
EPROM 97,131,267
Epson 62,148,150
Escape 171
Executive 64 7
Expander Port 181,183
Expansion Board 54
Expansion Connector 54
Expansion Slot 181
Exponentiation 25

F

Fat 40 21
File 116,117,119,125,134,169,180,262
Filename 118,264
Filesaver 227
File Scratching 109
Filter 187,189,191
Flags 22,24,42,67,236
Flashkiller 75,76
Flex File 145
Floating Point Arithmetic 25,188
Floppy Disks (see disks)
Flowchart 68
Flowcharting 227
Format 104,119,153,169,78,183
Formatted Listing 62
Formatting 109
Formulae 169,170
FOR/NEXT loops 61,62,93,236
Fortran 109,121,122
Fountain, Mr. Laurie 107
Frequencies 186,187,190,192,194,197, 198,202,205,206
FRE (0) 30
Function 32,104,106,264,265,266
Functions 153,175,176,182,

G

Games 109,226
Gemini 10x 169
Gemini 15x 169
Geometrical 105,109
Get Statement 22,47,82,236
Gosub 47,61,68,78,109,236
GOTO
Statements 47,63,67,68,78,109,236
Gould, Irving 4
Graphic Characters 24,62,105,107
Graphics 18,28,30,105,106,107,151,236
Graphs 236

H

Hackers 226,227
Hardware 13,181,187,188,189,228
Hardware Interrupt Vector 21,22
Hard Disk 227
Harmonic 187,194
Harrison, Howard 143
Hayden Books 187
Hertz 186
HES 183
Hesware 169
Heswriter 183
Hex 118,119,124,261,262,264,265
Hexadecimal 98,119,262
Hex Files 124
HIFI Magazines 190
High Densities 119
High Resolution 28,60,107,109
High Resolution Graphics 60,109
Home 236
Hook, David 3
Hub Ring 140
Hybrid System 186
Hyperboles 105
Hyszka, Michael 3
Hz 186

I

IBM 60
IBMpc 169
IBM Selectric 150
IBM 327X 76
IC Board 228,230
IC Socket 228,230
Ideagrams 60
IEEE 123,130
IEEE488 Buss 151,162
IF THEN 61,92,236
IF THEN GOTO 62
Implementing 191
Increment 261,265
Indefinite Pitch 186
Indents 61
Indent Lines 63
Indeterminate Pitch 186

Indicates 171
Infinite Loop 47
INFOGLOBE 212
Initiation interrupt 187
Inline Coupler 219
Input 24,47,61,63,78,79,80,82,236
Input output 169
Input Connector 219
Insert 25,109
Insert/delete Key (see also delete key) 182
Inserts 62
INST/DEL (see also delete key) 171
Instructional Mode 186
Integer 25,28,104
Integer Basic Code 104
Integer Exponentiation 25
Integer Format 169
Integrated 105
Intel 122
Intellivision 13
Interactive Mass 60
Interactive Program 61
Interpreter 107
Interpretive Assembler 119
Interrupt 30
Intraspecies Communication 60
Inverters 230
I/O 122,189,267
I/O mapped 122
I/O Port 36

J

JMP 92,94
Joystick 38,106,107,152,188
JSR 94,95
Jumpman 46
Justification 151

K

Kernal 36,75,122,267
Keyboard 169,188,226,236
Keyboard Buffer 22
Keyboard Commands 130
Keystroke 171
KHz 190
Kilobaud Magazine 230
Kilobytes Of Memory 123
KIM 12,13
Knockout Port 219

L

Language 105
Language Coding 262
LAX 101
LDA 89,90,98
LDX 90,98
LDY 90,98
LED 142,230
Letterquality 148
Level 189
Light Pens 152
Linked File 179
Lisa 38
List 23,62,104,107
Listing 61
Lists 107
List Formatter 62,63
Load 63,104,109,124,267
Loaded 119,175
Load/run 122
Lobyte/hibyte 90,98
Logical File Number 134
Logo 106,115
Logo Newsletter 107
Loop 93,191
Lower Case 24,169
Lowpass 189,191
Low Power Schottky Ls 53

M

Machine 89
Machine Code 19,68,118,266
Machine Cycles 109,109,109
Machine Language 18,22,24,32,39,76,86, 104,122,188,191,236
Machine Language Merge 261
Machine Language Monitor 124,236,267
Machine Language Programs 106,125,223
Machine Language Routines 19,93,106
Machine Language Subroutine 36,40
Macrotime 188
Madison Zram Board 120
Magazines 105,107
Magic Desk 163
Mail Merge Program 182
Manager 159
Mannesmann 150
Manouvers 187
Mansfield 154
Manual 18,106,109,122,123,178,222
Margin 153,183
Master Program 180
Mathematical 105,191
Matrix 169
Megabyte 213,227
Melody 188
Memory 22,62,63,105,107,110,116,120, 122,124, 169,170,175,190,236,262, 264,265,266,267
Memory conserving 169
Memory Limitation 106
Memory Locations 30,31,38,107,120,265
Memory Map 21
Menu 171
Menu Driven Program 222
Message System 212
Metaphors 163
MHz 190,192
Micro 13,186
Microcomputers 46,105,106,128, 140,150,169,179,188,212,213,228
Micromon 23,267
Microprocessors 53,54,98,99,120, 122,187,190,195,202,204,209
Microsecond 188,191
Microsoft Basic 162
Microsoft Multiplan Version 106 169
Microtech 179
Microtext Editing 179
Microtime 188
Midnite Software Gazette 2
Minicomputer 13,218
MIT 105
Mite 124
ML (see machine language)
Mnemonics 118
Modems 121,124,152,212,213,214 219,220
Modem80 217
Modified 192
Modular Programs 81
Modulation 187
Monitor 227,264,267
Monitor/Assembler 99
Morepower 143
More routine 92
Mos Programming Manual 100
Mos Technology 4,5,209
MTU 188
MTU130 Computer 189,195,204
Multi 109
Multicolour 28,30,109
Multiplan 169
Multiple 191
Music 105,106,112
Musical 187
Music Compiler 195
Music Keyboard 204
Music Synthesis 188
Music Synthesizer 199
Music Tutor 186

N

NEC 150
NEC Spinwriter 150
Nesting Of Loops 67
NEW 32
Newsletter 107
NEXT 62
Noise 198
Nonarray Variables 109
Noncommodore 169
Noncommodore Printer 62
Noninteger 192
NOPS 191
NPN Transistor 230
Numerical 190

O

Object Code 119
Oddnumbered Harmonics 187
Ohio Scientific C8P 188,189
Okidata 148,150
Olivetti Praxis 151

On/Off Switch 139,181
Opcodes 100,101
Open 104,109
Operating System 120,122
Operation 124
Oracle 159,160
Oscillators 186
Oscilloscope Photo 194
Outputs 107
Outputting 191
Output Files 175
Overflow 191

P

Painted 109
Palette 187
Paperclip 159
Paraboles 105
Parallel Interface 151
Parameters 109,265
Pascal 121
Pcb 51
Peek 18,21,38,107,109,110,236
Peeking 21
Perimeters 148
Peripherals 118,121,130,152,183
PET 5,13,18,22,23,24,30,60,74,83,95, 97,100,104,148,162,188,194,213,217, 228,230,236
PET/CBM 21,52,82,89,90,98,99,175
Petscii 222
Petspeed 223
Phonemes 51
Pi 62
Piano 186,187
Pico Farad 53
Pin 149
Pitch 197,212
Pitches 186
Pixels 25,28,60,109
Playback 190
Plotted 24
Plotting 105,109
Plugging 0
Pl1 121
Poke 18,19,21,22,23,48,62,98,107,124, 125,197,228,230,236
Poked 32
Pokes 23,24,51,75,76,93,110,188
Pokes 109
Ponzo 236
Ponzo tutor 236
Port 189
Powaid 143
Power 22,143
Power Play 2
Practicalc 64 170,171
Print 19,83,109,151
Printer 62,148,149,150,153,172, 174,175,222
Printers 121,150,151,154,175
Printer Interface 182
Printer Ribbon Cassette 154
Printouts 169,228
Print File 63
Print Statement 47
Processors 193
Program 175,176,179,180,187,222
Programmable 189
Programmable Characters 30
Programmer 188
Programmer's Reference Manual 100
Programming Aids 109
Program Files 134
Program Memory 78
PROM 124
Proportional Type 151
Pseudo code 227
PTRF 192
Publications 2
Pulse Wave 187
Pyramid 186

Q

Quadraphonics 189
Qume 150
Quotient 28

R

Radiation 128
Radio Shack 105,150,210
Radio Shack Colour Computer 109
RAM 7,19,30,36,46,54,75,76,86,97,110, 117,120,204,227,267
Rarefraction 186
Raster Scan Interrupt 24
Read 78,79,80,81
Reading 236
Read/Write Head 134,143
Read Error 119
Rectangles 109
Reedy hollow 187
References 107
Relative Record Files 23
Release 112
Relocating 94
REM 38,61
RENAME 116
Repetitive 191
Resistor 230
Restore 21,30,80,81,104,266
Retarded Children 105
Return 180,212
Reverse Off 25
Reverse On 25
Rewritten 191
Rf Modulator 51
Rhythm 188
Ribbons 151,154
Richvale Telecommunications 3,130,236
Ring Modulation 202
Robot 7
Robotics 7
ROM 19,23,24,30,36,52,75,86,97, 110,175,179,181,183,222 179,181,183,222
ROM Rabbit 137
ROM Routines 21
Routine 61,153,261,264
Routines 24,106,122,262,267
Rs232 121,124,151,153,182
Rs232 Port 123
Rs232 Printers 182
RTC (see Richvale)
RTS 91
Run/Stop 21,266
Run/stop/restore 30,32
Run And Poke 2

S

Salkind, Neil J. 12
SAVE 104,116,124,228,235
Sawtooth Or Ramp Wave 187
Scan 191
Scanning 191
Schools 226
School Systems 105
Scientific Software 123
Score 187
Screen 91,261
Screen Color 28
Screen Editing 162
Secondary Addresses 63
SEI 76
Seitz, John K 12
Semantic 67
Sequential 222
Serial 182
Serial Interface 151,218
Sheet Music 189
SID 188, 196,197,198,199,201,202,203, 205, 206
Silicon 120
Sinclair ZX81 13
Sine Wave 187
Skinny 40 21
Slug 174,213
Smart 64 Terminal 222,223
Smith Corona 148,151
Smith, Dennis 47
SM-KIT 162
Snare Drums 186
Software 51,120,125,142,181,187,188, 189,190
Software Automatic Mouth 40
Software Commands 130
Sorted 171
Sound 228
Sounds 227
Sound Generators 186
Sound Synthesis 60
Sound Synthesizer 186
Source 218
Source, The 212
Spacing 62
Speakeasy 51
Speaker 51
Speakers 150
Special Characters 104
Speech Synthesizers 51
Spelling Checker 179,180,181
Spellmaster 179,180,181
Spellpro 179
Spot Check 187
Spreadsheet 151,170,171

Sprites 30,42,105,106,107,236
Square Wave 187
STA 89,90,98
Start Address 93
Start Of BASIC 18
Star Micronics 150
Status Register 90,236
Stepper-motor Hiccups 142
Stop 63
Stop/restore 63
Storage 137,178,262
Store 89
String Array 80
String Variables 47,236
Structured Flowchart 67
STX 90
STY 90
Submenu 172
Subroutine 19,25,38,40
Subroutines 24,61,67,74,78,81,109,112
Subscripting 25
Supermon 88,90,97,98
Supermon64 236
SuperPET 120,218,227
Superscript 179,180
Superspell 179,180,181
Sustain 112
Swapping 169
Switching 219
SYM 119
Synchronization 112,202
Synch Marks 142
Syntax Errors 79
Synthesis 190,192
Synthesize 190
Synthesizer 201
Synthesizers 196
Synthesizing 191
SYS 18,32,93,99,264
Sysop 212,213,217
Sysops 214, 213
S02 Clock 54

T

Tabs 25
Tabulated 191
Tape 187,222,262
Tapes 175
Tape Drive 174
Technology 105,149
Telecomputing 218
Teletypes 116
Telidon 227
Telidon Graphics 227
Tempo 192,194
Texas Instruments 14,105,106,109
Text 107,109,116,153
Text editing Program 179
Text Editor 179,180,181
THEN 67
Theoretically 190
Three dimension 107
Timbre 186
Time Clock 21
Time Delay 61
Time Delay Routine 61
TINYMON 88
TI Programmer 124
Tone 191
Tones 205
Toolkit 162
TPA 124
TPUG 239
Track 116
Trackball 227
Tractor 149
Tractorfeed Mechanism 151
Tramiel, Jack 4,8
Transactor 2
Transceiver 54
Transistor 230
Triangle Wave 187
Tristate 53
TRS80 60,149
TTL Load 53
Tune 186
Turtle 105,107
Turtle Graphics 105
Turtle News 107
Typewriter 179

U

Up/Down Crsr Key 262
Upgrade Pet 143
Upload 222
Upload Files 222
Upper and lowercase Mode 63
Upper/lower 19
Upper/lower Case 21,22
Upper Case 18,24,169
User 171
User-defined Character 75
User Code 212,217
User Log 212
User Manual 31
User Port 121,189,228
USR 18
USR File 139
Utilities 68
Utility 106,262,267

V

Variable 24
Variables 104,109,236
Variant 187
VAX 218
Verbatim 141
Vibration 186
Vibrations 186
VIC 5,12,13,18,40,46,47,48,51,52,53,54, 60,75,82,86,89,90,91,94,95,99, 100,121,131,137,149,153,174,175 181,182,183,188,209,210,214,219, 226,228,230,262,267
VICaid 209,210
VICmodem 218,219,220
VICmon 88
VIC Audio Interface Device 209
VIC 1515 153
VIC 1525 169
VIC Tiny Aid 93
Video 63
Video Monitor 131
Video Pak 181
Visicalc 120,179
Voice 106,112
Voltage 189
Volume 196
Volume Level 112
Votrax 51

W

WAIT 18
Waveforms 186,187,188,189,191,192, 193,197,198,202
Waveform Harmonics 186
Waveform Tables 194
Weaker 186
Wedge 40,222,265
Wedge64 261,262,264
Whistle 187
Wordpro 115,120,178,179,180,181,223
Word Manager 181,182
Word Processing 121
Word Processor 36,46,131,174,175, 176,178, 182,222,223
Workhorse 171
World Of Commodore 7
Wraggett 188
Wraparound 191
Write protected 140
Write Error 119

Z

Zero 31,191
ZRAM 121
Z80 118,122,123,124,188,190
Z80 Assembler 118

1515 222
1525 222
1541 122,123
1541 Backup 144
16 bit 7
16k 188
2.0 ROMs 2001 218
22 column Screen 13
32k 188
4000 Series Pet 121
40 column Pet 90
40 column Screen 123
4.0 ROMs 18
4000/8000 Series 120
4040 Dual Drives 138,180
4040/1541 Format 122
64 User Manual 31
6502 Microprocessor 53,120,267
6510 120,122
6809 120
8 bit 191
8k Pet 12,13
8080 Z80 120
80 column 123
80 column Screen 121
8023 154
8032 89,178
8080 Microprocessor 120,122
8085 122

THESE DISKETTES ARE ALL DESCRIBED IN DETAIL IN THIS BOOK

The TORPET Best Programs

1. Diskette for the C64/VIC (Includes all the lengthy programs in this book plus many more). **The one diskette to get if you get only one diskette.**

10 Best TORPET Diskettes

1. Games#1.C	C64
2. Games#2.C	C64
3. Games#3.C	C64
4. Torpet Manual.C (The Programs in the Commodore Manual & User's Guide)	C64
*5. Torpet Educ.C (Teaches you how to program)	C64
6. Miscellaneous.C	C64
* 7. Utilities-64.C	C64
8. Games.V	VIC
9. Educa/Simula.V	VIC
*10. Misc/Utilities.V	VIC

***The Second Most Important Diskettes to Get!**

Education (E) Series C64/PET

1. AA — Administration
2. BA — Business
3. BB — Business
4. BC — Business
5. CA — Computer Science
6. CB — Computer Science
7. D1 — Commodore 64
8. D2 — Commodore 64
9. D3 — Commodore 64
10. EA — English
11. EB — English
12. EC — English
13. ED — English
14. EE — English
15. EF — English
16. EG — English
17. EH — English
18. EI — English
19. FA — French
20. GA — Games
21. GB — Games
22. GC — Games
23. GD — Games
24. GE — Games
25. RA — Geography
26. RB — Geography
27. RC — Geography
28. JA — Language
29. LA — Language and Problem Solving
30. LB — Logic and Problem Solving
31. LC — Logic and Problem Solving
32. LD — Logic and Problem Solving
33. MA — Mathematics
34. MB — Mathematics
35. MC — Mathematics
36. MD — Mathematics
37. ME — Mathematics
38. MF — Mathematics
39. MG — Mathematics
40. MH — Mathematics
41. MI — Mathematics
42. MJ — Mathematics
43. MK — Mathematics
44. ML — Mathematics
45. MM — Mathematics
46. MN — Mathematics
47. MO — Mathematics
48. NA — Music
49. PA — Physical and Health Education
50. SA — Science
51. SB — Science
52. SC — Science
53. SD — Science
54. SE — Science
55. SF — Science
56. SG — Science
57. TA — Technology
58. UA — Utilities

Any of these Diskettes are available from:

TORPET DISKETTES
Horning's Mills
Ontario. L0N 1J0
Canada

or

TORPET DISKETTES
1 Brinkman Ave.
Buffalo, N.Y.
14211, U.S.A.

Single Diskettes are $10 each (send cash, cheque of money order), or $350 for all 69 diskettes.

1 2 3 4 5 140620 (book) 140621 (book & disk) 88 87 86 85 84

Descriptions and multiple sources for over

1000 FREE PROGRAMS

for the

COMMODORE 64* & VIC-20*

CHIPP cartoon strips that explain BASIC for the

BEGINNING PROGRAMMER

Articles for the

INTERMEDIATE PROGRAMMER

on MACHINE LANGUAGE COMPILERS AND OTHER LANGUAGES

Complete MEMORY AND CROSS REFERENCE MAPS printed

IN LARGE TYPE

for the

ADVANCING PROGRAMMER

along with complete instructions for
MICROMON, WEDGE-64 and BYTEDS
The three most powerful public domain utilities.

A thorough compilation on

COMMODORE COMPUTER MUSIC

by the foremost authorities.

User documentation for the

200 BEST GAMES, UTILITIES

and other programs in the public domain.

Articles that give simple and complete explanations of TELECOMMUNICATIONS, DISKS, PRINTERS, TAPE, WORD-PROCESSING, DATABASES, SPREADSHEETS

PLUS MUCH MORE

THE ONE BOOK EVERY COMMODORE OWNER SHOULD HAVE.

Copp Clark Pitman Ltd.

ISBN 0-7730-4080-3