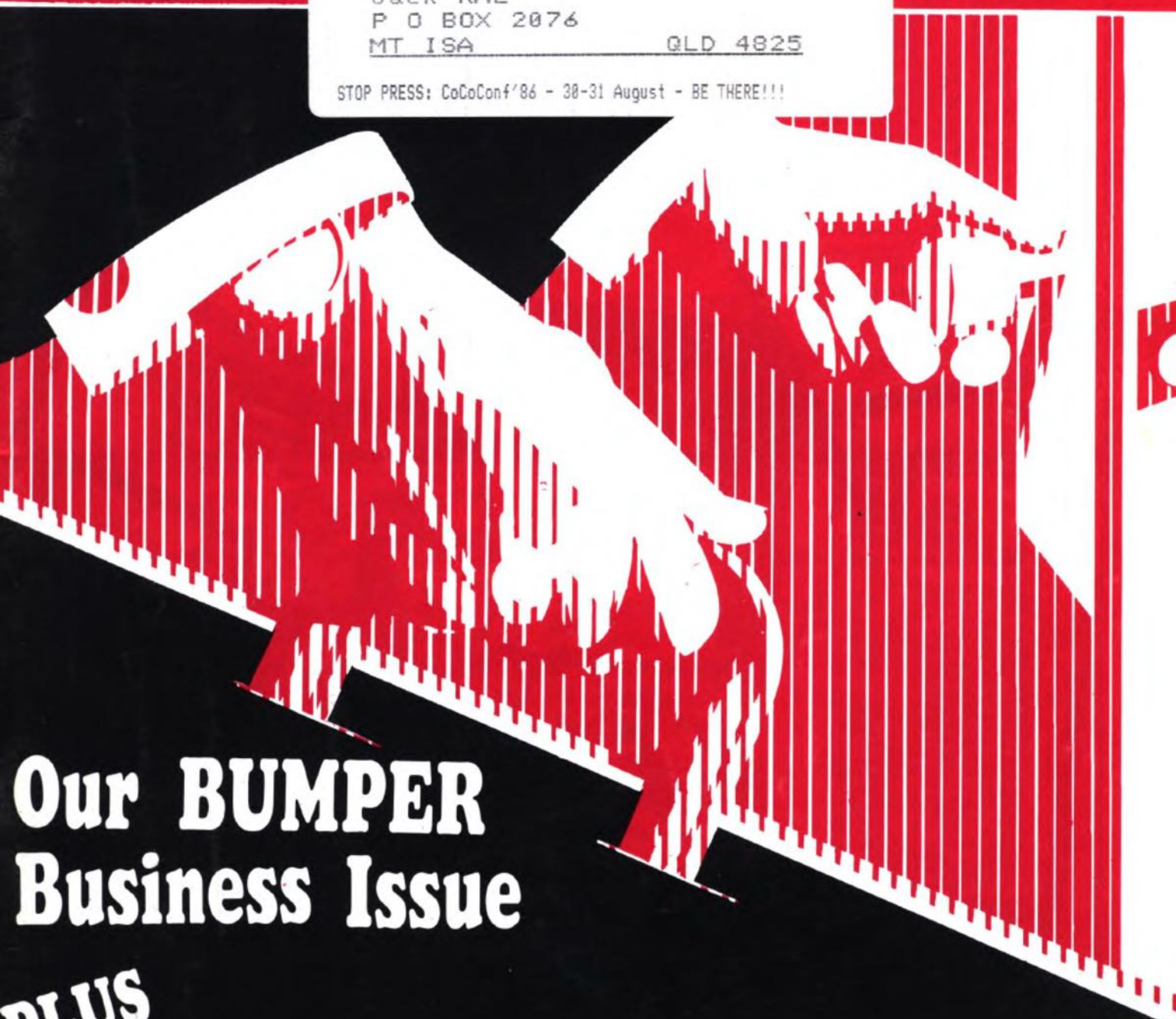
For your TANDY Color Computer

Registered by Australia Post Publication No QBG 4009 PNG K4.95 NZ $5.20 $4.50

AUSTRALIAN RAINBOW

April, 1986 No.58

STOP PRESS: CoCoConf'86 - 30-31 August - BE THERE!!!



Our BUMPER Business Issue

PLUS

Advice for disc tinkerers
Animation OS 9 Recipe Maker
and GAMES

COCOCONF '86

WHAT'S HAPPENING:- Tutorials on Advanced BASIC, Basic BASIC, Educational use of computers, OS9, MS DOS/GW BASIC, FORTH, The CoCoConnection HARDWARE mods include:- high K upgrades (128,256,512,1mb) AND THAT'S JUST SATURDAY!! Saturday night we have our dinner and prize session. (this is included in your registration fee) SUNDAY continues with MORE tutorials plus the opportunity to browse/buy the large range of software and hardware available for the CoCo and T1000. There will be lots of bargains!

SPEAK UP!:- Now is your chance to suggest your ideas for any tutorials we may not have mentioned. (participants only).

LOCATION:-
SEAGULLS RUGBY LEAGUE CLUB TWEED HEADS.

DATE:- Sat 30th & Sun 31st August 1986.

REGISTER NOW!!
We can only accept a limited number of people this year. DON'T MISS OUT! on a top weekend of FUN, FRIENDSHIP and LEARNING.

Name: .....

Address: .....

Phone: .....

No. People attending: .....

$39.95 per person/1st family member
$20.00 per additional family member
$9.95 dep. balance by 15/8/86
Cost includes:- tutorials, dinner Sat. night, morning and afternoon tea.

Tutorials likely to attend: .....

.....

Please find enclosed:
chq/money order/bankcard/visa/mastercard

Card No. .....

Signature: .....

GOLDSOFT

Hardware & Software for your TANDY computer.

HARDWARE

The CoCoConnection: Connect your CoCo to the real world and control robots, models, experiments, burglar alarms, water reticulation systems — most electrical things. Features two MC 6821 PIAs; provides four programmable ports; each port provides eight lines, which can be programmed as an input or output; comes complete with tutorial documentation and software; supplied with LED demonstration unit. Switchable memory addressing allows use with disk controller or other modules via a multipack interface; plugs into Cartridge Slot or Multipack, uses gold plate connectors; a MUST for the hardware designer and debugger!		$206.00
Video-Amp: Connects simply to your CoCo to drive a Colour or Mono monitor.	With instructions With instructions and sound	$25.00 $35.00
The Probe: A temperature measuring device which attaches to the joystick port of your CoCo or T1000, or to the joystick port of your CoCo Max. Comes with programs to start you thinking, and is supported monthly in Australian CoCo magazine.	With amplifer	$49.95 $59.95

SOFTWARE

Magazines: Australian Rainbow Magazine — THE magazine for advanced CoCo users! Australian CoCo Magazine — THE magazine for the new user of a Tandy computer. Also suits owners of CoCos, MC 10s, Tandy 1000s, 100s, 200s & 2000s.		
Back Issues: Australian Rainbow Magazine. (Dec '81 to now.) Australian CoCo Magazine. (Aug '84 to now.) CoCoBug Magazine. For CoCo — usually 8 programs in each magazine. (Sep '84 to Oct '85) Australian MiCo Magazine. For Tandy MC 10 computers. (Dec '83 to Jul '84) Australian GoCo Magazine. For Tandy Model 100 users. (Jul '83 to Jul '84)	Feb '85 through through Jan '85 each each each each	$4.80 $2.50 $3.45 $1.00 $2.00 $1.50
CoCoOz, on Tape or Disk: The programs you see listed in Australian CoCo Magazine are available on CoCoOz! No laborious typing — just (C)LOAD and Go! Back issues of CoCoOz are always available	Each Tape Subscription, 6 months 12 months Each DISK Subscription on disk, 12 months	$9.50 $42.00 $75.00 $10.95 $102.50
Rainbow on Tape, or Disk: Australian. The programs you see listed in Australian Rainbow Magazine are available on tape. A boon if you don't understand the language! American. We also supply the programs found in American Rainbow on tape. Please specify either Australian or American.	Each Tape Subscription, 12 months **NEW** for 1986 ONLY Each DISK Subscription on disk, 12 months	$15.00 $144.00 $15.00 $172.00
MiCoOz: The programs in the MiCo section of Australian CoCo Magazine. (For MC 10 computers only) Back issues of MiCoOz are always available.	Each Tape Subscription, 12 months	$9.50 $75.00
GOLDDISK 1000 — programs from 'softgold' for your Tandy 1000 on disk		$10.95
CoCoLink: CoCoLink is our Bulletin Board which you can access with any computer if you have a 300 baud modem and a suitable terminal program. There is a free visitor's facility, alternatively membership entitles you to greater access of the many files available. We can also be contacted through Minerva (OTC) and Viatel (Telecom).	Subscription to CoCoLink, 12 months	$29.00
Books: HELP: A quick reference guide for CoCo users. BYTE: Guide for new CoCo users. MiCo HELP: A quick reference for owners of MC 10 computers.		$9.95 $4.00 $9.95
Othello: by Darryl Berry The board game for your CoCo.	Tape 16K ECB	$15.95
Say the Wordz: by Oz Wiz & Pixel Software Two curriculum based speller programs for your Tandy Speech/Sound Pack.	Tape 32K ECB	$39.95
Bric a Brac: Blank tapes . . . 12 for $18.00 or $1.70 each. Cassette cases . . . 15 for $5.00. Disks . . . (they work!) $3.50 each or $28.95 per box of 10.		

HOW TO ORDER

Option 1: Use the subscription form in this magazine.
Option 2: Phone and have ready your Bankcard, Mastercard or Visa number.
Option 3: Leave an order on Viatel, Minerva or CoColink, but be sure to include your Name, Address, Phone Number, Credit Card Number and a clear indication of what you require, plus the amount of money you are authorising us to bill you.

RAINBOW Info

How To Read Rainbow

Please note that all the BASIC program listings you find in THE RAINBOW are formatted for a 32-character screen — so they show up just as they do on your CoCo screen. One easy way to check on the accuracy of your typing is to compare what character "goes under" what. If the characters match — and your line endings come out the same — you have a pretty good way of knowing that your typing is accurate.

We also have "key boxes" to show you the *minimum* system a program needs. But, *do* read the text before you start typing.

Finally, the little cassette symbol on the table of contents and at the beginning of articles indicates that the program is available through our RAINBOW ON TAPE service. An order form for this service is on the insert card bound in the magazine.

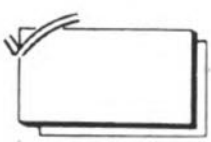
What's A CoCo

CoCo is an affectionate name that was first given to the Tandy Color Computer by its many fans, users and owners.

However, when we use the term CoCo, we refer to both the Tandy Color Computer and the TDP System-100 Computer. It is easier than using both of the "given" names throughout THE RAINBOW.

In most cases, when a specific computer is mentioned, the application is for that specific computer. However, since the TDP System-100 and Tandy Color are, for all purposes, the same computer in a different case, these terms are almost always interchangeable.

The Rainbow Check Plus



The small box you see accompanying a program listing in THE RAINBOW is a "check sum" system, which is designed to help you type in programs accurately.

Rainbow Check PLUS counts the number and values of characters you type in. You can then compare the number you get to those printed in THE RAINBOW. On longer programs, some benchmark lines are given. When you reach the end of one of those lines with your typing, simply check to see if the numbers match.

To use *Rainbow Check PLUS*, type in the program and CSAVE it for later use, then type in the command RUN and press ENTER. Once the program has run, type NEW and ENTER to remove it from the area where the program you're typing in will go.

Now, while keying in a listing from THE RAINBOW, whenever you press the down-arrow key, your CoCo gives the check sum based on the length and content of the program in memory. This is to check against the numbers printed in THE RAINBOW. If your number is different, check the listing carefully to be sure you typed in the correct BASIC program code. For more details on this helpful utility, refer to H. Allen Curtis' article on Page 21 of the February 1984 RAINBOW.

Since *Rainbow Check PLUS* counts spaces and punctuation, be sure to type in the listing exactly the way it's given in the magazine.

```
10 CLS:X=256*PEEK(35)+178
20 CLEAR 25,X-1
30 X=256*PEEK (35)+178
40 FOR Z=X TO X+77
50 READ Y:W=W+Y:PRINT Z,Y;W
60 POKE Z,Y:NEXT
70 IFW=7985THENB0ELSEPRINT
   "DATA ERROR":STOP
80 EXEC X:END
90 DATA 182, 1, 106, 167, 140, 60, 134
100 DATA 126, 183, 1, 106, 190, 1, 107
110 DATA 175, 140, 50, 48, 140, 4, 191
120 DATA 1, 107, 57, 129, 10, 38, 38
130 DATA 52, 22, 79, 158, 25, 230, 129
140 DATA 39, 12, 171, 128, 171, 128
150 DATA 230, 132, 38, 250, 48, 1, 32
160 DATA 240, 183, 2, 222, 48, 140, 14
170 DATA 159, 166, 166, 132, 28, 254
180 DATA 189, 173, 198, 53, 22, 126, 0
190 DATA 0, 135, 255, 134, 40, 55
200 DATA 51, 52, 41, 0
```

Using Machine Language

Machine language programs are one of the features of THE RAINBOW. There are a number of ways to "get" these programs into memory so you can operate them.

The easiest way is by using an editor/assembler, a program you can purchase from a number of sources.

An editor/assembler allows you to enter mnemonics into your CoCo and then have the editor/assembler assemble them into specific instructions that are understood by the 6809 chip that controls your computer.

When you use an editor/assembler, all you have to do, essentially, is copy the relevant instructions from THE RAINBOW's listing into CoCo.

Another method of getting an assembly language listing into CoCo is called "hand assembly." As the name implies, you do the assembly by hand. This can *sometimes* cause problems when you have to set up an ORIGIN statement or an EQUATE. In short, you have to know something about assembly to hand-assemble some programs.

Use the following program if you wish to hand-assemble machine language listings:

```
10 CLEAR200,&H3F00:I=&H3FB0
20 PRINT "ADDRESS:";HEX$(I):
30 INPUT "BYTE";B$
40 POKE I,VAL("&H"+B$)
50 I=I+1:GOTO 20
```

This program assumes you have a 16K CoCo. If you have 32K, change the &H3F00 in Line 10 to &H7F00 and change the value of I to &H7FB0.

The Crew

Founder Greg Wilson
Publishers Graham & Annette Morphett
Managing Editor Graham Morphett
Accounts Annette Morphett
Assistand Editor Sonya Young
Advertising Tracey Yapp
Art Jim Bentick
Sub Editors
Assembly Language: Kevin Mischewski
MC-10: Jim Rogers
softgold: Barry Cawley
Forth: John Poxon
OS-9: Jack Fricker
Special Thanks to
Brian Dougan, Paul Humphreys, Alex Hartmann, Michael Horn, Darcy O'Toole, Martha Gritwhistle, Geoff Fiala, John Redmond and Mike Turk.

Phones: **(075) 51 0577 Voice**
(075) 32 6370 CoCoLink
Deadlines:
7th of the procedding month.
Printed by:
Australian Rainbow Magazine
P.O. Box 1742
Southport. Qld. 4215.
Registered Publication QBG 4009.

PRINT #-2

Before you read any further, if you are a subscriber, please check your mailing label.

We've been working on some of the glitches in our database, and as part of this task, have reallocated many numbers.

As a reminder, your subscriber's number is the number in the top left hand area of the mailing label. Then follows the renewal dates for the various items to which you subscribe. We replace the '8' in '86' with a code letter, except if you only receive Australian Rainbow Magazine. Some of the more usual code letters include

B - Both Australian Rainbow & Australian CoCo
I - Australian CoCo
C - CoCoOz
R - Australian Rainbow on Tape.

Of course we supply many more items than these on subscription, more details can be found on the Goldsoft pages towards the end of this magazine.

We've been having a bit of a tidy up this month - not just with the Data Base but also with some of our equipment.

I have two grey case CoCo's here, neither works well. One merges programs automatically, and probably does other nasty things too; the other is just weird. Both are fitted with Video Amps.

We've decided to make these available to HACKERS only at $160 each. If you are not into hardware and solder and chips and that sort of gear, don't even ask.

We also have a pair of Remex double sided drives for sale. One works, the other I would only use for spares.

Remex drives are a bit strange, but they are good gear if you know your hardware. Again, we would only be interested in selling these to someone who knows what he/she is doing, for $200 the pair.

We have a couple of printers to sell too. However as these will not be released from their current duties for another month or so, I will advise you of these next month.

CoCoConf'86 plans are starting to gel.

It appears that Education will be a major tutorial area on its own this year, as will the MS DOS / T1000 section.

The whole thing seems to be growing very quickly around me - and I confess a little surprise at the keen attitude of many who have already booked to come.

We did have a great time last year, and many of last year's participants have booked just on the strength of that meeting.

I will, in fact, be finalising the list of tutorials and tutorial leaders for publication next month.

The latest tutorial to be offered is one by Bob Delbourgo, whose subject will be "The Use of CoCos in Education". This is a tutorial anyone with even a remote interest in Education will not want to miss!

On the commercial side, a number of suppliers have committed themselves to being at CoCoConf'86, including The Computer Hut from Bowen, Paris Radio, Queensland Colour Software Supplies, and Geoff Fiala (who makes THE printer interface for CoCo!).

So how about it?

The conference is on, it is on at a great time of the year to be somewhere else, and if you come, you're going to see some of the most interesting and inovative gear available. You'll also learn from Australia's foremost CoCo teachers; and you'll get the chance to meet many of the famous contributors to the CoCo scene.

What's more, the venue is one which draws visitors back to the coast year after year.

You can't miss this year - reserve your place - we need to know now if you are going to be here!

Finally, our zanny mate Andrew White, has come up with a few suggestions for those who own a Commodore 64 and don't quite know what to do with it. Andrew has some ideas for interfacing it to the real world.

Look for Andrew's timely hints in next month's Australian CoCo!

Educating with Electronic Communications and Research

By Michael Plog, Ph.D.

In case you have not tried your free hour on the Delphi telecommunications network, I strongly encourage you to take advantage of this offer from THE RAINBOW. Like many of you, I have been playing around with Delphi for a while now, trying to learn the shortest way to get from one point to another.

The folks who created the Delphi system must have been poets, because the name itself implies majesty, mystery and a reference to answers. "Delphi" was a special place to the ancient Greeks. It was the most important Greek temple and home of the oracle of Apollo. Also, the Greeks considered Delphi to be the center of the world. In the temple itself, a stone marked the exact spot of the world's center, called the "navel."

The term "oracle" is actually a Latin word, not Greek. Traditionally, the oracle at Delphi belonged first to Mother Earth. Apollo either stole the oracle or was given it by Mother Earth. The medium of the oracle (the person actually doing the speaking) was always a woman over 50. The procedures to obtain an answer from the oracle were complex and rigid. A "reading" could only be given at certain times of the year. A ritual cake was required, along with a sacrificial animal conforming to rigorous physical standards.

The oracle and her consultants bathed in a special spring, drank from a sacred stream, then entered the temple. The oracle went to a basement cell in the temple, sat on a sacred tripod and chewed leaves of the laurel tree (this was Apollo's special tree). While sitting and chewing on the leaves, the oracle would speak. Her words, however, were not given directly to the person asking the question. They were interpreted and written by the priests, often in highly ambiguous verse.

Delphi has been continuously inhabited since the 14th century B.C. The height of the oracle's prestige and popularity was in the 4th century B.C.

When you stop and think about it, the present electronic Delphi is somewhat similar to the ancient oracle. People approach with a question or a need for information. It is always helpful to have a ritual cake (maybe a sandwich, but I find it easier to use a cookie). The sacrificial animal has been replaced with a plastic credit card, but still requires rigorous standards. (If you don't pay, you don't stay.) Your computer does not have to be in a basement, but you are figuratively apart from the rest of the world. The messages we receive from our electronic oracle are sometimes ambiguous.

I have learned a few things while on Delphi, other than about the system itself. It seems that everyone wants a RAINBOWfest held in a city close to where they live. Most of the Color Computer users responding to a poll have one or more disk drives. Of the 29 respondents to one poll, 75 percent of them use more than one disk operating system for their Color Computer. I have also read some interesting messages about modems and operating systems, and have obtained some public domain software.

I wonder about educational applications of Delphi for schools and students. It seems that two major applications can be expected. First is the communication potential of Delphi. You can send messages to other people (perhaps those having some special expertise) and share ideas with others of a similar interest. The second benefit for education is the research capabilities of Delphi.

The idea of communicating with other people with similar interests is important for the field of education. Several bulletin board services exist for special interest groups in education. These services connect people with similar interests and can serve many functions. For example, local school districts can send applications for special funding to state agencies or foundations by electronic means. One special interest group, educators for the handicapped, has an electronic communications service with one of the features being updates on proposed legislation. Subscribers know what is being discussed in Washington and have an ideal opportunity to contact their representatives and encourage a vote one way or the other.

Electronic research capabilities have revolutionized decision making in business and government, but not yet in schools. Delphi has a connection with a system called DIALOG, a collection of over 200 databases. Some of these are highly specialized, technical databases, such as medical experiments or legal search organizations. A person might want to know how many microcomputers were sold to schools last year; searching the appropriate database provides the answer.

The problems involved in using elec-

Building Language Arts Skills

By Steve Blyn

"Back to basics" skills are the trend in many school systems throughout the country. This renewed emphasis on the traditional language arts and math skills is probably the most popular way computers are currently being used in schools. These types of programs are commonly referred to as C.A.I. — Computer-Assisted Instruction.

Way before the age of computers arrived, there existed many wonderful skill series of language arts workbooks. One such series is published by Barnell Loft, Ltd. These workbooks cover a wide range of skills, including following directions, using the context, locating the answer, getting the facts, getting the main idea and drawing conclusions. The individual booklets are available for the first grade up to high school level.

Almost every school I have visited uses some of these skill booklets. I have used them in my classroom for several years. This month's program is an all-purpose language arts program. It is easily adaptable to any of the ideas presented in such skill books.

By varying the types of questions, this program can be geared to any of the mentioned skill areas. Similarly, the program can be made suitable for almost any level by changing the story content as well as the questions.

One might ask, "Why is it necessary to replicate on the computer what already exists in booklet form?" Tsk! Tsk! There are several very good reasons.

Computers may be presenting similar material, but through a very different medium. Children are naturally attracted to computers and sometimes learn better through them than by traditional means. Computers have the ability to give immediate feedback to the student — you know at every step of the way whether you are succeeding or not.

Computers are also impersonal. By that, I don't mean they don't have any personality. Morris, my original CoCo, is very dear to me and has practically become a family member. (We do celebrate his birthday.) By impersonal, I mean they are tireless reformers of mistakes. They do not complain to the user, nor do they make any judgments against the user. It is almost impossible for a person to remain impartial to successes and failures as computers do.

The program presents part of the "Jack and the Beanstalk" story. This is located on lines 90 and 100. Our story is merely used as an example. The choice of story and the grade level for which it is written should be yours. The story of Jack was taken from a third-grade reader.

A series of questions about the story is placed in the DATA lines 260-390. Our questions use a scattered approach: They purposely do not cover any one specific skill area. They are meant to illustrate the variety of questions you may use. They even go beyond the Barnell Loft areas mentioned earlier. Your questions can be as diverse as your imagination or just remain with one skill area.

Fourteen questions were entered as illustrations. We chose the number 14 to show you that we are not limited only to 10 questions. Often, computer newcomers think there is something magical about the number 10 or its multiples. Not so! A short program such as this one can have literally hundreds of questions entered. You must, however,

Steve Blyn teaches both exceptional and gifted children, holds two master's degrees and has won awards for the design of programs to aid the handicapped. He owns Computer Island and lives in Staten Island, New York.

tronic research are generally cost and training. The price for using some of the databases can be very high. Some of the specialized databases can cost thousands of dollars per year, plus online connection time. Most schools are not willing to incur such expenses for student projects. Also, each database requires special procedures for searching. Those procedures can become complex for the untrained person.

I believe these problems will be eliminated in the near future. The procedures for specialized searching are a matter of software. Computer professionals are currently working on new languages to help the human and machine understand each other better. The cost factor may be a little more difficult to solve. It is expensive to maintain even a simple database — a lot of time is required (which must be paid for by someone). A lot of expensive equipment is also necessary to provide a database. The more people who use such systems, however, the less each will have to pay.

Will we ever reach a time when electronic communications and research are common practices for elementary and secondary students? Will we have, for example, a sixth-grade student in Florida writing a report about earthquakes, and including as part of that report, an interview with a California student who recently experienced an earthquake?

The future is unknown; our current Delphi oracles only share present information, not future happenings. However, if you or your school is using electronic communications or research, I would like to know about it. Please share your experiences and efforts. My Delphi username is MPLOG — why not drop me a line? □

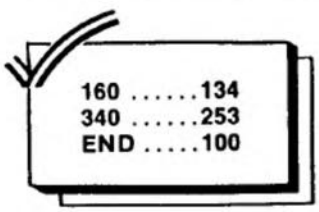
tell the computer how many questions to read. Our number of questions is indicated in the dimension statement on Line 30 and also on lines 40 and 50. Be sure to include the number of questions you use on those lines.

The program is set to ask 10 of the story questions each round. Common sense dictates that rounds come in sets of 10 questions. Other numbers that divide evenly into 100 are also good choices. The questions appear one at a time and do not repeat in a given round.

Correct answers receive a happy tune and a message that says "correct" on Line 210. The incorrect answers, however, are really the important ones. Each time a question is answered incorrectly, its question and correct answer are stored in lines 400-420. This makes possible a review of these questions and answers. This is similar to a study-review sheet.

The program handles the review on lines 430-480. If you have a printer, it is a good idea to print out the review for the student's future reference. To get a printout, change the PRINT statements on lines 430, 440 and 460 to PRINT#-2, statements. You may add these to the existing lines to get the output on both the screen and the printer.

It is hoped that you use this program as a model for incorporating your own versions. You may create fun programs to reinforce children's favorite stories or programs to strictly strengthen specific language arts skills. A combination of the two might be the best route to go. Remember to save each of your versions on tape or disk before proceeding to your next creation. □



The listing: FINDWORD

```
1Ø REM"FIND THE WORD"
2Ø REM"STEVE BLYN,COMPUTER ISLAN
D,NY,1986
3Ø DIM A$(14),B$(14),X$(1Ø),Y$(1
Ø)
4Ø FOR T=1 TO 14:READ A$(T),B$(T
):NEXT T
5Ø R=RND(14)
6Ø XY=RND(-TIMER)
7Ø CLS
8Ø PRINT@32,STRING$(32,2Ø7);
9Ø PRINT"   JACK CLIMBED THE BEA
NSTALK A SECOND TIME. HE WAS AGA
IN HELPEDBY THE GIANT'S WIFE. TH
IS TIME  HE TOOK THE HEN THAT LA
ID THE   GOLDEN EGGS. HE ESCAPED
 QUICKLY.";
1ØØ PRINT"  JACK PICKED UP THE M
AGIC HARP ON HIS THIRD TRIP. BUT
 THE HARP CALLED OUT AND WOKE TH
E GIANT.  THE GIANT BEGAN TO CHA
SE JACK."
11Ø N=N+1
12Ø PRINT@Ø,"N=";N;"       ** JA
CK  **   R=";CR;
13Ø IF N>1Ø THEN GOTO 43Ø
14Ø PRINT@352,STRING$(32,(RND(12
8)+127));
15Ø PRINT@416,STRING$(95," ");
16Ø PRINT@384,"TRY TO FIND THE W
ORD THAT..."
17Ø IF R>13 THEN R=Ø
18Ø R=R+1
19Ø PRINTA$(R)
2ØØ INPUT C$
21Ø IF C$=B$(R) THEN PLAY"L1ØØCE
GCEGCC":PRINT"CORRECT. PRESS ENT
ER TO GO ON";:CR=CR+1
22Ø IF C$<>B$(R) THEN PLAY "L4CC
":PRINTB$(R)" IS THE ANSWER.";:G
OSUB 4ØØ
23Ø EN$=INKEY$
24Ø IF EN$=CHR$(13) THEN 11Ø
25Ø GOTO 23Ø
26Ø DATA IS A COMPOUND WORD,BEAN
STALK
27Ø DATA IS A COLOR,GOLDEN
28Ø DATA HAS AN APOSTROPHE,GIANT
'S
29Ø DATA IS THE OPPOSITE OF HUSB
AND,WIFE
3ØØ DATA IS GOOD TO EAT FOR BREA
KFAST,EGGS
31Ø DATA MEANS MORE THAN TWO,TIM
ES,THIRD
32Ø DATA MEANS THE OPPOSITE OF S
MALL,GIANT
33Ø DATA MEANS THE SAME AS FAST,
QUICKLY
34Ø DATA MEANS THE SAME AS A VOY
AGE,TRIP
35Ø DATA IS THE NAME OF AN ANIMA
L,HEN
36Ø DATA IS A MUSICAL INSTRUMENT
,HARP
37Ø DATA IS THE OPPOSITE OF IN,O
UT
38Ø DATA THAT APPEARS MOST OFTEN
 ABOVE,THE
39Ø DATA THAT IS USED 4 TIMES,JA
CK
4ØØ X$(J)=A$(R):Y$(J)=B$(R)
41Ø J=J+1
42Ø RETURN
43Ø CLS:PLAY"CDEFG":PRINT"HERE I
S YOUR REVIEW"
44Ø IF CR=1Ø THEN PRINT:PRINT"VE
RY GOOD ... 1ØØ%":GOTO 49Ø
45Ø FOR K=Ø TO J-1
46Ø PRINT K+1;".";Y$(K);" IS THE
 WORD THAT":PRINTX$(K):PRINT
47Ø EN$=INKEY$
48Ø IF EN$=CHR$(13) THEN NEXT K
ELSE 47Ø
49Ø END
```

A nonviolent game for children . . .

Set your Sails, Keep a Weather Eye out for Storms and Beware of the Jolly Roger!

By David Compton

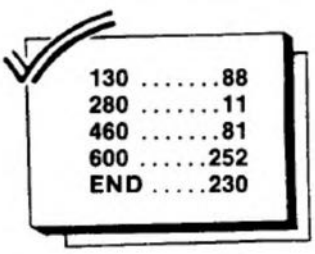
Several times I have seen in the pages of RAINBOW a plea for non violent games for children.

World Trader is written for the young child, perhaps a second or third grader. Even younger children can use it with parental help.

World Trader is a text Adventure, but the reading is kept to a minimum. It essentially seeks to teach children the names of some countries and the products for which they are best known. The player can't "lose" the game; he is just sent back to the beginning to start over. At the same time, there is an element of nonfatal excitement - pirates or storms may strike at any moment, costing you money, or you may arrive in a country to sell your goods, only to find that your customers aren't interested!

All of the instructions are given at the beginning of the game, but here's a summary: The idea is to earn $25,000 or more by buying merchandise in one country and selling it at a profit in another. Note that you may have only one of each item in your hold at any one time. Only four commands are needed by the player, GO, which presents a list of 10 countries you may travel to; BUY, which displays what the inhabitants have for sale (and reduces your cash on hand if you decide to buy); SELL, which disposes of your cargo and updates your cash; and INV, which informs you what's in your cargo hold.

The player must also bear in mind that each time he uses the commmand GO, his funds are reduced by $100 to pay the cost of shipping and salaries for the crew. The first few trips should be made carefully, or the captain (you) will quickly run out of money.



The listing: TRADER

```
1Ø 'TRADER
12 ' BY DAVID COMPTON, 252 N. MA
IN ST., SUFFIELD, CT Ø6Ø78
13 'COPYRIGHT 1985
2Ø CLS
3Ø AA$="THE GAME OF":BB$="WORLD
TRADER":PRINT@2Ø8-(LEN(AA$)*.5),
AA$:PRINT@272-(LEN(BB$)*.5),BB$
4Ø FOR DL=1TO2ØØØ:NEXT:CLS
5Ø PRINT"  IN THIS GAME, YOU STA
RT WITH  A SHIP AND $1ØØØ. THE O
BJECT IS TO TRAVEL AROUND THE WO
RLD,     BUYING AND SELLING, UNT
IL YOU   EITHER RUN OUT OF MONEY
 OR EARN ENOUGH TO RETIRE."
6Ø PRINT"  EACH VOYAGE WILL COST
 YOU     $1ØØ. IN ADDITION, YOU'
LL HAVE  TO BE CAREFUL OF PIRATE
S AND    STORMS."
7Ø PRINT"  YOUR CREW UNDERSTANDS
 THE COM-MANDS 'GO', 'BUY', 'SEL
L' AND   'INV' (INVENTORY)."
8Ø PRINT@448,"PRESS ANY KEY TO B
EGIN"
9Ø IFINKEY$=""THEN9Ø
1ØØ CLS
11Ø WE=1ØØØ:W$="$$##,###"
12Ø DIMCN$(1Ø,8)
13Ø DATA BRAZIL,BRAZILIANS,COFFE
E,5ØØ,Ø,RELICS,2ØØØ,Ø
14Ø DATA HOLLAND,DUTCH,CHOCOLATE
,1ØØ,Ø,DIAMONDS,1ØØØØ,Ø
15Ø DATA FRANCE,FRENCH,WINE,1ØØØ
,Ø,CHEESE,25Ø,Ø
16Ø DATA GERMANY,GERMANS,BEER,3Ø
Ø,Ø,CLOTHING,1ØØØ,Ø
17Ø DATA NORWAY,NORWEGIANS,FISH,
2ØØ,Ø,CRYSTAL,5ØØØ,Ø
18Ø DATA CHINA,CHINESE,TEA,1ØØ,Ø
,SPICES,2ØØØ,Ø
19Ø DATA ARGENTINA,ARGENTINES,BE
EF,1ØØØ,Ø,HORSES,3ØØØ,Ø
2ØØ DATA ITALY,ITALIANS,PASTA,35
Ø,Ø,STATUES,6ØØØ,Ø
21Ø DATA DENMARK,DANES,FURNITURE
,5ØØØ,Ø,CLOTH,1ØØØ,Ø
22Ø DATA INDIA,INDIANS,IVORY,85Ø
Ø,Ø,FABRICS,7ØØ,Ø
23Ø FORX=1TO1Ø:FORY=1TO8:READCN$
(X,Y):NEXTY,X
24Ø PRINT"YOU BEGIN YOUR VOYAGE
IN LONDON.YOU HAVE";:PRINTUSINGW
$;WE
25Ø PRINTSTRING$(32,"$");
26Ø IFWE>24999THENPRINT"YOU HAVE
";:PRINTUSINGW$;WE:PRINT"A SUCCE
SSFUL VOYAGE! YOU RETURN TO ENGL
AND IN TRIUMPH!":END
27Ø PRINT"YOUR ORDERS,SIR?"
28Ø INPUTO$
29Ø IFO$="SELL"THENGOTO42Ø
3ØØ IFO$="BUY"THENGOTO56Ø
31Ø IFO$="GO"THENFORX=1TO1Ø:PRIN
TX;CN$(X,1):NEXT:GOTO34Ø
```

continued on Page 10

Invasion of the Flying Saucer People

By Allen B. Carlisle

If you like fast action games that challenge your reflexes and offer different skill levels, you will like *Saucer*. The scenario goes something like this: You are on a desolate planet, and while safe from attack from the dreaded Saucer people from within your base camp (lower right-hand corner of the graphics screen), you must venture out to get the supply boxes, which are present at the lower left-hand corner of the graphics screen. The moment you venture out, or your robot ventures out (for those who abhor violence), a saucer enters the scene and randomly flies around shooting its laser beam at you. If you are hit, another man, or robot, exits the base and heads for the needed supplies. Each player gets four men per round.

You are not left without defense, however, as you can shoot back by skillful use of the right joystick. Of course, the firebutton activates your laser gun, but aiming it takes a few attempts to master. Each time you succeed in knocking out a saucer, 100 points are scored. Aiming your gun involves watching a blinking cursor at the uppermost horizontal part of the screen or the far left vertical position of the graphics screen. Your ray fires at that cursor position as you press the button. Movement to the right of the joystick causes the cursor to appear at the top, while movement to the left moves the cursor to the left vertical part of the screen. Up on the joystick causes the cursor to move up the screen if it is at the left vertical position, and to the left if it is at the upper horizontal position. Of course, down performs the opposite movements.

When I decided to write *Saucer*, I knew the main mathematical task would be to obtain the formula that calculates the coordinates of the point on a line (laser ray) that is on the segment perpendicular to some other point off the line (center of the saucer). After having looked in all my analytical geometry texts, I found nothing that would give me what I needed, so I took some time to derive the formulae I needed. Line 1090 of this program is what gives the coordinates of this point (LX,LY), where (A,B) and (C,D) are two points of the laser beam shot at the saucer and (X,Y) is the coordinate of the center of the saucer. The actual distance is calculated in Line 1100.

Variable QED is the test variable for this distance, which is larger for the lower skill level so that at skill level one, the beam does not necessarily have to touch the saucer in order to score a hit, but must be very close.

Good luck knocking those nasty Saucer people from the sky! □

250 32	970 154
380 182	1110 112
610 74	END 254
770 138	

The listing: SAUCER

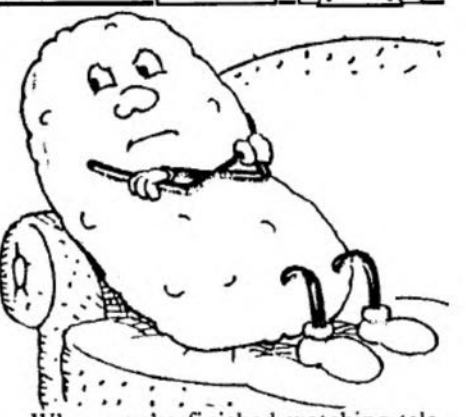
```
1Ø REM SAUCER ****************
2Ø REM (C) ALLEN B. CARLISLE 198
5****
3Ø REM ALL RIGHTS RESERVED******
*
4Ø '****************************
5Ø '****************************
6Ø REM INITIALIZE GAME
7Ø CLS:PCLEAR4:POKE65495,Ø
8Ø DIMA$(4):DIMA(4):INPUT"NUMBER
 OF PLAYERS";ZZ
9Ø FOR PL=1 TO ZZ:CLS:INPUT"ENTE
R NAMES";A$(PL):NEXT:CLS:PL=1
1ØØ INPUT"SKILL LEVEL(1-5)";L
11Ø IF L<1 OR L>5 THEN 1ØØ ELSE
L=ABS(INT(L))
12Ø IF L=1 THEN 13Ø ELSE 14Ø
13Ø DIS=Ø:QED=17:GOTO25Ø
14Ø IF L=2 THEN 15Ø ELSE 16Ø
15Ø DIS=1:QED=15:GOTO25Ø
16Ø IF L=3 THEN 17Ø ELSE 18Ø
17Ø DIS=3:QED=13:GOTO25Ø
18Ø IF L=4 THEN 19Ø ELSE 2ØØ
19Ø DIS=3:QED=1Ø:GOTO25Ø
2ØØ DIS=3:QED=8:GOTO25Ø
21Ø IF PL>ZZ THEN PL=1
22Ø A(PL)=A(PL)+SCR
23Ø IF MEN=Ø THEN PL=PL+1
24Ø IF MEN=Ø THEN GOTO117Ø ELSE
RETURN
25Ø REM DRAW SAUCER & GET IT
26Ø PMODE4,1:U=214:SCREEN1,1:PCL
SØ:MEN=8
27Ø H1=2Ø:H2=44:V1=7:V2=7:V3=7:V
4=7:FL=-1:BOX=Ø:M=92
28Ø FOR I=1 TO 4: LINE(H1,V3)-(H
2,V4),PSET:LINE(H1,V1)-(H2,V2),P
SET
29Ø H1=H1+1:H2=H2-1:V1=V1-1:V2=V
2-1:V3=V3+1:V4=V4+1:PSET(31,2):P
SET(32,2):NEXT
3ØØ DIMS(12,26):GET(19,1)-(45,13
),S,G:LINE(19,1)-(45,13),PRESET,
BF
31Ø REM DRAW &GET ROBOT
32Ø DRAW"BM217,18ØR3D1L3R2D6L1U3
L1;B;U1L2U1;B;D9R1U1;B;R1U2;B;R3
D1;B;R1;B;D1D1L1"
33Ø DIMR(11,8):GET(214,18Ø)-(222
,191),R,G
34Ø REM DRAW BASE
35Ø LINE(24Ø,164)-(255,191),PSET
,BF:LINE(228,176)-(244,191),PSET
,BF
```

```
36Ø FOR I=168 TO 184 STEP 8:LINE
(248,I)-(251,I+3),PRESET,BF:NEXT
37Ø LINE(232,184)-(235,187),PRES
ET,BF:LINE(24Ø,184)-(243,187),PR
ESET,BF
38Ø REM DRAW SUPPLY BOXES
39Ø FOR I=Ø TO 88 STEP 8
4ØØ LINE(I,18Ø)-(I+3,191),PSET,B
:NEXT:SCREEN1,1
41Ø REM ENTER SAUCER
42Ø FOR I=Ø TO 128 STEP 8
43Ø W=2Ø
44Ø PUT(I,2Ø)-(I+26,32),S,PSET
45Ø LINE(I,2Ø)-(I+26,32),PRESET,
BF
46Ø ABR=RND(9):IF ABR=2 THEN 48Ø
ELSENEXT
47Ø GOTO 49Ø
48Ø GOTO 51Ø
49Ø I=128:PUT(I,W)-(I+26,W+12),S
,PSET:GOTO51Ø
5ØØ REM MOVE SAUCER
51Ø Q1=RND(Ø):IF Q1<.5 THEN A=-1
 ELSE A=1
52Ø Q2=RND(Ø):IF Q2<.5 THEN B=-1
 ELSE B=1:H=RND(6):V=RND(6)
53Ø H=H*A:V=V*B:L=RND(2Ø)
54Ø FOR E=1 TO L
55Ø IF I>226 THEN 63Ø
56Ø IF W>14Ø THEN 63Ø
57Ø IF I<Ø THEN 63Ø
58Ø IF W<Ø THEN 63Ø
59Ø PUT(I,W)-(I+26,W+12),S,PSET:
X=I:Y=W
6ØØ GOSUB 64Ø
61Ø LINE(I,W)-(I+26,W+12),PRESET
,BF:I=I+H:W=W+V
62Ø NEXT:GOTO63Ø
63Ø PUT(X,Y)-(X+26,Y+12),S,PSET:
GOSUB78Ø:FOR DLY=1 TO 2:SOUND216
,1:SOUND226,1:GOSUB 82Ø:NEXT:I=X
:W=Y:GOSUB86Ø:GOTO51Ø
64Ø REM MOVE ROBOT
65Ø GOSUB86Ø
66Ø IF FL>Ø THEN 7ØØ
67Ø LINE(U,18Ø)-(U+8,191),PRESET
,BF:U=U-1:PUT(U,18Ø)-(U+8,191),R
,PSET
68Ø IF M=U THEN 69Ø ELSE RETURN
69Ø FL=FL*-1:BOX=BOX+1:IF BOX=1
THEN 75Ø ELSE U=U-8
7ØØ IF U<Ø THEN U=Ø
71Ø LINE(U,18Ø)-(U+13,191),PRESE
T,BF:PUT(U,18Ø)-(U+13,191),B,PSE
T:U=U+1
72Ø IF U=215 THEN 73Ø ELSE RETUR
N
73Ø LINE(U,18Ø)-(U+13,191),PRESE
T,BF:FL=FL*-1:U=214:PUT(U,18Ø)-(
U+13,191),R,PSET
74Ø M=M-8:SCR=1ØØØ:GOSUB21Ø:IF M
<4 THEN 125Ø ELSE RETURN
75Ø REM GET ROBOT & BOX
76Ø DIMB(11,13):GET(87,18Ø)-(1ØØ
,191),B,G:U=87:GOTO 7ØØ
77Ø REM SHOOT LASER BEAM
78Ø XX=X+13:L=U+4:R1=RND(5):R2=R
ND(5):R3=RND(5):R=R1+R2+R3
79Ø IF L-9+R<Ø THEN R=9
8ØØ LINE(XX,Y)-(L-8+R,191),PSET:
FOR DLY=1 TO 4Ø:NEXT
81Ø LINE(XX,Y)-(L-8+R,191),PRESE
T:RETURN
82Ø IF ABS(R-8)<=DIS THEN 83Ø EL
SE RETURN
83Ø LINE(U,18Ø)-(U+8,191),PRESET
,BF:MEN=MEN-1
84Ø SCR=Ø
85Ø U=214:FL=-1:GOSUB21Ø:RETURN
86Ø REM ROBOT SHOOTS
87Ø K=PEEK(6528Ø):IF K=126 OR K=
254 THEN 1Ø2Ø
88Ø JØ=JOYSTK(Ø):J1=JOYSTK(1)
89Ø IF J1>54 THEN UN=UN+8
9ØØ IF J1>45 AND J1<=54 THEN UN=
UN+5
91Ø IF J1>36 AND J1<=45 THEN UN=
UN+2
92Ø IF J1>27 AND J1<=36 THEN UN=
UN
93Ø IF J1>18 AND J1<=27 THEN UN=
UN-2
94Ø IF J1>9 AND J1<=18 THEN UN=U
N-5
95Ø IF J1>=Ø AND J1<=9 THEN UN=U
N-8
96Ø IF UN<Ø THEN UN=Ø
97Ø IF JØ>32 THEN 1ØØØ
98Ø IF UN>16Ø THEN UN=16Ø
99Ø LINE(Ø,UN)-(4,UN+4),PSET,BF
:LINE(Ø,UN)-(4,UN+4),PRESET,BF:R
ETURN
1ØØØ IF UN>254 THEN UN=254
1Ø1Ø LINE(UN,Ø)-(UN+4,4),PSET,BF
:LINE(UN,Ø)-(UN+4,4),PRESET,BF:R
ETURN
1Ø2Ø IF JØ>32 THEN 1Ø5Ø
1Ø3Ø A=Ø:B=UN:C=U:D=18Ø
1Ø4Ø LINE(Ø,UN)-(U,18Ø),PSET:FOR
 DLY=1TO2:SOUND216,1:SOUND226,1:
NEXT:GOSUB1Ø7Ø:LINE(Ø,UN)-(U,18Ø
),PRESET:RETURN
1Ø5Ø LINE(UN,Ø)-(U,18Ø),PSET:A=U
N:B=Ø:C=U:D=18Ø
1Ø6Ø FORDLY=1TO2:SOUND216,1:SOUN
D226,1:NEXT:GOSUB1Ø7Ø:LINE(UN,Ø)
-(U,18Ø),PRESET:RETURN
1Ø7Ø REM CHECK IF SAUCER HIT
1Ø8Ø X=X+13:Y=Y+6
1Ø9Ø LX=(X*(C-A)^2+A*(D-B)^2+(D-
B)*(C-A)*(Y-B))/((D-B)^2+(C-A)^2
):LY=(B*(C-A)^2+Y*(B-D)^2+(B-D)*
(C-A)*(A-X))/((D-B)^2+(C-A)^2)
11ØØ IF SQR((LY-Y)^2+(LX-X)^2)<=
QED THEN 112Ø
111Ø X=X-13:Y=Y-6:FORDLY=1TOX:AB
R=RND(Ø):NEXT:RETURN
112Ø REM SAUCER HIT
113Ø LINE(I,W)-(I+26,W+12),PRESE
T,BF:FORDLY=1TO2Ø:SOUND2ØØ,1:RH=
RND(26):RV=RND(12):PSET(I+RH,W+R
V):NEXT:X=X-13:Y=Y-6
114Ø LINE(I,W)-(I+26,W+12),PRESE
T,BF
115Ø IF JØ>32 THEN LINE(UN,Ø)-(U
,18Ø),PRESETELSELINE(Ø,UN)-(U,18
Ø),PRESET
116Ø SCR=1ØØ:GOSUB21Ø:GOTO41Ø
117Ø CLS:FORK=1 TO ZZ:PRINT@(64+
K*64),A$(K);"'S SCORE= ";A(K):NE
XT
118Ø PRINT@384,"TO END GAME PRES
S 'E'":PRINT@352,"NEXT ROUND PRE
SS 'N'"
119Ø B$=INKEY$:IF B$="E"THEN126Ø
ELSE IFB$="N"THEN122ØELSEIFFLAG=
1THENRETURNELSE119Ø
12ØØ IF FLAG=1THENRETURN
121Ø CLS
122Ø IF PL>ZZ THEN PL=1
123Ø PRINT@23Ø,"R E A D Y ";A$(P
L);:FORDLY=1TO 5ØØØ:NEXT
124Ø LINE(X,Y)-(X+26,Y+12),PRESE
T,BF:SCREEN1,1:MEN=8:GOTO41Ø
125Ø CLS:PRINT@1Ø3,"G A M E  O V
 E R !!!!":NEXT:FOR DLY=1 TO 15Ø
ØØ:NEXT:FLAG=1:GOSUB117Ø
126Ø POKE113,Ø:EXEC4Ø999
```

continued from Page 8

```
32Ø IFO$="INV"THENGOTO72Ø
33Ø PRINT"I DON'T UNDERSTAND, CA
PTAIN":GOTO25Ø
34Ø INPUTDE:IFDE<1ORDE>1ØTHEN25Ø
35Ø GOSUB67Ø
36Ø CLS:L$=CN$(DE,1):PRINT"YOU A
RE IN ";CN$(DE,1)
37Ø WE=WE-1ØØ
38Ø PRINT"YOU HAVE";:PRINTUSINGW
$;WE
39Ø IFWE<ØTHENPRINT"WE'RE OUT OF
 MONEY, CAPTAIN.    WE'LL HAVE T
O RETURN TO ENGLAND TO GET A LOA
N.":FORX=1TO2ØØØ:NEXT:END
4ØØ GOTO25Ø
41Ø '********SELL*******
42Ø FORD=1TO1Ø:IFCN$(D,5)="1"THE
N45ØELSENEXTD
43Ø FORD=1TO1Ø:IFCN$(D,8)="1"THE
N45ØELSENEXTD
44Ø PRINT"YOU HAVE NOTHING TO SE
LL.":GOTO27Ø
45Ø IFRND(1Ø)=6THENPRINT"THE ";C
N$(DE,2);" AREN'T":PRINT"INTERES
TED IN BUYING.":O$="GO":GOTO31Ø
46Ø PRINT"THE ";CN$(DE,2);" WILL
 BUY ";
47Ø FORX=1TO1Ø:IFCN$(X,5)="1"THE
NPRINTCN$(X,3):PRINT"SOLD FOR";:
PRINTUSINGW$;VAL(CN$(X,4))+(.3*V
AL(CN$(X,4))):WE=WE+VAL(CN$(X,4)
)+(.3*VAL(CN$(X,4)))
48Ø CN$(X,5)="Ø"
49Ø NEXTX
5ØØ FORX=1TO1Ø:IFCN$(X,8)="1"THE
NPRINTCN$(X,6):PRINT"SOLD FOR";:
PRINTUSINGW$;VAL(CN$(X,7))+(.3*V
AL(CN$(X,7))):WE=WE+VAL(CN$(X,7)
)+(.3*VAL(CN$(X,7)))
51Ø CN$(X,8)="Ø"
52Ø NEXTX
53Ø PRINT"YOU HAVE";:PRINTUSINGW
$;WE
54Ø GOTO25Ø
55Ø '*******BUY******
56Ø PRINT"THE ";CN$(DE,2);" WISH
 TO SELL:":PRINTCN$(DE,3),:PRINT
USINGW$;VAL(CN$(DE,4)):PRINTCN$(
DE,6),:PRINTUSINGW$;VAL(CN$(DE,7
))
57Ø PRINT"WHICH WILL YOU BUY?"
58Ø INPUTBU$
59Ø IFBU$<>CN$(DE,3)ANDBU$<>CN$(
DE,6)THENPRINT"THEY DON'T HAVE A
NY FOR SALE"
6ØØ IFBU$=CN$(DE,3)THENIFWE<VAL(
CN$(DE,4))THENPRINT"NOT ENOUGH M
ONEY":GOTO25Ø
61Ø IFBU$=CN$(DE,6)THENIFWE<VAL(
CN$(DE,7))THENPRINT"NOT ENOUGH M
ONEY":GOTO25Ø
62Ø IFBU$=CN$(DE,3)THENCN$(DE,5)
="1":WE=WE-VAL(CN$(DE,4))
63Ø IFBU$=CN$(DE,6)THENCN$(DE,8)
="1":WE=WE-VAL(CN$(DE,7))
64Ø PRINT"YOU HAVE";:PRINTUSINGW
$;WE
65Ø GOTO25Ø
66Ø END
67Ø CA=RND(-TIMER):CA=RND(1Ø)
68Ø IFCA=1THENLO=RND(1Ø)*1Ø:PRIN
T"YOU'RE ATTACKED BY PIRATES DUR
- ING THE VOYAGE. THEY STEAL SOM
E OF YOUR MONEY.":WE=WE-LO:FORDL
=1TO4ØØØ:NEXTDL:RETURN
69Ø IFCA=2THENLO=RND(1Ø)*1Ø:PRIN
T"A BAD STORM! SEA WATER DAMAGES
  YOUR CARGO. YOU'VE LOST $";LO;
"IN MERCHANDISE.":WE=WE-LO:FORDL
=1TO4ØØØ:NEXTDL:RETURN
7ØØ RETURN
71Ø '**********INV******
72Ø FORD=1TO1Ø:IFCN$(D,5)="1"THE
NPRINT CN$(D,3)
73Ø NEXTD
74Ø FORD=1TO1Ø:IFCN$(D,8)="1"THE
N PRINTCN$(D,6)
75Ø NEXT D
76Ø GOTO25Ø
```

A PEACE TREATY FOR COMPUTER HACKERS AND COUCH POTATOES



by Bill Bernico

What's the matter? Is your family on your case because your computer's always hooked up to the television and they can't watch their favourite shows? Now you can "solve" that problem with TV Shows (provided they haven't already disconnected you from the tube.)

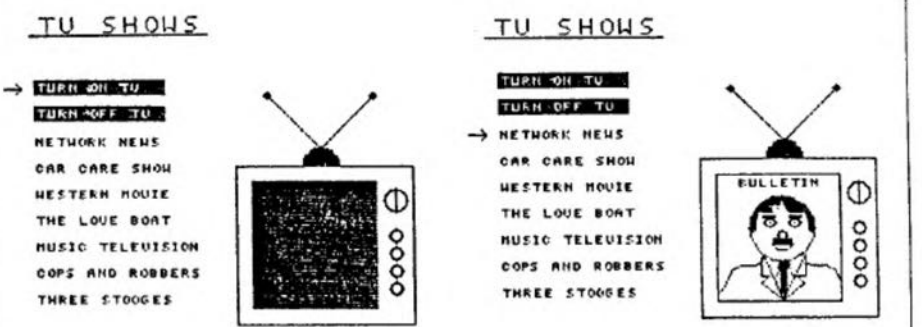
Actually "solve" is not quite accurate. Though this program doesn't truly solve that very real dilemma, it does offer a tongue-in-cheek response to it by using the CoCo's sound and graphics capabilities.

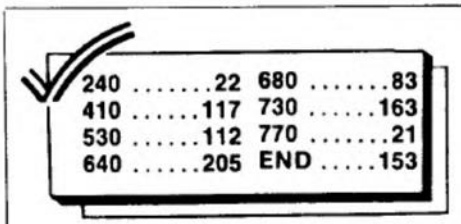
Upon running the program you'll see a familiar sight - a television set. Along the left side of the screen you are presented with a list of nine choices. The last seven are things to watch on the TV and the first two are options that allow you to either turn the television on or off. Your choice is selected by using the up and down arrows. When you have chosen your option, press ENTER.

What! You selected a television show but nothing happened? Did you remember to turn the set on? Just as in real life, you can't watch anything until you first turn on the TV. Move the arrow to the top and press ENTER. Now the set's on and you can go ahead and select a program.

When you're finished watching television, don't forget to turn it off again. You can do this by moving the arrow up to the second option and pressing ENTER. Once the set is turned off, trying to select anything else is useless; the set has to be on first.

Sample Printouts





The listing: TV SHOWS

```
10 'TELEVISION SHOWS
20 'BY BILL BERNICO
30 '708 MICHIGAN AVE.
40 'SHEBOYGAN, WI 53081
50 '(414) 459-7350
60 '
70 CLEAR 500
80 SP$="BR6
90 PD$="BR3R
100 WA$="F2R2E2UDF2R2E2UDF2R2E2U
DF2R2E2UDF2R2E2UDF2R2E2UDF2R2E2U
DF2R2E2UDF2R2E2UDF2R2E2UDF2R2E2U
DF
110 N1$="BR3NU4
120 N3$="BR3R3U2NL2U2NL3BD4
130 A$="BR3U3ERFDNL3D2
140 B$="BR3U4R2FGNL2FGL2BR3
150 C$="BR4REGLHU2ERFBD3
160 D$="BR3U4R2FD2GL2BR3
170 E$="BR3NR3U2NR2U2R3BD4
180 F$="BR3U2NR2U2R3BD4
190 G$="BR3BU4BR3L2GD2FREULBR2BD
2
200 H$="BR3U4D2R3U2D4
210 I$="BR3R2LU4NLRBD4
220 J$="BR3BUFREU3LR2BD4
230 K$="BR3U2RNF2NE2LU2BR3BD4
240 L$="BR3NR3U4BR3BD4
250 M$="BR3U4FRED4
260 N$="BR3U4F3DNU4
270 O$="BR3BUU2ERFD2GLNHBR
280 P$="BR3U4R2FGL2BR2BD2
290 Q$="BR3BUU2ERFD2GNUNRLHBR3BD
300 R$="BR3U4R2FGL2RF2
310 S$="BR3R2EHLHER2BD4
320 T$="BR3BU4R4L2D4BR
330 U$="BR3NU4R3NU4
340 V$="BR3BU4D3FRENU3BD
350 W$="BR3NU4ERFNU4
360 X$="BR3UE2UDGHUDF2D
370 Y$="BR3BU4DFEUDGD2BR
380 PMODE4,1:PCLS1:SCREEN1,1:COL
OR0,1:DRAW"S4BM130,170C0U90R100D
90L100BE9U72R70D72L70":PAINT(142
,91),0,0:CIRCLE(220,100),7:FOR X
=120 TO 150 STEP10:CIRCLE(220,X)
,4:NEXT:CIRCLE(178,80),11,0,1,.5
,1:DRAW"BM178,70NH30E30BM220,100
U7D14":CIRCLE(148,40),2
390 CIRCLE(208,40),2:PAINT(178,7
7),0,0:DRAW"BM10,9S8"+T$+V$+SP$+
S$+H$+O$+W$+S$:DRAW"BM10,11R50
400 DRAW"S4":LINE(17,39)-(90,31)
,PSET,B:PAINT(18,36),0,0:DRAW"BM
16,37C1"+T$+U$+R$+N$+SP$+O$+N$+S
P$+T$+V$:DRAW"C0":LINE(17,54)-(9
0,46),PSET,B:PAINT(18,51),0,0:DR
AW"BM16,52C1"+T$+U$+R$+N$+SP$+O$
+F$+F$+SP$+T$+V$
410 DRAW"S4BM15,68C0"+N$+E$+T$+W
$+O$+R$+K$+SP$+N$+E$+W$+S$:DRAW"
BM15,83"+C$+A$+R$+SP$+C$+A$+R$+E
$+SP$+S$+H$+O$+W$:DRAW"BM15,98"+
W$+E$+S$+T$+E$+R$+N$+SP$+M$+O$+V
$+I$+E$:DRAW"BM15,113"+T$+H$+E$+
SP$+L$+O$+V$+E$+SP$+B$+O$+A$+T$
420 DRAW"BM15,128"+M$+U$+S$+I$+C
$+SP$+T$+E$+L$+E$+V$+I$+S$+I$+O$
+N$:DRAW"BM15,143"+C$+O$+P$+S$+S
P$+A$+N$+D$+SP$+R$+O$+B$+B$+E$+R
$+S$:DRAW"BM15,158"+T$+H$+R$+E$+
E$+SP$+S$+T$+O$+O$+G$+E$+S$
430 H=0:V=96:ZZ=0
440 BB$="R10NH3G3
450 DRAW"C0BM=H;,=V;"+BB$
460 II$=INKEY$
470 IF II$=CHR$(94)THEN DRAW"C1B
M=H;,=V;"+BB$:SOUND 210,1:V=V-15
480 IF II$=CHR$(10)THEN DRAW"C1B
```

```
M=H;,=V;"+BB$:SOUND 21Ø,1:V=V+15
49Ø IF II$=CHR$(13)AND V=36 AND
ZZ=Ø THEN 82Ø
5ØØ IF II$=CHR$(13)AND V=51 AND
ZZ=1 THEN 81Ø
51Ø IF II$=CHR$(13)AND V=66 AND
ZZ=1 THEN 61Ø
52Ø IF II$=CHR$(13)AND V=81 AND
ZZ=1 THEN 66Ø
53Ø IF II$=CHR$(13)AND V=96 AND
ZZ=1 THEN 68Ø
54Ø IF II$=CHR$(13)AND V=111 AND
 ZZ=1 THEN 7ØØ
55Ø IF II$=CHR$(13)AND V=126 AND
 ZZ=1 THEN 71Ø
56Ø IF II$=CHR$(13)AND V=141 AND
 ZZ=1 THEN 73Ø
57Ø IF II$=CHR$(13)AND V=156 AND
 ZZ=1 THEN 75Ø
58Ø IF V>156 THEN V=156
59Ø IF V<36 THEN V=36
6ØØ GOTO 45Ø
61Ø GOSUB 83Ø:DRAW"CØ":CIRCLE(17
6,12Ø),19:CIRCLE(176,124),3
62Ø CIRCLE(169,117),4,Ø,.9:CIRCL
E(183,117),4,Ø,.9:DRAW"BM171,126
R1ØDL1ØDR1ØDL1ØBM165,114R7UL7BR1
5R7DL7BM17Ø,129F3R6E3BM176,1Ø2D3
FDFDRDRDR8F2DFD6FRBM176,1Ø4DGDGD
LDLDL8G2DGD6GLBM166,135GDGDGLGL9
GLGLGDGDGDGDGDG3BM187,135FDFDFRF
R9FRFRFDFDFDFBM165,135
63Ø DRAW"M176,16ØM189,135D12L5F4
M176,161LBM164,137D1ØR4G4M176,16
Ø":CIRCLE(177,142),4:PAINT(177,1
42),Ø,Ø:DRAW"BM177,139DF2DF2DF2B
M177,139DG2DG2DG2BM175,142D14RU1
4RD15RU14":PAINT(177,1Ø3),Ø,Ø:PA
INT(175,1Ø3),Ø,Ø:PSET(169,117):P
SET(183,117)
64Ø DRAW"BM149,96"+B$+U$+L$+L$+E
$+T$+I$+N$:FOR X=1 TO 2ØØØ:NEXT:
GOSUB83Ø:DRAW"BM142,96"+N$+E$+W$
+S$+SP$+F$+L$+A$+S$+H$:DRAW"BM15
Ø,124"+R$+A$+I$+N$+B$+O$+W$:DRAW
"BM15Ø,132"+M$+A$+G$+A$+Z$+I$+N$
+E$:DRAW"BM141,14Ø"+N$+A$+M$+E$+
D$+SP$+N$+O$+PD$+SP$+N1$
65Ø DRAW"BM155,148"+A$+G$+A$+I$+
N$+PD$:FOR X=1 TO 3:QZ$="1;2;3;4
;5;6;7;8;9;1Ø;11;12;":PLAY"T9Ø5L
24V15;":PLAY QZ$:NEXT:GOTO 45Ø
66Ø GOSUB 83Ø:DRAW"BM143,1ØØ"+C$
+A$+R$+SP$+C$+L$+I$+N$+I$+C$:DRA
W"BM141,14ØR68":CIRCLE(157,135),
5:CIRCLE(192,135),5:CIRCLE(157,1
35),1:CIRCLE(192,135),1:DRAW"BM1
62,136R25BM147,136NR3U2ØR3ØF7R18
D12L3BM15Ø,118ND6R17D6L17R9U6BR1
1ND6R5F6ND12L11D12
67Ø FOR X=1 TO 15ØØ:NEXT:EXEC 43
345:DRAW"BM184,123E13":FOR K=2Ø
TO 45 STEP.2:POKE 14Ø,RND(4Ø)+K:
EXEC 43345:NEXT:GOTO 45Ø
68Ø GOSUB 83Ø:DRAW"BM141,12ØM158
,11ØND2M164,1Ø4D6M171,116D2M178,
132D2GDGLGLGNH5LGL2HLHUHUHLHLHL2
HL2HUHU2HU2BM141,146E15":CIRCLE(
167,12Ø),2:CIRCLE(175,133),2:DRA
W"BM142,12ØND1ØBR3BU2ND1ØBR3BU2N
D1ØBR3BU2ND1ØBR3BU2ND1ØBR3BU2D1Ø
BM179,134R3ØBL48L8
69Ø DRAW"BM186,134E12G3H3F12BM19
3,134UERFDBM186,134BE9BU4H2U2E4U
2H4U2E4U2":FOR X=1 TO 2:PLAY"O4T
4L9CP8L12CCP8L12CCP8L12CGP8L12EG
P8L12EGP8L12E":NEXT:PLAY"L3C":GO
TO 45Ø
7ØØ GOSUB 83Ø:DRAW"BM147,1ØØ"+L$
+O$+V$+E$+SP$+B$+O$+A$+T$:DRAW"B
M148,128NR48R8E3NR26E6R4G2R22G3N
D3E3L4E7L5G7L5E7L5G7R19G3D3R12D7
L44H5BM14Ø,134"+WA$:GOTO 45Ø
71Ø GOSUB 83Ø:DRAW"BM145,125CØU3
ØR1ØF15E15R1ØD3ØL12U18G13H13D18L
12BM187,116E8G4FRFRFRFRFRFBM198,
1Ø7FRFRFRFU1ØBM14Ø,14ØR68BD3L68B
D3R68BD3L68BD3R68":CIRCLE(15Ø,14
2),2:CIRCLE(158,149),2:CIRCLE(16
6,152),2:CIRCLE(174,146),2:DRAW"
BM151,142U8BM159,149U8
72Ø DRAW"BM167,152U8BM175,146U8B
M18Ø,14ØD12":CIRCLE(187,15Ø),2:C
IRCLE(195,155),2:CIRCLE(2Ø3,145)
,2:DRAW"BM188,15ØU8BM196,155U8BM
2Ø4,155U8":PLAY"V15T8L4O3CEGO4CP
4O3GO4L3C":GOTO 45Ø
73Ø GOSUB83Ø:DRAW"BM15Ø,1ØØCØ"+D
$+R$+A$+G$+N$+E$+T$:DRAW"BM172,1
1ØM176,117M185,117M177,122M18Ø,1
29M172,126M165,129M168,122M161,1
17M169,117M172,11ØBM149,15Ø"+S$+
T$+A$+R$+R$+I$+N$+G$:DRAW"BM145,
157"+J$+A$+C$+K$+SP$+W$+E$+B$+B$
74Ø CIRCLE(172,12Ø),13:SOUND 1,1
2:SOUND 34,4:SOUND 44,2:FOR X=1
TO 34Ø:NEXT:SOUND 1,9:FOR X=1 TO
 7ØØ:NEXT:SOUND 1,12:SOUND 34,4:
SOUND 44,2:FOR X=1 TO 34Ø:NEXT:S
OUND 1,9:SOUND 79,12:GOTO 45Ø
75Ø GOSUB 83Ø:DRAW"BM145,1ØØCØ"+
N3$+SP$+S$+T$+O$+O$+G$+E$+S$:CIR
CLE(153,13Ø),1Ø:CIRCLE(175,13Ø),
1Ø:CIRCLE(197,13Ø),1Ø:DRAW"BM145
,128R16":PAINT(146,126),Ø,Ø:CIRC
LE(167,128),2:CIRCLE(168,125),2:
CIRCLE(169,122),2:CIRCLE(183,128
),2:CIRCLE(182,125),2
76Ø CIRCLE(181,122),2:PSET(153,1
33):PSET(175,133):PSET(197,133):
PSET(149,129):PSET(157,129):PSET
(171,129):PSET(179,129):PSET(193
,129):PSET(2Ø1,129):DRAW"BM14Ø,1
42E6BR13F3ND21E3BR16F4ND21E4BR15
F3BM147,155"+M$:DRAW"BM172,155"+
L$:DRAW"BM195,155"+C$
77Ø DRAW"BM15Ø,136R6BR16R6BR16R6
78Ø SOUND 125,6:SOUND 133,2:SOUN
D 146,12:SOUND 146,4:SOUND 17Ø,8
:SOUND 159,6:SOUND 146,2:SOUND 1
46,4:SOUND 175,5:FOR X=1 TO 35Ø:
NEXT X:SOUND 159,6:SOUND 146,2:S
OUND 146,4:SOUND 17Ø,5:FOR X=1 T
O 35Ø:NEXT X:SOUND 159,6:SOUND 1
46,2:SOUND 146,4
79Ø SOUND 175,5:FOR X=1 TO 35Ø:N
EXT X:SOUND 125,6:SOUND 133,2:SO
UND 146,12:SOUND 146,4:SOUND 17Ø
,8:SOUND 159,6:SOUND 146,2:SOUND
 146,4:SOUND 175,5:FOR X=1 TO 52
5:NEXT X:PLAY"O3T2V25L8EFAAAAGFD
":SOUND 89,3:SOUND 1Ø9,3:SOUND 1
25,3:SOUND 1Ø9,3
8ØØ FOR X=1 TO 12Ø:NEXT X:SOUND
125,3:SOUND 89,3:FOR X=1 TO 2ØØ:
NEXT X:SOUND 175,2:GOTO 45Ø
81Ø GOSUB 83Ø:PAINT(142,91),Ø,Ø:
ZZ=Ø:GOTO 45Ø
82Ø GOSUB 83Ø:GOTO 45Ø
83Ø LINE(14Ø,9Ø)-(2Ø8,16Ø),PRESE
T,BF:ZZ=1:RETURN
```

32K Disk

CoCocad Modification

"CoCocad: The Schematic Scoundrel" (October 1985, Page 130): Peter Kerckhoff writes to tell us that some other printers besides the Gemini-10X can be used with this program.

First of all, there is a rather roundabout method that works with any printer that can be used to make screen dumps of regular CoCo graphics, if you have a screen dump program for your printer. Delete Line 1990 from the main program. When you request a printout, *CoCocad* will now save nine screen files on the disk. Each has the name *PRT.PGn*; 'n' is the page number (so *PRT.PG3* would be the third page).

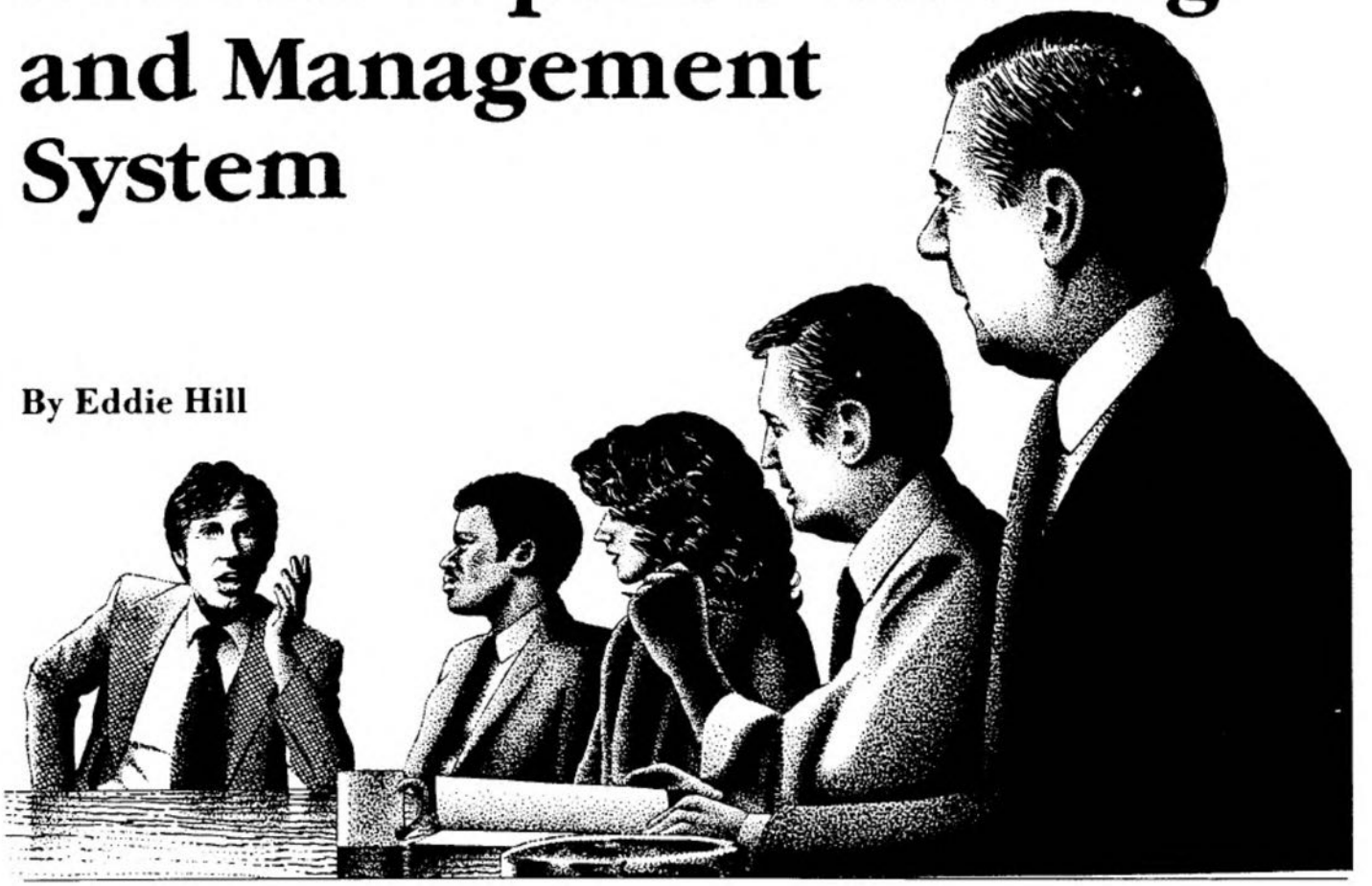
Now load in your screen print program and type the command PMODE 4,1:SCREEN1,1 and press ENTER. Then, for each file type LOADM"PRT.PGn", and after the file is loaded activate the screen print routine. Once you have printed all nine pictures, cut them out and tape them together.

Peter also included a new version of the *Cadprint* printer driver that is designed for the Tandy/Radio Shack DMP series printers. Here it is:

The listing: CADPRINT

```
1Ø 'CADPRINT VR1.Ø BY P.KERCKHOF
F 4335 HENDRIX WAY SAN JOSE CA -
 1985 RAINBOW MAG (OCT)
2Ø 'MODIFIED FOR USE WITH RADIO
SHACK DMP-12Ø PRINTER BY DAVISSO
N ON 1Ø/15/85
3Ø CLEAR1ØØ,&H379A:CLS:PRINT"cad
print RUNNING":FORX=&H379B TO &H
37FA:READ A:POKEX,A:NEXTX
4Ø POKE15Ø,41 '12ØØ BAUD
5Ø PRINT#-2,CHR$(18)
6Ø FOR PG=ØTO8STEP3
7Ø LOADM"PRT.PG"+RIGHT$(STR$(PG)
,1),&H2AØØ
8Ø LOADM"PRT.PG"+RIGHT$(STR$(PG+
1),1),&H42ØØ
9Ø LOADM"PRT.PG"+RIGHT$(STR$(PG+
2),1),&H5AØØ
1ØØ EXEC&H379B
11Ø KILL"PRT.PG"+RIGHT$(STR$(PG)
,1):KILL"PRT.PG"+RIGHT$(STR$(PG+
1),1):KILL"PRT.PG"+RIGHT$(STR$(P
G+2),1):NEXT PG
12Ø PRINT:PRINT"DONE.":PRINT#-2,
CHR$(3Ø):STOP
13Ø DATA 134,254,151,111,134,1,1
42,126,224,48,134,52,2,141,38,53
,2,142,1Ø2,224,48,134,52,2,141,2
7,53,2,142,78,224,48,134,52,2,14
1,16,134,13,173,159,16Ø,2,53,2,7
6,129,31,38,212,15,111,57
14Ø DATA 198,156,52,2Ø,23Ø,132,1
6,142,Ø,8,16,191,55,249,88,7Ø,12
2,55,25Ø,16,19Ø,55,249,38,245,67
,138,128,173,159,16Ø,2,53,2Ø,48,
136,224,9Ø,38,218,57,Ø,Ø
```

Annual Expense Tracking and Management System



By Eddie Hill

This system (set of three programs) allows an individual, or possibly a small business, to maintain and track expenses for one year. A maximum of 135 transactions per month for up to 100 account codes may be tracked by month for an entire year (12 month period). Each expense you wish to track must have a numeric code in the range of 1 through 100 with a description not exceeding 27 characters. The system provides for screen display and printing of data and reports and is designed for use on a 64K CoCo with one disk drive and a DMP-100 printer. In addition to maintaining actual transaction data, the system provides budgetary analysis for accounts over the year. This feature allows for comparative analysis of actual versus budget for an account (display and printout). It should be noted that budget or transaction summary amounts for an account cannot exceed $99,999.99 and a detailed transaction entry for an account cannot be greater than $9,999.99.

Before proceeding, a discussion of a few basic concepts employed in the design of the system and programs is warranted. The programs feature extensive use of arrays which facilitate fast display of data and fast data entry and maintenance. The disk file access methods are extremely simple and straightforward and, for the most part, use array concepts. This extensive utilization of arrays in the programs yields a system which is both efficient and inefficient, but overall it provides an effective and simple-to-use-and-understand method for tracking expenses.

The design of the system requires that an entire diskette be dedicated for the recording of budgeted and actual expenditures for a year. If you utilize diskette backup for your files and system programs, then two diskettes will be required for one year of data (assuming you have only one backup diskette). Although the system can be used without a printer, one is highly recommended to achieve best results.

As mentioned previously, the system consists of three programs. One program (*Crexpfle*) creates the basic files required for system utilization. The system will not run until this program has been successfully executed. The second program (*Exptrakr*) allows the entry and maintenance of all budget and actual data. It also permits various screen displays of the data (both budget and actual). The third program (*Reptgenr*) prints various listings of budget and actual data. Execution of this program can be independent of *Exptrakr* or may be selected from the main menu of *Exptrakr*. After execution of *Reptgenr* you may return to *Exptrakr* by exercising the appropriate selection option from the main menu of *Reptgenr*. You may freely transfer between *Exptrakr* and *Reptgenr* or run either as a stand-alone program. It is important that you name the programs "*Exptrakr*" and "*Reptgenr*" because these are the names used in the call routines for the programs.

Both *Exptrakr* and *Reptgenr* are menu driven programs

Exhibit 1

MONTHLY TRANSACTIONS FOR JANUARY 1986

REC NUM	ACT NUM	.....ACCOUNT DESCRIPTION.....	TRAN I.D.	DA	..AMOUNT..	TRANS. DESCRIPTION
1	1	HOUSE MORTGAGE	1ØØ	31	2ØØ.ØØ	PAYMENT
2	2	INSURANCE	5Ø	31	1ØØ.ØØ	PREMIUM
3	3	BANK CARDS	125	31	15Ø.ØØ	PAYMENT
4	4	ELECTRICITY	15Ø	31	125.ØØ	BILL
5	5	AUTO PAYMENT	11Ø	31	18Ø.ØØ	PAYMENT #1Ø
6	6	GROCERIES	9Ø	15	175.ØØ	W.D.
7	7	CLOTHING	95	25	8Ø.ØØ	JIM'S
8	8	MISCELLANEOUS	98	28	75.ØØ	SOFTWARE
9	6	GROCERIES	99	31	245.ØØ	J&D GROCERY
1Ø	7	CLOTHING	134	31	3Ø.ØØ	SHOES
		TOTAL---------->		$	1,36Ø.ØØ	

with submenus, instructions and comments as required. This approach offers easy access to (and exits from) routines within the programs. As with all BASIC programs, if you wish to exit a routine before completion, you may hit the BREAK key. Use extreme caution when exercising this option. An entire file or files can be easily "garbaged." Therefore, it is not recommended. You may want to consider a BREAK disable routine in the programs.

Crexpfle

Crexpfle creates the basic files for the system. It merely formats and sizes the following files:

Budget Summary
Transaction Summary
Chart of Accounts
Detail Transaction Filenames

It must be executed before attempting to run *Exptrakr* or *Reptgenr.*

Exptrakr

This program is the heart of the system. It allows for entry and maintenance of all data utilized by the system. This includes charts of accounts, budget and transaction data. *Exptrakr* opens up with a main menu consisting of 14 options. The main menu appears as shown below.

1) Add/change/input budget
2) Account YTD summary trans.
3) Account actual vs. budget
4) Account budget for year
5) Chart of accounts maint.
6) Display chart of accounts
7) Add/chg/del/input trans.
8) Display monthly budget
9) Display monthly trans.
10) Display actual vs. budget
11) Report generator
12) File deletion
13) Backup files
14) End session

A discussion of each option follows.

1) Add/Change/Input Budget

Allows input and maintenance of budget data for an account. You select the month you wish to enter or change by entering the appropriate number for the month (1-12). If you wish to enter or change data for all months, enter "99." You return to the main menu by entering '0'.

2) Account YTD Summary Transaction

A display of a specified account showing summary amounts by month. Pressing ENTER returns you to the main menu.

3) Account Actual vs. Budget

Permits display of summarized actual versus budget amounts for a month or year-to-date through a given month.

4) Account Budget For Year

Displays the yearly budget month by month for an account.

5) Chart of Accounts Maintenance

As stated earlier, each expense you wish to track must have a numeric code in the range 1 through 100 and a description not exceeding 27 characters. The first five positions of the description cannot be "XXXXX" since this denotes to the system that the account has not been established for use. If you inadvertently enter a description with more than 27 characters or "XXXXX" in the first five positions, the system will prompt you to re-enter the description.

This selection gives you three options as follows:

Option 1 — Allows for the entry of account descriptions in numeric order. You are first prompted for the description of Account 1, then Account 2 and so forth until you have entered descriptions for all 100 accounts. If you wish to terminate entry of descriptions at any point before Account 100, merely type THATS ALL and press ENTER. The system will automatically return to the main menu just as it does when the description for Account 100 has been entered.

Option 2 — Allows for the addition or changing of account descriptions. The same procedure is used to add

Exhibit 2

BUDGET FOR JANUARY 1986

ACT	<--D E S C R I P T I O N-->	..AMOUNT..
1	HOUSE MORTGAGE	2ØØ.ØØ
2	INSURANCE	5Ø.ØØ
3	BANK CARDS	1ØØ.ØØ
4	ELECTRICITY	15Ø.ØØ
5	AUTO PAYMENT	18Ø.ØØ
6	GROCERIES	4ØØ.ØØ
7	CLOTHING	1ØØ.ØØ
8	MISCELLANEOUS	5Ø.ØØ
	TOTAL	1,23Ø.ØØ

or change an account description. A prompt appears that asks for the account number. After entering a valid account number the account number and description will be displayed. You will be asked if this is the account you wish to add or change. If so, type YES and press ENTER. If not, type NO and ENTER. After entering a changed or added account description you will be asked if you wish to add or change any more account descriptions. The process will repeat as long as you respond "yes." A "no" response will return you to the main menu.

Option 3 — Choosing this option will return you to the main menu.

6) Display Chart of Accounts

Allows for the display of all account numbers with their descriptions (Option 1) or for a single account number with its description (Option 2). If an account description is all X's, it has not been set up for use by the system. Option 3 returns you to the main menu.

7) Add/Chg/Del/Input Transaction

Permits the entry and maintenance of detail transactions (expenses) for a selected month.

Option 1 — This allows the entry of detail expenses for a selected month. You are asked for the number of the month (1-12). The program then checks to see if transactions have already been entered for the month selected. If so, you cannot re-enter the transactions unless you delete the entire month's transaction file (see Selection 12, File deletion, for instructions). If transactions for the month you wish to enter are not on file, the program allows you to continue and prompts you through entry of your expenses. Please notice the program assigns each entry a record number which will be used for changing or deleting transactions, if necessary, in the future. Each addition to a month's transaction must be assigned a record number. This must be an unused number in the range 1-100. More on changes, deletions and additions later.

The entry of detail transactions requires a valid account number for each transaction. Therefore, you must set your chart of accounts file up prior to using this option. Other information you must enter is as follows:

Date — This is the day the transaction occurred.

Trans. I.D. — A one- to four-digit identifying tag (e.g., check number). May be alphabetic or numeric.

Amount — The amount of the transaction (not to exceed $9,999.99) may be debit or credit.

Description — This is an optional identification, not exceeding 14 characters, for the transaction (e.g., monthly payment).

When the entry of the detail transaction is complete, press ENTER. This writes the transaction to the file and returns to a fresh transaction entry screen for input of the next transaction. Upon completing the entry of all detail transactions for the month, enter "ZZZ" in the

Exhibit 3

BUDGET FOR 1986	JAN MAY SEP	FEB JUN OCT	MAR JUL NOV	APR AUG DEC
HOUSE MORTGAGE	200.00	200.00	200.00	200.00
	200.00	200.00	200.00	200.00
1 TOTAL..$ 2,400.00	200.00	200.00	200.00	200.00
INSURANCE	50.00	50.00	50.00	50.00
	50.00	50.00	50.00	50.00
2 TOTAL..$ 600.00	50.00	50.00	50.00	50.00
BANK CARDS	100.00	100.00	100.00	100.00
	100.00	100.00	100.00	100.00
3 TOTAL..$ 1,200.00	100.00	100.00	100.00	100.00
ELECTRICITY	150.00	150.00	150.00	150.00
	150.00	150.00	150.00	150.00
4 TOTAL..$ 1,800.00	150.00	150.00	150.00	150.00
AUTO PAYMENT	180.00	180.00	180.00	180.00
	180.00	180.00	180.00	180.00
5 TOTAL..$ 2,160.00	180.00	180.00	180.00	180.00
GROCERIES	400.00	400.00	400.00	400.00
	400.00	400.00	400.00	400.00
6 TOTAL..$ 4,800.00	400.00	400.00	400.00	400.00
CLOTHING	100.00	100.00	100.00	100.00
	100.00	100.00	100.00	100.00
7 TOTAL..$ 1,200.00	100.00	100.00	100.00	100.00
MISCELLANEOUS	50.00	50.00	50.00	50.00
	50.00	50.00	50.00	50.00
8 TOTAL..$ 600.00	50.00	50.00	50.00	50.00
TOTAL BUDGET FOR YEAR	1,230.00	1,230.00	1,230.00	1,230.00
	1,230.00	1,230.00	1,230.00	1,230.00
999 TOTAL..$ 14,760.00	1,230.00	1,230.00	1,230.00	1,230.00

Exhibit 4

ACTUAL FOR 1986		JAN MAY SEP	FEB JUN OCT	MAR JUL NOV	APR AUG DEC
HOUSE MORTGAGE		2ØØ.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
1 TOTAL..$	2ØØ.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
INSURANCE		1ØØ.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
2 TOTAL..$	1ØØ.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
BANK CARDS		15Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
3 TOTAL..$	15Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
ELECTRICITY		125.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
4 TOTAL..$	125.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
AUTO PAYMENT		18Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
5 TOTAL..$	18Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
GROCERIES		42Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
6 TOTAL..$	42Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
CLOTHING		11Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
7 TOTAL..$	11Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
MISCELLANEOUS		75.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
8 TOTAL..$	75.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
TOTAL ACTUAL FOR YEAR		1,36Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
		Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ
999 TOTAL..$	1,36Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ	Ø.ØØ

account number field and press ENTER. This will complete the writing of the transaction file and post transaction summary totals for the month. After this is complete you are returned to the data entry submenu.

Option 2 — Allows the entering of new information for a transaction which has already been entered. You must specify the month and the record number of the transaction (shown on detail printout of month's transactions) you wish to change. A check is made to ensure that you have entered a correct record number. If the record number is valid, the transaction is displayed. Next, you are asked if it is the one you wish to change. If so, all of the data must be re-entered for the transaction (just as if it were new). Upon completion of entry of the new data for the transaction, press ENTER. This records the transaction on the file and asks if you wish to change more transactions. If you answer 'Y' then the process repeats, else the transaction summary file is updated and you are returned to the data entry submenu.

Exhibit 5

TRANSACTION SUMMARY FOR 1986 ....A/Ø Ø1/31/86

ACT	....ACCOUNT DESCRIPTION....	...AMOUNT...
1	HOUSE MORTGAGE	2ØØ.ØØ
2	INSURANCE	1ØØ.ØØ
3	BANK CARDS	15Ø.ØØ
4	ELECTRICITY	125.ØØ
5	AUTO PAYMENT	18Ø.ØØ
6	GROCERIES	42Ø.ØØ
7	CLOTHING	11Ø.ØØ
8	MISCELLANEOUS	75.ØØ
	TOTAL	$ 1,36Ø.ØØ

Option 3 — This option will add new transactions (e.g., overlooked, not available at time data for month was entered) to a month's file. You must specify the record number for the record to be added. This may be derived by looking at a detail printout for the month and adding one to the record number shown for the last transaction. The program will not permit the addition of a transaction with an existing record number, as it checks this before allowing you to proceed. Once it is determined that you are not trying to add a duplicate record, you are presented with the data entry screen. The procedure from this point is the same as for entering change data in Option 2 of this selection. Instead of referring to transaction changes, the prompts relate to transaction additions.

Option 4 — Allows you to delete a transaction from a month's file (e.g., a transaction was included in the wrong month). Again you must work with a record number,

Exhibit 6

ACTUAL VS. BUDGET FOR JANUARY 1986

ACT	...D E S C R I P T I O N...	...ACTUAL..	...BUDGET..	..VARIANCE.
1	HOUSE MORTGAGE	2ØØ.ØØ	2ØØ.ØØ	Ø.ØØ
2	INSURANCE	1ØØ.ØØ	5Ø.ØØ	5Ø.ØØ-
3	BANK CARDS	15Ø.ØØ	1ØØ.ØØ	5Ø.ØØ-
4	ELECTRICITY	125.ØØ	15Ø.ØØ	25.ØØ
5	AUTO PAYMENT	18Ø.ØØ	18Ø.ØØ	Ø.ØØ
6	GROCERIES	42Ø.ØØ	4ØØ.ØØ	2Ø.ØØ-
7	CLOTHING	11Ø.ØØ	1ØØ.ØØ	1Ø.ØØ-
8	MISCELLANEOUS	75.ØØ	5Ø.ØØ	25.ØØ-
	TOTAL ---->	$ 1,36Ø.ØØ	$ 1,23Ø.ØØ	$ 13Ø.ØØ-

Exhibit 7

YEAR TO DATE A/O JANUARY 1986

ACT	...D E S C R I P T I O N...	...ACTUAL..	...BUDGET..	..VARIANCE.
1	HOUSE MORTGAGE	2ØØ.ØØ	2ØØ.ØØ	Ø.ØØ
2	INSURANCE	1ØØ.ØØ	5Ø.ØØ	5Ø.ØØ-
3	BANK CARDS	15Ø.ØØ	1ØØ.ØØ	5Ø.ØØ-
4	ELECTRICITY	125.ØØ	15Ø.ØØ	25.ØØ
5	AUTO PAYMENT	18Ø.ØØ	18Ø.ØØ	Ø.ØØ
6	GROCERIES	42Ø.ØØ	4ØØ.ØØ	2Ø.ØØ-
7	CLOTHING	11Ø.ØØ	1ØØ.ØØ	1Ø.ØØ-
8	MISCELLANEOUS	75.ØØ	5Ø.ØØ	25.ØØ-
	TOTAL ---->	$ 1,36Ø.ØØ	$ 1,23Ø.ØØ	$ 13Ø.ØØ-

which may be obtained from the printout of the detail transactions for the month. You are asked for the deletion record number, then the program checks to see if it is valid. If so, it displays the record for a visual verification. You are asked if this is the record you wish to delete. A response of 'N' prompts you for another record number while a 'Y' response deletes the record and updates the appropriate files. The steps from this point follow the same logic as options 2 and 3 in this selection, except comments that relate to deletions.

Option 5 — Returns to the main menu. It should be noted that options 1 through 4 for this selection allow for submenu return in case you chose an option in error.

8) Display Monthly Budget

Displays budgeted account expenditures for a selected month.

9) Display Monthly Transaction

Displays the summary amount of each account's transactions for a selected month.

10) Display Actual vs. Budget

You have the choice of choosing a specific month or year-to-date through a specific month. The submenu will guide you.

11) Report Generator

This selection allows you to print out reports from the data you have entered. This is a separate program (*Reptgenr*, program Listing 3) and is called from this menu choice. You are given seven print selections plus a selection to return to the main program (*Exptrakr*) or to terminate the session. Details on this selection are covered in the comments on program Listing 3.

12) File Deletion

This selection allows you to delete either a budget or transaction file for a specified month. The appropriate transactions summaries are adjusted. After completion of this option, data for a given month (budget or transaction) may be re-entered.

13) Backup Files

The method used is the single disk copy procedure provided for but not documented in the Radio Shack manuals. I recommend at least a two-generation backup system (backup of current files, plus prior generation).

14) End Session

Choose this option when you are ready to end the program (*Exptrakr*). Remember to backup your files.

Reptgenr

This program provides for various listings of your data. A discussion of the selections follows:

1) List Monthly Transactions

This selection gives you a listing of all the transactions for a specified month. The listing is the only place where this detail is given. Record numbers are obtained from this listing. See Exhibit 1.

2) List Monthly Budget

This option provides a listing of the budget amounts for each account with budgeted expenditures in a specified month. See Exhibit 2.

3) List Budget For Year

Provides a month-by-month listing of each account's budgeted amounts for the year. See Exhibit 3.

4) List Actual For Year

Provides a month-by-month listing of each account's actual amounts for the year. See Exhibit 4.

5) List Transaction Summary For Year

Provides a summarized listing of all account amounts entered. See Exhibit 5.

continued on Page 28

RECEIPT MAKER AND FILE

by Bill Tottingham

Once you've had the CoCo for a while, as most of you have probably experienced, you begin to look for ways to use it for about anything you can think of, if for no other reason than to prove your computer is more than "just a toy", as the guy down the street the the $5000 IBM says. Any of us who have spent any time with the CoCo know that's a ridiculous notion. I know several people who run small volume businesses who would love to use the CoCo due to its relatively low cost. One particular person wanted a program that not only printed out a receipt (he was writing them out by hand), but also would save the information to disk for later tax purposes. This program is the result - no need for a million dollar computer and another million in software.

To use Receipt File you must have at least 32K and a disk, and should have a printer. The program is set to use a DMP-100 at 1200 Baud. If you are going to use a different printer, change the received control codes in the following lines:

Line	DMP-100	Function
830	POKE149,0:POKE150,41	1200 Baud printer
850	CHR$(15)	Enable underline
860	CHR$(14)	Disable underline
870	CHR$(31)	Elongated mode
880	CHR$(30)	Character mode

Entering Data

After making any needed changes, you are ready to run the program. The first thing you will see is the main menu. On it will be five choices (8see Ffigure 1). Since we're just getting started, press '2' for "Enter new data." You will then be asked the date. Enter it in the same format the example shows, then press ENTER. You will then be asked if the information given was correct. (When entering new data you will always be asked if the information provided was correct.)

Next you will be asked for a "Receipt number." *You must enter a number.* This is the number the program uses for filing. It also must be a number different from one already on file. The best bet is to give the receipt a number incremented by one over the preceding receipt. For the first one, enter 001.

You will now be asked to enter "Received of." Here you may enter the name of your customer.

Next comes the "Dollar amount paid." This is exactly what it says. Enter this like the example shows.

Now we come to "Amount of Account." This is the total purchase price. You can use the balance due from any previous payment of the same account. This will be printed on the receipt with the amount paid and a balance due. If the amount of account is the same as the amount paid, simply enter the amount paid here again if you wish. If not, press ENTER.

Finally we come to the comment line. Enter anything you wish; however, it is customary to use this for what was purchased, or the nature of the transaction. If you are the purchaser, you might want to enter the name of the other party, so if the hard copy is lost you will still have the information come tax time. There is a 30 character space here.

You now find yourself back at the main menu. If you want to examine the information to be printed and/or saved, press '3'. If any of the information is wrong, press 'M' to return to the main menu and press '2' to re-enter all the data. If everything is correct, the receipt can be printed or saved to disk. Press 'M' to get back to the menu.

Saving to Disk

If you are planning to save this data on disk, it can be done now or after printing the receipt.

To save on disk, press '1' for the disk menu (Figure 2). Here again there will be five choices. Since we are saving data, press '3'. You will then be asked

to press ENTER for save or 'M' for menu. After the save is completed, you will be returned to the main menu.

Printing the Receipt

To print the receipt, press '4'. You will be asked if you want a copy. Pressing 'Y' tells the printer to print two receipts. Before answering this prompt make sure your printer is on and ready to go.

After printing, you will be returned to the main menu. Here you may save the data (if not previously saved), enter new data or look at and/or load previous accounts.

Entering Data from Disk

To load data from disk, first get to the disk menu by pressing '1' on the main menu. The easiest way to load is to press '1' from the disk menu. A list of receipt numbers will appear on the screen preceded by a number; for example: 1) 001. To load receipt #001, press '1' and ENTER and the file will load. If you already know the receipt number, press '2' on the disk menu and enter the number at the prompt.

After loading, you will automatically be in the examine mode (Figure 3). From there you can print out a receipt, load a different receipt or enter new data.

As with all programs, this one can be modified to your specific needs. Some ideas might be to incorporate a different filing system if you are in a higher volume business. To enter data faster, you could hack off everything after the line input statements in lines 240 through 270 and in Line 280 after the BD$=STR$(BD) statement. A line could also be added in the printer routine that would print a line under the receipt for written comments. □

Figure 1

```
          MENU

[1]       DISK MENU

[2]       ENTER NEW DATA

[3]       EXAMINE

[4]       PRINT RECEIPT

[Q]       QUIT PROGRAM
```

Figure 2

```
        DISK MENU

[1]       DIRECTORY OF FILES

[2]       LOAD FROM DISK

[3]       SAVE TO DISK

[M]       MAIN MENU

[Q]       QUIT PROGRAM
```

Figure 3

```
#001                          6/25/86

JOHN Q. SMITH

AMT. OF ACCOUNT             $302.43

AMT. PAID                   $259.99

BALANCE DUE                  $42.44

COMMENTS: PART # 35771

        PRESS [M] FOR MENU
```

Figure 4

NO. 001 Date 6/25/86

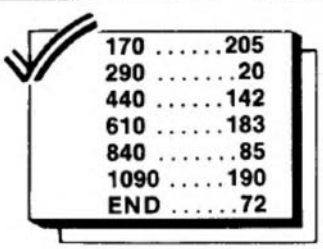
Received of JOHN Q. SMITH

$259.99 100 DOLLARS

AMT. OF ACCOUNT 302.43
AMT. PAID 259.99
BALANCE DUE 42.44

CMTS PART #35771

$



The listing: RECEIPT

```
1Ø CLEAR
2Ø CLEAR5ØØØ
3Ø VERIFY ON
4Ø DIMP$(35,2)
5Ø CLS:PRINT@46,"MENU":PRINT@165
,"[1]     DISK MENU":PRINT@229,"
[2]     ENTER NEW DATA":PRINT@29
3,"[3]     EXAMINE":PRINT@357,"[
4]     PRINT RECEIPT":PRINT@421,
"[Q]     QUIT PROGRAM"
6Ø AN$=INKEY$:IFAN$=""GOTO6Ø
7Ø IFAN$="Q"GOTO13Ø
8Ø IFAN$="1"GOTO14Ø
9Ø IFAN$="2"GOTO22Ø
1ØØ IFAN$="3"GOTO52Ø
11Ø IFAN$="4"GOTO76Ø
12Ø GOTO6Ø
13Ø GOSUB57Ø:CLS:END
14Ø CLS:PRINT@44,"DISK MENU":PRI
NT@165,"[1]     DIRECTORY OF FIL
ES":PRINT@229,"[2]     LOAD FROM
 DISK":PRINT@293,"[3]     SAVE T
O DISK":PRINT@357,"[M]     MAIN
MENU":PRINT@421,"[Q]     QUIT PR
OGRAM"
15Ø I$=INKEY$:IFI$=""GOTO15Ø
16Ø IFI$="1"GOTO11ØØ
17Ø IFI$="2"GOTO45Ø
18Ø IFI$="3"GOTO37Ø
19Ø IFI$="M"GOTO5Ø
2ØØ IFI$="Q"GOTO13Ø
21Ø GOTO15Ø
22Ø 'ENTER DATA ROUTINE
23Ø GOSUB57Ø
24Ø CLS:PRINT@1Ø3,"ENTER RECEIPT
 DATE":PRINT@136,"(EX. 6/25/85)"
:LINEINPUT"          ";D$:GOSUB
34Ø:PRINT@32Ø,"      -"D$"-":GOS
UB32Ø:IFI$="N"GOTO24Ø
25Ø CLS:PRINT@1Ø3,"ENTER RECEIPT
 #":PRINT@136,"(EX. ØØ1)":LINEIN
PUT"          ";N$:GOSUB34Ø:PRIN
T@32Ø,"     -"N$"-":GOSUB32Ø:IFI
$="N"GOTO25Ø
26Ø CLS:PRINT@1Ø3,"ENTER RECEIVE
D OF":PRINT@134,"(EX. JOHN Q. SM
ITH)":LINEINPUT"    ";RO$:GOSUB34
Ø:PRINT@32Ø,"     -"RO$"-":GOSUB
32Ø:IFI$="N"THEN26Ø
27Ø CLS:PRINT@1ØØ,"ENTER DOLLAR
AMOUNT PAID":PRINT@134,"(EX. 259
.99 OR 499)":LINEINPUT"
";DO$:DO=VAL(DO$):GOSUB34Ø:PRINT
@322,"  -";:PRINTUSING"$$###.##"
;DO;:PRINT"-":GOSUB32Ø:IFI$="N"T
HEN27Ø
28Ø CLS:PRINT@99,"ENTER TOTAL AM
T. OF ACCOUNT":PRINT@135,"(EX. 3
Ø2.43 OR 5ØØ)":LINEINPUT"
    ";TA$:TA=VAL(TA$):BD=(TA-DO)
:BD$=STR$(BD):GOSUB34Ø:PRINT@323
,"  -";:PRINTUSING"$$####.##";TA
;:PRINT"-":GOSUB32Ø:IFI$="N"THEN
28Ø
29Ø CLS:PRINT@1Ø2,"ENTER COMMENT
 LINE":PRINT@134,"(EX. PART #357
71)":PRINT@223,"]":PRINT@192,"["
;:LINEINPUTC$
3ØØ IFLEN(C$)>3ØGOTO36Ø
31Ø GOSUB57Ø:GOTO5Ø
32Ø I$=INKEY$:IFI$=""THEN32Ø
33Ø RETURN
34Ø PRINT@29Ø,"YOU ENTERED-":PRI
NT@354,"IS THAT CORRECT? (Y/N)":
RETURN
35Ø GOTO5Ø
36Ø CLS:PRINT@17Ø,"LINE TO LONG"
:FOR T=1TO5ØØ:NEXTT:GOTO29Ø
37Ø 'DISK ROUTINE
38Ø BD$=STR$(BD)
39Ø CLS:PRINT@166,"HIT <ENTER> T
O SAVE":PRINT@232,"OR <M> FOR ME
NU"
4ØØ I$=INKEY$:IFI$=""GOTO4ØØ
41Ø IFI$="M"GOTO5Ø
42Ø IFI$=CHR$(13)GOTO44Ø
43Ø GOTO4ØØ
44Ø GOSUB57Ø:CLS:GOSUB58Ø:GOSUB7
4Ø:GOSUB57Ø:GOTO5Ø
45Ø CLS:PRINT@17Ø,"ENTER FILE #"
:PRINT@33Ø,"[M] FOR MENU"
46Ø PRINT@242,"]":PRINT@236,"[";
:LINEINPUTN$
47Ø Z=LEN(N$):IFZ>5GOTO45Ø
48Ø IFN$="M"GOTO5Ø
49Ø CLS:PRINT@168,"LOADING #";N$
```

```
:PRINT@182,STRING$(1Ø,32)
5ØØ GOSUB57Ø:GOSUB58Ø:GOSUB75Ø:G
OSUB67Ø
51Ø TA=VAL(TA$):DO=VAL(DO$):BD=V
AL(BD$)
52Ø BD=(TA-DO)
53Ø CLS:PRINT@1,"#";N$:PRINT@2Ø,
D$:PRINT@67,RO$:PRINT@131,"AMT.
OF ACCOUNT  ";:PRINTUSING"$$###.
##";TA:PRINT@2Ø1,"AMT. PAID  ";:
PRINTUSING"$$###.##";DO:PRINT@26
3,"BALANCE DUE  ";:PRINTUSING"$$
###.##";BD:PRINT@323,"COMMENTS:
";C$
54Ø PRINT@458,"[M] FOR MENU"
55Ø I$=INKEY$:IFI$="M"GOTO5Ø
56Ø GOTO55Ø
57Ø CLOSE#1:RETURN
58Ø OPEN"D",#1,N$,11Ø
59Ø FIELD#1, 1Ø AS XTA$,1Ø AS XD
$,1Ø AS XDO$,1Ø AS XBD$,3Ø AS XR
O$,3Ø AS XC$,1Ø AS XDT$
6ØØ LSET XTA$=TA$
61Ø LSET XD$=DY$
62Ø LSET XDO$=DO$
63Ø LSET XBD$=BD$
64Ø LSET XRO$=RO$
65Ø LSET XC$=C$
66Ø LSET XDT$=D$
67Ø TA$=XTA$
68Ø D$=XDT$
69Ø DO$=XDO$
7ØØ BD$=XBD$
71Ø RO$=XRO$
72Ø C$=XC$:RETURN
73Ø RETURN
74Ø PUT#1:RETURN
75Ø GET#1:RETURN
76Ø 'PRINTER ROUTINE
77Ø CLS:PRINT@133,"DO YOU WANT A
 COPY FOR":PRINT@165,"FOR YOUR R
ECORDS ALSO?":PRINT@2Ø5,"(Y/N)"
78Ø PC$=INKEY$:IFPC$=""GOTO78Ø
79Ø IF PC$="Y" THEN PC=2
8ØØ IFPC$="Y"ORPC$="N"GOTO82Ø
81Ø GOTO78Ø
82Ø CLS:PRINT@236,"PRINTING"
83Ø POKE149,Ø:POKE15Ø,41:'12ØØBA
UD
84Ø FORPP=1TOPC
85Ø U$=CHR$(15):'UNDERLINING ON
86Ø DU$=CHR$(14):'LINING OFF
87Ø E$=CHR$(31):'DOUBLEWIDTH/ON
88Ø DE$=CHR$(3Ø):'D.W./OFF
89Ø NO$="NO.":DT$="Date ":RC$="R
eceived of":DL$="DOLLARS"
9ØØ AC$="AMT. OF ACCOUNT  "
91Ø AP$="      AMT. PAID  "
92Ø BD$="    BALANCE DUE  "
93Ø LF$=STRING$(2,1Ø):S$=CHR$(32
)
94Ø PRINT#-2,U$;STRING$(8Ø,32);S
TRING$(2,1Ø);DU$
95Ø PRINT#-2,E$;NO$;DE$;U$;S$;N$
;S$;DU$;STRING$(46,32);E$;DT$;DE
$;U$;S$;D$;S$;DU$;LF$
96Ø LR=LEN(RO$):LS=(47-LR)
97Ø PRINT#-2,E$;RC$;DE$;U$;S$;S$
;S$;S$;S$;RO$;STRING$(LS,32);DU$
;LF$
98Ø PRINT#-2,U$;STRING$(3Ø,32);E
$;:PRINT#-2,USING"$$###.##";DO;:
PRINT#-2,DE$;STRING$(11,42);DU$;
E$;DL$;DE$
99Ø PRINT#-2,U$;STRING$(25,32);D
U$;STRING$(28,32);"1ØØ"
1ØØØ PRINT#-2,AC$;:PRINT#-2,USIN
G"####.##";TA
1Ø1Ø C=LEN(C$):CL=(32-C)
1Ø2Ø PRINT#-2,AP$;U$;:PRINT#-2,U
SING"####.##";DO;:PRINT#-2,DU$;S
TRING$(15,32);"CMTS  ";U$;C$;STR
ING$(CL,32);DU$
1Ø3Ø PRINT#-2,U$;BD$;:PRINT#-2,U
SING"####.##";BD;:PRINT#-2,CHR$(
32)
1Ø4Ø PRINT#-2,CHR$(1Ø);DU$;STRIN
G$(43,32);E$;"$";DE$;U$;STRING$(
32,32)
1Ø5Ø PRINT#-2,U$;STRING$(8Ø,32);
DU$
1Ø6Ø PRINT#-2,STRING$(1Ø,1Ø)
1Ø62 IFPC=2GOTO1Ø65
1Ø63 GOTO1Ø7Ø
1Ø65 IFPP=1THENPRINT#-2,STRING$(
38,1Ø)
1Ø7Ø NEXTPP
1Ø8Ø GOTO5Ø
1Ø9Ø CLS:END
11ØØ CLS:'DIRECTORY ROUTINE
111Ø CLEAR
112Ø GOSUB57Ø
113Ø FORZ=3TO11
114Ø DSKI$Ø,17,Z,A$,B$
115Ø X$=A$:GOSUB119Ø
116Ø X$=B$:GOSUB119Ø
117Ø NEXTZ
118Ø GOTO128Ø
119Ø FORJ=1TO128STEP32
12ØØ R=R+1
121Ø P$(R,1)=MID$(X$,J,8)
122Ø IFLEFT$(P$(R,1),1)=CHR$(255
)THENR=R-1:GOTO128Ø
123Ø IFLEFT$(P$(R,1),1)=CHR$(Ø)T
HENR=R-1:GOTO126Ø
124Ø P$(R,2)=MID$(X$,J+8,3)
125Ø IFP$(R,2)<>"DAT"THENR=R-1
126Ø NEXTJ
127Ø RETURN
128Ø FORK=1TOR
129Ø PRINTUSING"##";K;:PRINT"] #
";P$(K,1),
13ØØ IFK=R THENPRINT,
131Ø NEXTK
132Ø LCN=48Ø
133Ø PRINT@LCN,"LOAD WHICH FILE
(M FOR MENU)";:INPUTP1$
134Ø IFP1$="M"GOTO14Ø
135Ø P=VAL(P1$)
136Ø IFP<1ORP>R THENLCN=448:GOTO
133Ø
137Ø FILE$=P$(P,1)
138Ø N$=FILE$
139Ø GOTO49Ø
```

Financing: The Economic Advantage

By Bill Bernico

Having been a car salesman for six years and a car rental manager for another four years, I've learned a lot about human nature and how people spend their money. Folks purchasing a new car might think that by taking the money out of their savings account to pay cash for their purchase that they are saving all that interest on the loan. Surprise! They've actually lost money doing it that way. This program will show in cold, hard figures which is the more economical move. It will also give skeptics a printout to take home and think about if they are not initially convinced after running the program.

When using *Cash vs. Financing*, input the same dollar amounts and the same number of months in each case. In other words, compare "apples with apples." (Or should I say "CoCos with CoCos.") If you're considering a $10,000 car and you have that amount in the bank, use $10,000 in the financing section of the program in order to get an honest comparison.

Naturally, the input for interest will vary between savings and financing, but use the same number of months in the comparison. Let's go through a sample session. For savings amount, input $10000 (no commas). For savings interest, let's use 8.75 (no percent sign necessary) and for months, input 48. Compound periods in this sample will be 2. The program will show you that by the end of the 48 month period, your nest egg will be worth $14085.49 or a gain of $4085.49 in interest.

This will put you into the finance part of the program. Your first input will be the finance amount. Again, use $10000. For finance interest, use 13.5 (even at this higher rate, you'll be surprised by the outcome). For the number of finance months, again use 48 and it will show you that your monthly payment will be $270.76 or a total of $12996.48 over 48 months. The interest you will have paid in those 48 months is $2996.48.

Comparing this figure with the $4085.49 in interest you would have earned from the savings account, you can see that you've saved $1089.01 by financing and leaving your nest egg alone. If you can find finance rates lower than 13.5, then the savings will be that much more.

At this point in the program, if the customer has still not been convinced to finance the car, the salesman can select from three options. Option 1 is a printout of the comparison. Option 2 is to start over with new figures. Option 3 is to end the program.

Option 1, the printout, will ask for the customer's name as well as the salesman's name. After the salesman inputs his name, the program will send the information to the printer. (This printout was tailored for the TP-10 printer which we have in our showroom, but will work with other printers as well.) The printout will personalize the hardcopy that the customer gets. It also has the dealership name and address as well as the salesman who serviced him. It's nice to stay fresh in the customer's mind.

Here is a sample printout from the program we just ran. This program need not be restricted to automobile purchases. It can be used on any item you like for comparison purposes.

One last note: I have fictionalized the name of the dealership in the printout as per my employer's request, and my apologies if there is really someone out there by the name of "Joe Average."

Sample Printout

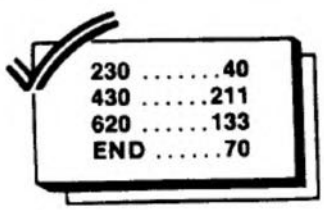
```
        HOMETOWN DODGE
    1234 NORTH 56TH STREET
     SHEBOYGAN, WI   53081
        (414) 555-4861
ASK FOR BILL BERNICO

SAVINGS AMOUNT.... 10000
SAVINGS INTEREST.. 8.75 %
SAVINGS MONTHS.... 48
COMPOUND PERIODS.. 2
SAVINGS AMOUNT
AFTER 48 MONTHS... 14085.49
INTEREST GAINED... 4085.49
------------------------------

FINANCE AMOUNT.... 10000
FINANCE INT....... 13.5 %
FINANCE MONTHS.... 48
MONTHLY PMT....... 270.76
TOTAL OF PMTS..... 12996.48
TOTAL INTEREST.... 2996.48

JOE AVERAGE CAN SAVE $ 1089.01
BY FINANCING THIS VEHICLE

ASK US FOR ASSISTANCE IN
ARRANGING A LOAN FOR YOU.
```



The listing: CASH

```
10 '***         CASH         ***
20 '*** BY BILL BERNICO      ***
30 '*** 708 MICHIGAN AV.     ***
40 '***  SHEBOYGAN, WI       ***
50 '***       53081          ***
60 '***  (414)459-7350       ***
70 '***                      ***
80 CLS:B$=CHR$(128)
90 PRINT@43,"advantages";B$;"of
100 PRINT@107,"financing";B$;"vs
110 PRINT@171,"paying";B$;"cash
120 PRINT@299,"COURTESY OF"
```

continued on Page 24

Forecast your budget, so in the future you can . . .

Juggle Bills, Juggle Bills, Juggle All the Way!

By Glen Dufur

This program, *Home Budget Analysis*, is used to assist in budgeting and forecasting personal finances in order to plan and adjust cash flow for three future periods (paydays). The program allows you to enter and update income and expense items for each of the three periods. The total of income and expenses is calculated and displayed for instant analysis of cash flow, as items are added or updated. The balance is money that has not been committed, or over-committed, if a negative balance is calculated.

As a period passes, you may shift all amounts so the second period becomes current and a new third period is opened. This allows you to continue budgeting for future periods. Also the ability is given to save or load a file of personal finance data.

Create New File/Load Existing File

You are given two options upon running the program. Press 'C' to create a new file or 'L' to load an existing file.

- Create New File: You are prompted to enter the dates of the three future periods to be budgeted. Enter each period date in the format "MM/DD."
- Load Existing File: You are prompted to ready the cassette. Press any key when ready to load the file.

Upon completion of entering the dates for creation of a new file or loading an existing file, the INCOME DISPLAY appears. You are now ready to begin entry or update of your personal finance file.

Add/Update/Delete Expense and Income Items

Income items are accessed via the income display, expense items via the expense display. Press 'N' to add a new item, or "A-H" to update or delete an existing item. Be sure to include all known income and expenses that occur during each period, for example, groceries, car expenses, rent, utilities, installment payments, savings, wages and other income.

- Add Item: A prompt is given to enter the DESCRIPTION. A description is required for each item entered. You are then prompted to enter an AUTO AMOUNT. This amount is automatically entered for each period of the item being added. Press ENTER without an amount if you do not wish an auto amount. This function is handy when the amount is the same for each period. You are now ready to enter amounts for the added item (see Update).
- Update Item: When updating an item, the display prompts for the period to update, '1', '2' or '3'. Enter the proper period and a prompt appears to enter the AMOUNT. To change the description and/or auto amount, press 'D'. A prompt appears to enter the new description. You are then prompted to enter the auto amount.
- Delete Item: After selecting the item to be deleted, press '*'. The item is deleted and you are automatically returned to the income or expense display.

Press 'R' to return to the income or expense display.

Expense and Income Displays

The income and expense displays list the items that represent the total of expenses and income. Options are available to move between the expense and income displays, in addition to adding or updating items. Press 'I' while in the expense display to call the income display. The expense display may be recalled by pressing 'X'.

Scrolling of items is accomplished by the up/down arrows while in either display. The last entry on the screen appears at the top when scrolled down and vice versa when scrolled up.

The calculated total of expenses and income is shown for each period with the balance of uncommitted or over-committed amounts.

Open New Period

As a period has past, you may delete the current period and move all income and expenses forward and, therefore, open a new period.

Press 'O' while in the expense or income display. The function prompts you to enter the date of the new period in the form "MM/DD." The program automatically shifts all amounts and drops the values for the current period. If you had entered an auto amount for any income or expense item, this amount is automatically entered into the new period for the item.

Technical Information

A maximum of 25 expense items and five income items have been imposed. Should your budget require more items, change the value of EN (expense items) and IN (income items) in Line 3000.

Logic Flow

Frequent Subroutines

10	INKEY
12	prompt alarm
15	blank two lines
20	top line
25	bottom line
30	screen load expenses
45	screen load income
60	calculate and print totals

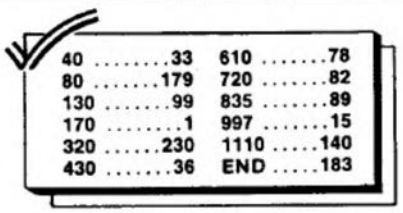
Other Subroutines

100	accept desc/default
120	delete expense
130	delete income

140	accept amount
150	basic screen
170	item screen
185	item bottom query
190	file full
300	expense display
400	expense item update
500	income display
600	income item update
700	shift period
800	LOAD and SAVE
900	new file
1000	initialize and start
1100	menu
2000	PCLEAR

Variables

ED$(EN)	expense description
EA(EN,3)	expense amount
DE(EN)	expense default
LE	last record expense
ID$(IN)	income description
IA(IN,3)	income amount
DA(IN)	income default
LI	last record income
PD$(3)	period dates
TE(3)	period total expense
TI(3)	period total income
PB(3)	period balance
EN	maximum expense records (preset to 25)
IN	maximum income records (preset to 5)
IE$	flag E=expense, I=income
I(8)	item addressability
IX$(8)	item addressability
I$(8)	item addressability



The listing: HOMEBDGT

```
1 GOTO2000
7 '
8 'FREQUENT SUBROUTINES
9 '
10 K$=INKEY$:IFK$=""THENGOTO10ELSEK=VAL(K$):RETURN
12 FORSS=1TO2:SOUND220,1:NEXTSS:RETURN
13 FORSS=1TO9:SOUND220,1:NEXTSS:RETURN
15 PRINTTAB(1)STRING$(30," "):PRINTTAB(1)STRING$(30," ");:RETURN
20 PRINTTAB(1)STRING$(30,CHR$(140));:RETURN
25 PRINTTAB(1)STRING$(30,CHR$(131));:RETURN
30 IFLE=0 THENPRINT@163,"enter"BB$;"expense"BB$"items";:SOUND200,1:RETURN
31 PP=64:PX=PP:IX=1:FORX=I(1) TO (I(1)+6):PRINT@PP+0,STRING$(32,CHR$(143));
35 IFX<(EN+1) THENPX=PP:PRINT@PP+0,IX$(IX);:PX=PX+1:PRINT@PX,USING"%     %";ED$(X);:PX=PX+7:FORY=1TO3:IFEA(X,Y)<>0 THENPRINT@PX,USING N2$;EA(X,Y);:PX=PX+8:NEXTY: ELSEPX=PX+8:NEXTY
40 I(IX)=X:IX=IX+1:PP=PP+32:NEXTX:SOUND200,1:RETURN
45 IFLI=0 THENPRINT@163,"enter"BB$;"income"BB$"items";:SOUND200,1:RETURN
46 PP=64:PX=PP:IX=1:FORX=I(1) TO (I(1)+6):PRINT@PP+0,STRING$(32,CHR$(143));
50 IFX<(IN+1) THENPX=PP:PRINT@PP+0,IX$(IX);:PX=PX+1:PRINT@PX,USING"%     %";ID$(X);:PX=PX+7:FORY=1TO3:IFIA(X,Y)<>0 THENPRINT@PX,USING N2$;IA(X,Y);:PX=PX+8:NEXTY: ELSEPX=PX+8:NEXTY
55 I(IX)=X:IX=IX+1:PP=PP+32:NEXTX:SOUND200,1:RETURN
60 FORX=1TO3:TE(X)=0:FORY=1TOLE:TE(X)=TE(X)+EA(Y,X):NEXTY,X:FORX=1TO3:TI(X)=0:FORY=1TOLI:TI(X)=TI(X)+IA(Y,X):NEXTY,X:FORX=1TO3:PB(X)=TI(X)-TE(X):NEXTX
65 PRINT@288,CHR$(140);:GOSUB20:PRINT@320,"expense";:PP=320:FORX=1TO3:PP=PP+8:PRINT@PP,USINGN2$;TE(X);:NEXTX
70 PRINT@352,"income";BB$;:PP=352:FORX=1TO3:PP=PP+8:PRINT@PP,USINGN2$;TI(X);:NEXTX
75 PRINT@384,"balance";:PP=384:FORX=1TO3:PP=PP+8:PRINT@PP,USINGN2$;PB(X);:NEXTX
80 PRINT@416,CHR$(131);:GOSUB25:RETURN
97 '
98 'OTHER SUBROUTINES
99 '
100 PRINT@448,"";:GOSUB15:PRINT@449,"";:GOSUB12:LINEINPUT"DESC:";XX$:IFXX$="" THENSOUND1,1:GOTO100ELSEPRINT@132,"*** "XX$" ***";
102 PRINT@448,"";:GOSUB15:PRINT@449,"AUTO AMOUNT";:GOSUB12:INPUTXX
105 IFIE$="E" THENED$(I)=XX$:DE(I)=XX:IFEA(I,1)=0 ANDEA(I,2)=0 ANDEA(I,3)=0 THENFORX=1TO3:EA(I,X)=DE(I):NEXTX:RETURN:ELSERETURN
110 ID$(I)=XX$:DI(I)=XX:IFIA(I,1)=0 ANDIA(I,2)=0 ANDIA(I,3)=0 THENFORX=1TO3:IA(I,X)=DI(I):NEXTX:RETURN:ELSERETURN
120 IFI=LE THENED$(I)="":DE(I)=0:FORZ=1TO3:EA(I,Z)=0:NEXTZ:LE=LE-1: ELSEFORX=I TOLE-1:IFED$(X+1)<>"" THENED$(X)=ED$(X+1):ED$(X+1)="":DE(X)=DE(X+1):DE(X+1)=0:FORZ=1TO3:EA(X,Z)=EA(X+1,Z):EA(X+1,Z)=0:NEXTZ:NEXTX:LE=LE-1: ELSENEXTX
125 RETURN
130 IFI=LI THENID$(I)="":DI(I)=0:FORZ=1 TO3:IA(I,Z)=0:NEXTZ:LI=LI-1: ELSEFORX=I TOLI-1:IFID$(X+1)<>"" THENID$(X)=ID$(X+1):ID$(X+1)="":DI(X)=DI(X+1):DI(X+1)=0:FORZ=1TO3:IA(X,Z)=IA(X+1,Z):IA(X+1,Z)=0:NEXTZ:NEXTX:LI=LI-1: ELSENEXTX
135 RETURN
140 PRINT@448,"";:GOSUB15:PRINT@449,"PERIOD"K"AMOUNT";:GOSUB12:INPUTXX:IFIE$="E" THENEA(I,K)=XX ELSEIA(I,K)=XX
145 RETURN
150 CLS:PRINT"";:IFIE$="E" THENPRINTSTRING$(32,CHR$(242)); ELSEPRINTSTRING$(32,CHR$(162));
155 PRINT@16,"display";BB$;:PRINT@32,BB$;"period>";:PRINT@42,USINGN3$;PD$(1);:PRINT@50,USINGN3$;PD$(2);:PRINT@58,USINGN3$;PD$(3);:IFIE$="E" THEN160 ELSE165
160 PRINT@7,BB$;"expense";BB$;:PRINT@449,"a-h  nEW EXPENSE  iNCOME  sAVE";:PRINT@481,"  <ARROWS>  oPEN NEW PERIOD";:RETURN
165 PRINT@8,BB$;"income";BB$;:PRINT@449,"a-h  nEW INCOME  ExPENSE  sAVE";:PRINT@481,"  <ARROWS>  oPEN NEW PERIOD";:RETURN
170 CLS:PRINT@193,"     PERIOD     AMOUNT  ";:PRINT@227,STRING$(10,CHR$(131));:PRINT@242,STRING$(8,CHR$(131));:PRINT@385,"AUTO AMOUNT:";:PRINT@416,"";:GOSUB25
175 PRINT@32,"";:GOSUB20:PRINT:IFIE$="E" THENPRINTTAB(5)"EXPENSE ITEM UPDATE":ELSEPRINTTAB(5)"INCOME ITEM UPDATE"
180 GOSUB25:RETURN
185 PRINT@449,"";:GOSUB15:PRINT@449,"   UPDATE PERIOD <1> <2> <3>    dESC *DELETE  nEW ITEM  rETURN";:RETURN
190 PRINT@449,"";:GOSUB15:PRINT@449," file full - press enter";:GOSUB10:RETURN
297 '
298 'EXPENSE ITEM UPDATE
299 '
300 IE$="E":I(1)=1
305 PP=32:GOSUB150:GOSUB60:GOSUB30
310 I=0:GOSUB10
315 IFK$=CHR$(10) THENI(1)=I(1)+6:IFI(1)>LE THENI(1)=LE:GOSUB30:GOTO310:ELSEGOSUB30:GOTO310
320 IFK$="^" THENI(1)=I(1)-6:IFI(1)<1THEN I(1)=1:GOSUB30:GOTO310 ELSEGOSUB30:GOTO310
325 IFK$="N" THENGOSUB400:GOTO305:ELSEIFK$="I" THEN500 ELSEIFK$="S" THENIO=2:GOSUB800:GOTO300 ELSEIFK$="O" THENGOSUB700:GOTO300
330 FORX=1TO8:IFK$=I$(X) THENI=I(X):NEXTX:ELSENEXTX
335 IFI=0 THEN340 ELSEIFI>ENTRIES THEN340 ELSEIFED$(I)="" THEN340 ELSEGOSUB400:GOTO305
340 SOUND1,1:GOTO310
397 '
398 'EXPENSE ITEM UPDATE
399 '
400 SOUND200,1
405 IFK$="N" THENLE=LE+1:I=LE:IFLE>EN THENLE=EN:GOSUB190:RETURN
410 GOSUB170:PRINT@132,"*** "ED$(I)" ***";:PP=259:FORX=1TO3:PRINT@PP,X;:PRINT@PP+2,">";:PRINT@PP+4,USINGN3$;PD$(X);:PRINT@PP+15,USING N2$;EA(I,X);:PP=PP+32:NEXTX:PRINT@402,USINGN2$;DE(I);
415 IFK$="N" THENGOSUB100:K$="":GOTO410: ELSEGOSUB185
420 GOSUB10:IFK$="R" THENRETURN ELSEIFK$="*" THENGOSUB120:RETURN:ELSEIFK$="D" THENGOSUB100:GOTO400ELSEIFK$="N" THENGOTO400
425 IF K>0 AND K<4 THENGOSUB140:GOTO410
430 SOUND1,1:GOTO420
497 '
498 'INCOME DISPLAY
499 '
500 IE$="I":I(1)=1
505 PP=32:GOSUB150:GOSUB60:GOSUB45
510 I=0:GOSUB10
515 IFK$=CHR$(10) THENI(1)=I(1)+6:IFI(1)>LI THENI(1)=LI:GOSUB45:GOTO510:ELSEGOSUB45:GOTO510
520 IFK$="^" THENI(1)=I(1)-6:IFI(1)<1THEN I(1)=1:GOSUB45:GOTO510ELSEGOSUB45:GOTO510
525 IFK$="N" THENGOSUB600:GOTO500:ELSEIFK$="X" THEN300 ELSEIFK$="S" THENIO=2:GOSUB800:GOTO300 ELSEIFK$="O" THENGOSUB700:GOTO500
530 FORX=1TO8:IFK$=I$(X) THENI=I(X):NEXTX:ELSENEXTX
535 IFI=0 THEN540 ELSEIFI>9 THEN540 ELSEIFID$(I)="" THEN540 ELSEGOSUB600:GOTO505
540 SOUND1,1:GOTO510
597 '
598 'INCOME ITEM UPDATE
```

```
599 '
6ØØ SOUND2ØØ,1
6Ø5 IFK$="N" THENLI=LI+1:I=LI:IF
LI>IN THENLI=IN:GOSUB19Ø:RETURN
61Ø GOSUB17Ø:PRINT@132,"*** "ID$
(I)" ***";:PP=259:FORX=1TO3:PRIN
T@PP,X;:PRINT@PP+2,">";:PRINT@PP
+5,USINGN3$;PD$(X);:PRINT@PP+15,
USING N2$;IA(I,X);:PP=PP+32:NEXT
X:PRINT@4Ø2,USINGN2$;DI(I);
615 IFK$="N" THENGOSUB1ØØ:K$="":
GOTO61Ø: ELSEGOSUB185
62Ø GOSUB1Ø:IFK$="R" THENRETURN
ELSEIFK$="*" THENGOSUB13Ø:RETURN
:ELSEIFK$="D" THENGOSUB1ØØ:GOTO6
ØØELSEIFK$="N" THENGOTO6ØØ
625 IF K>Ø AND K<4 THENGOSUB14Ø:
GOTO61Ø
63Ø SOUND1,1:GOTO62Ø
697 '
698 'OPEN NEW PERIOD
699 '
7ØØ CLS:SOUND2ØØ,1
7Ø5 PRINT@129,STRING$(3Ø,CHR$(14
Ø));:PRINTTAB(3)"SHIFT AND OPEN N
EW PERIOD":PRINT TAB(1)STRING$(3
Ø,CHR$(131))
71Ø GOSUB12:PRINT@292,"HIT ANY K
EY TO CONTINUE                OR r
ETURN";:GOSUB1Ø:IF K$="R" THENRE
TURN
715 PRINT@288,"":GOSUB15,:PRINT@
289,"ENTER NEW PERIOD DATE (MM/D
D)";:PRINT@335,"";:GOSUB12:LINEI
NPUT"";XX$:PRINT@288,"";:GOSUB15
:PRINT@294,"NEW PERIOD FOR ";XX$
:PRINTTAB(6)" NOW BEING OPENED";
72Ø FORX=1TOLE:EA(X,1)=EA(X,2):E
A(X,2)=EA(X,3):EA(X,3)=DE(X):NEX
TX:FORX=1TOLI:IA(X,1)=IA(X,2):IA
(X,2)=IA(X,3):IA(X,3)=DI(X):NEXT
X:PD$(1)=PD$(2):PD$(2)=PD$(3):PD
$(3)=XX$:RETURN
797 '
798 'I/O ROUTINES
799 '
8ØØ CLS:SOUND2ØØ,1
8Ø5 IFIO=1 THENIO$="LOAD" ELSEIO
$="SAVE"
81Ø PRINT@64,"";:GOSUB2Ø:PRINT@1
Ø5,IO$;" FILE";:PRINT@128,"";:GO
SUB25
815 GOSUB13:PRINT@193,"POSITION
TAPE...":PRINT" READY CASSETTE..
.":PRINT" PRESS ANY KEY TO CONTI
NUE":GOSUB1Ø:PRINT@192,"":GOSUB1
5:GOSUB15:IFIO=2 THEN845
82Ø '*---LOAD FILE---*
825 SOUND22Ø,1:PRINT@193," LOADI
NG BUDGET FILE";
83Ø OPEN"I",#-1,"BUDGET":SOUND22
Ø,1
835 INPUT#-1,LE,LI:FORX=1TOLE:IN
PUT#-1,ED$(X),DE(X):FORY=1TO3:IN
PUT#-1,EA(X,Y):NEXTY,X:FORX=1TOL
I:INPUT#-1,ID$(X),DI(X):FORY=1TO
3:INPUT#-1,IA(X,Y):NEXTY,X:FORX=
1TO3:INPUT#-1,PD$(X):NEXTX
84Ø CLOSE#-1:RETURN
845 SOUND22Ø,1
85Ø PRINT@193," SAVING BUDGET FI
LE";:MOTORON:FORX=1TO6ØØ:NEXTX
855 OPEN "O",#-1,"BUDGET":SOUND2
2Ø,1
86Ø PRINT#-1,LE,LI:FORX=1TOLE:PR
INT#-1,ED$(X),DE(X):FORY=1TO3:PR
INT#-1,EA(X,Y):NEXTY,X:FORX=1TOL
I:PRINT#-1,ID$(X),DI(X):FORY=1TO
3:PRINT#-1,IA(X,Y):NEXTY,X:FORX=
1TO3:PRINT#-1,PD$(X):NEXTX
865 CLOSE#-1:RETURN
897 '
898 'CREATE NEW FILE
899 '
9ØØ CLS:SOUND2ØØ,1
9Ø5 PRINT@33,"":GOSUB2Ø:PRINT:PR
INTTAB(6)"ENTER PERIOD DATES":GO
SUB25
91Ø PP=36Ø:FORX=1TO3:PRINT@214,"
";:GOSUB15:PRINT@161,"PERIOD #";
X;:PRINT@227,"";:GOSUB12:LINEINP
UT"ENTER DATE (MM/DD): ";PD$(X):
PRINT@PP,"PERIOD ";X"--> ";:PRIN
TUSINGN3$;PD$(X):PP=PP+32:NEXTX:
GOTO5ØØ
997 '
998 'INITIALIZE PROGRAM
999 '
1ØØØ CLEAR5ØØ:SOUND2ØØ,1:EN=25:I
N=5:DIMED$(EN):DIMEA(EN,3):DIMPD
$(3):DIMIA(IN,3):DIMI$(8):DIMI(8
):DIMDE(EN):DIMID$(IN):DIMDI(IN)
:DIMIX$(8):DIMTE(3):DIMTI(3):DIM
PB(3)
1ØØ5 LE=Ø:LI=Ø:FORX=1TO8:READIX$
(X),I$(X),I(X):NEXTX
1Ø1Ø DATA a,A,1,b,B,2,c,C,3,d,D,
4,e,E,5,f,F,6,g,G,7,h,H,8
1Ø15 NØ$="##":N2$="####.##-":N1$
="+####.##":LD$="%  %":N3$="%
%":BB$=CHR$(128)
1Ø3Ø '
11ØØ SOUND2ØØ,1
11Ø5 CLS:PRINT@99,"home";BB$;"bu
dget";BB$;"analysis":PRINT:PRINT
"    BY: GLEN DUFUR":PRINT"    C
OPYRIGHT (C) 1985"
111Ø PRINT:PRINTTAB(4)"lOAD EXIS
TING BUDGET FILE":PRINT:PRINTTAB
(4)"cREATE NEW BUDGET FILE":PRIN
T@424,"SELECT OPTION";
1115 IFXX$=CHR$(161) THENXX$=CHR
$(162):XY$=CHR$(164):XZ$=CHR$(16
6): ELSEXX$=CHR$(161):XY$=CHR$(1
68):XZ$=CHR$(169)
112Ø PRINT@65,CHR$(138);:PRINTST
RING$(22,XX$);CHR$(133);:PRINT@1
29,CHR$(138);STRING$(22,XY$);CHR
$(133);:K$=INKEY$:PRINT@97,CHR$(
138);XZ$;:PRINT@119,XZ$;CHR$(133
)
1125 PRINT@438," ";:IFK$="" THEN
1115 ELSEIFK$="L" THENIO=1:GOSUB
8ØØ:GOTO5ØØ: ELSEIFK$="C" THEN9Ø
Ø
113Ø PRINT@438,K$;:SOUND1,5:GOTO
1115
12ØØ 'CSAVE "BUDGET" ROUTINE
12Ø5 FORX=1TO2:MOTORON:FORY=1TO6
ØØ:NEXTY:MOTOROFF:CSAVE"BUDGET":
NEXTX:FORX=1TO5:SOUND2ØØ,1:NEXT:
END
1999 'PCLEAR ROUTINE
2ØØØ PCLEAR1:GOTO1ØØØ
```

continued from Page 21

```
13Ø PRINT@363,"HOMETOWN DODGE
14Ø PRINT@484,"(HIT ANY KEY TO C
ONTINUE)";
15Ø EXEC 44539
16Ø A=Ø:I=Ø:T=Ø:Y=Ø:V=Ø:L=Ø:P=Ø:
X=Ø:N=Ø:K=Ø:O=Ø:U=Ø:E=Ø
17Ø CLS
18Ø INPUT"SAVINGS AMOUNT......";
A
19Ø INPUT"SAVINGS INTEREST....";
I:I=I/1ØØ
2ØØ INPUT"SAVINGS MONTHS......";
T:T=T/12
21Ø INPUT"COMPOUND PERIODS....";
Y
22Ø V=A*(1+I/Y)^(Y*T):V=INT(V*1Ø
Ø+.5)/1ØØ
23Ø PRINT"AFTER";T*12;"MONTHS...
..$";V
24Ø L=V-A
25Ø PRINT"INTEREST GAINED.....$
";L
26Ø PRINTSTRING$(32,"-");
27Ø INPUT"FINANCE AMOUNT......";
P
28Ø INPUT"FINANCE INTEREST....";
X:X=X/1ØØ
29Ø INPUT"FINANCE MONTHS......";
N
3ØØ K=P*(X/12)/(1-(1+(X/12))^-N)
31Ø K=INT(K*1ØØ+.5)/1ØØ
32Ø PRINT"MONTHLY PAYMENT.....$
";K
33Ø O=K*N
34Ø O=INT(O*1ØØ+.5)/1ØØ
35Ø PRINT"TOTAL OF PAYMENTS...$"
;O
36Ø U=INT((O-P)*1ØØ+.5)/1ØØ
37Ø PRINT"TOTAL INTEREST......$
";U
38Ø E=INT((L-U)*1ØØ+.5)/1ØØ
39Ø PRINT"AMOUNT SAVED
4ØØ PRINT"BY FINANCING........$
";E
41Ø PRINT@482,"pRINTOUT  sTART O
VER  eND";
42Ø A$=INKEY$:IF A$=""THEN 42Ø
43Ø IF A$="S"THEN 16Ø
44Ø IF A$="P"THEN 47Ø
45Ø IF A$="E"THEN 1Ø
46Ø GOTO 42Ø
47Ø CLS:INPUT"CUSTOMER'S NAME";C
N$
48Ø INPUT"SALESMAN'S NAME";SN$
49Ø CLS:PRINT@232,"....PRINTING.
...
5ØØ PRINT#-2,"         HOMETOWN D
ODGE
51Ø PRINT#-2,"     1234 NORTH 56T
H STREET
52Ø PRINT#-2,"      SHEBOYGAN, WI
   53Ø81
53Ø PRINT#-2,"          (414) 555
-4861
54Ø PRINT#-2,"ASK FOR ";SN$
55Ø PRINT#-2
56Ø PRINT#-2,"SAVINGS AMOUNT....
";A
57Ø I=I*1ØØ
58Ø PRINT#-2,"SAVINGS INTEREST..
";I;"%
59Ø T=T*12
6ØØ PRINT#-2,"SAVINGS MONTHS....
";T
61Ø PRINT#-2,"COMPOUND PERIODS..
";Y
62Ø PRINT#-2,"SAVINGS AMOUNT"
63Ø PRINT#-2,"AFTER";T;"MONTHS..
.";V
64Ø PRINT#-2,"INTEREST GAINED...
";L
65Ø PRINT#-2
66Ø PRINT#-2,"-----------------
-------------
67Ø PRINT#-2
68Ø PRINT#-2,"FINANCE AMOUNT....
";P
69Ø X=X*1ØØ
7ØØ PRINT#-2,"FINANCE INT.......
";X;"%
71Ø PRINT#-2,"FINANCE MONTHS....
";N
72Ø PRINT#-2,"MONTHLY PMT.......
";K
73Ø PRINT#-2,"TOTAL OF PMTS.....
";O
74Ø PRINT#-2,"TOTAL INTEREST....
";U
75Ø PRINT#-2
76Ø PRINT#-2,CN$;" CAN SAVE $";E
77Ø PRINT#-2,"BY FINANCING THIS
VEHICLE
78Ø PRINT#-2
79Ø PRINT#-2,"ASK US FOR ASSISTA
NCE IN
8ØØ PRINT#-2,"ARRANGING A LOAN F
OR YOU.
81Ø PRINT#-2:PRINT#-2:PRINT#-2:P
RINT#-2:PRINT#-2:PRINT#-2
82Ø CLS:PRINT@481,"aNOTHER PRINT
OUT  rESTART  eND";
83Ø A$=INKEY$:IFA$=""THEN83Ø
84Ø IF A$="A"THEN 5ØØ
85Ø IF A$="R"THEN 16Ø
86Ø IF A$="E"THEN 1Ø
87Ø GOTO 83Ø
```

The Joy$ of Early Amortization

By Edward R. Carson

Paying off a mortgage early to get a quicker equity buildup is the best idea I (a homeowner) have heard of in a long time. It used to cost a little extra each month to reach this goal, but a new kind of mortgage is just now taking hold in the United States that can make the process almost painless. Quicker mortgage payoff will save you a fortune in interest rates and can take years off the repayment schedule. It also has two strategic uses.

A young couple who pays off their mortgage early will then have a huge amount of equity on tap. This is also a method of forced savings; all of these gains are tax deferred. A quick payment mortgage is also suitable for middle-aged home buyers who want to own their home free and clear by the time they retire. The new way to faster home ownership is through a bi-weekly mortgage payment plan. The loan is amortized as if it were going to last for 30 years, but instead of paying once a month, one half the payment is made every two weeks. This method of repayment leads to the equivalent of 13 monthly payments rather than the usual 12. This may not sound like it would make a lot of difference, but the amount of money and time saved is astounding, as you will see when comparing Option 1 with Option 2. Since this method of repayment is not available with all lenders, two other options are included that can have the same effect and are accepted by most lenders.

Edward R. Carson is a head operator at the Timken Company in Columbus, Ohio. His interests encompass computers and baseball. He is married and has three sons.

There are two parts to the *Mortgage Planner.* The first part is a loan calculator. If you are planning a home purchase, the calculator figures your principal and interest payments. It returns the amount financed, amount of payment, interest rate and number of months required to retire the loan (see Figure 1). If you know the amount you want to pay per month but don't know the amount you can finance to arrive at your target payment, the calculator can help. When asked the amount to finance, just press ENTER; you are then asked the amount per month (enter what you want to pay per month) and the calculator gives the amount to finance and arrives at your target payment. You can go through as many calculations as you wish. The last amount calculated is automatically forwarded to the *Mortgage Planner.* It is not necessary to go to the calculator. If you have an existing mortgage, go directly to the Planner.

The *Mortgage Planner* has four options to choose from. Each is a different method of repayment. Three of these options can save thousands of dollars and many years off the mortgage. The

Figure 1

```
AMOUNT OF THE LOAN      $ 36000.00
NO. OF MONTHS             360
INTEREST RATE              10 %
MONTHLY PAYMENTS        $ 315.93
```

Option 1

```
YEAR      INT.PAID          PRINCIPAL PAID
 1      $  3590.17        $    20.94   /
 2      $  7159.30        $   422.92   /
 3      $ 10705.20        $   668.13
 4      $ 14225.42        $   939.02   /
 5      $ 17717.28        $  1238.26
 6      $ 21177.82        $  1568.84
 7      $ 24603.75        $  1934.02   /
 8      $ 27991.44        $  2337.43   /
 9      $ 31336.91        $  2783.07
10      $ 34635.72        $  3275.37   /
11      $ 37882.99        $  3819.21   /
12      $ 41073.33        $  4419.98   /
13      $ 44200.78        $  5083.64   /
14      $ 47258.75        $  5816.78   /
15      $ 50239.96        $  6626.68
16      $ 53136.39        $  7521.36   /
17      $ 55939.15        $  8509.70   /
18      $ 58638.45        $  9601.51
19      $ 61223.45        $ 10807.63   /
20      $ 63682.17        $ 12140.01
21      $ 66001.42        $ 13611.88
22      $ 68166.57        $ 15237.83   /
23      $ 70161.51        $ 17034.00   /
24      $ 71968.40        $ 19018.22   /
25      $ 73567.57        $ 21210.16
26      $ 74937.27        $ 23631.57
27      $ 76053.47        $ 26306.48   /
28      $ 76889.64        $ 29261.42   /
29      $ 77416.46        $ 32525.71   /

   YEARS          TOTAL INTEREST          TOTAL PRINCIPAL

   30.0           $ 77733.28              $ 36000.00
```

Option 2

```
YEAR      INT.PAID                PRINCIPAL PAID
 1      $  3564.58              $   542.46                /
 2      $  7ø72.38              $  1141.69                /
 3      $ 1ø517.47              $  18ø3.63                /
 4      $ 13893.28              $  2534.85
 5      $ 17192.57              $  3342.6ø                /
 6      $ 2ø4ø7.32              $  4234.89
 7      $ 23528.68              $  522ø.56                /
 8      $ 26546.89              $  63ø9.39                /
 9      $ 29451.14              $  7512.17
 1ø     $ 32229.51              $  884ø.84                /
 11     $ 34868.83              $ 1ø3ø8.55
 12     $ 37354.54              $ 11929.88                /
 13     $ 3967ø.57              $ 1372ø.89
 14     $ 41799.15              $ 15699.34                /
 15     $ 43728.67              $ 17884.85                /
 16     $ 45413.47              $ 2ø299.ø9                /
 17     $ 46853.6ø              $ 22966.øø                /
 18     $ 48ø14.61              $ 25912.ø2                /
 19     $ 48867.3ø              $ 29166.36                /
 2ø     $ 49379.4ø              $ 32761.3ø                /

YEARS       TOTAL INTEREST                TOTAL PRINCIPAL

22.6        $ 49615.88                    $ 36øøø.øø
```

choice of on-screen or printer displays is given. The printer routine gives a year-by-year printout of interest paid, principal, paid outstanding balance, total payments, and years and months required to retire the loan. There is an on-screen bar graph of interest paid at all options. Any calculated screen can also be dumped to the printer by pressing the 'P' key. I found this easier than writing down all the information on a scratch pad. The amount of money that can be saved with just a little extra each month literally amazes me, as I am sure it will you.

(Any questions you have about *Mortgage Planner* may be directed to Mr. Carson at 7600 Condit Road, Centerburg, OH 43011, phone 614-625-6936. Please include an SASE when writing.) □

```
111 .....218   1001 .....207
191 .....141   1061 .....5
301 .....62    1111 .....120
391 .....7     1151 .....105
496 .....209   1246 .....206
556 .....1     1341 .....22
666 .....139   1441 .....197
771 .....54    1481 .....136
821 .....231   1571 .....240
921 .....242   END .....83
```

The listing: MORTGAGE

```
1 Y=1
6 CLS
11 X=32
16 CLS
21 Z$="SAVE"
26 PRINT@X,Z$
31 X=X+1ø
36 IFX=382 THEN 41 ELSE 26
41 FORT=1TO8ø:NEXTT
46 Y=Y+1:IFY=5THEN51ELSE6
51 FORT=1TO5øø:NEXTT
56 CLS:PRINT@164,"the mortgage p
lanner"
61 PRINT@236,"by"
66 PRINT@294,"edward r carson"
71 GOSUB1181
76 GOTO1ø21
81 CLS
86 PRINT:PRINT:PRINT "YOU HAVE F
OUR OPTIONS WITH THIS PORTION OF
 THE PROGRAM..."
91 PRINT:PRINTTAB(7)"THEY ARE AS
 FOLLOWS.."
96 PRINT:PRINT:PRINT"HIT ANY KEY
 TO CONTINUE"
1ø1 K$=INKEY$:IFK$=""THEN 1ø1ELS
E1ø6
1ø6 CLS:PRINT"1) CONTINUE TO MAK
E NORMAL        MONTHLY PAYMENTS
..."
111 PRINT:PRINT"2) MAKE 1/2 OF N
ORMAL PAYMENT     EVERY 14 DAYS.
.."
116 PRINT:PRINT"3) LUMP SUM (IN
EXCESS OF NORMAL PAYMENT)  ONCE
EACH YEAR..."
121 PRINT:PRINT"4) INCREASE MONT
HLY PAYMENT BY    (X) AMOUNT...(X
) AMOUNT USED TO REDUCE BALANCE
ON A MONTHLY      BASIS"
126 PRINT"HIT ANY KEY TO CONTINU
E"
131 K$=INKEY$:IFK$=""THEN 131ELS
E 136
136 CLS:PRINT@75,"options"
141 PRINT:PRINTTAB(1ø)"1  2  3
4"
146 PRINT:PRINTTAB(1ø)"select on
e"
151 PRINTSTRING$(32,"*")
156 PRINT"1= NORMAL PAYMENT"
161 PRINT:PRINT"2= 1/2 NORMAL PA
YMENT"
166 PRINT:PRINT"3= LUMP SUM"
171 PRINT:PRINT"4= EXCESS MONTHL
Y"
176 INPUT S
181 ON S GOTO 231,546,821,841
186 CLS:PRINT@195, "what is your
 normal...."
191 PRINT@263,"monthly payment..
.."
196 INPUT NP
2ø1 CLS:PRINT@193,"what is your
interest rate...."
2ø6 PRINT@258,"input as per exam
ple <.ø95ø >"
211 INPUT AI
216 CLS:PRINT@192,"what is your
current balance.."
221 INPUT CB
226 GOTO 81
231 CLS:PRINT@26ø,"DO YOU WANT A
 PRINTOUT OF..."
236 PRINT@324,"YEARLY ANALYSIS..
."
241 PRINT:PRINTTAB(11)" ( Y/N )"
246 K$=INKEY$:IFK$=""THEN246
251 IFK$="Y"THEN 421ELSEIFK$="N"
THEN256
256 CLS:PRINT@196,"calculating t
otals..."
261 PRINT@26ø,"please stand by..
."
266 POKE 65495,ø
271 DP=3ø.41:Z=1
276 DI=AI/365
281 IN=DI*CB*DP
286 P=NP-IN
291 PB=PB+P+EP
296 CB=CB-P
3ø1 TP=TP+NP+EP
3ø6 TI(S)=TI(S)+IN
311 CB=CB-EP
316 IF CB<=ØTHEN326ELSE321
321 IFZ=M THEN831ELSEZ=Z+1:GOTO2
81
326 TI(S)=TI(S)-CB:PB=PB+CB
331 CLS:PRINTTAB(12)"OPTION ";S
336 PRINT:PRINT"INTEREST PAID";:
PRINTTAB(21);:PRINTUSING"$######
.##";TI(S)
341 PRINT"PRINCIPAL PAID";:PRINT
TAB(21);:PRINTUSING"$######.##";
PB
346 PRINTTAB(21)STRING$(1ø,"-")
351 PRINT"TOTAL PAID";:PRINTTAB(
21);:PRINTUSING"$######.##";TP
356 Z=Z/12
361 PRINT:PRINT"YEARS TO RETIRE
LOAN.. ";:PRINTUSING"##.#";Z
366 PRINT"NORMAL PAYMENT........
 ";:PRINTUSING"$####.##";NP
371 PRINT"INTEREST RATE.........
 ";AI;"%"
376 IFS=3GOSUB1426ELSEIFS=4GOSUB
1431
381 POKE 65494,ø
386 C$=INKEY$:IFC$=""THEN386
391 IFC$="P"THENGOSUB 1381ELSE 3
96
396 IF S=4 THEN GOSUB 1246:GOTO
4ø1
4ø1 CB=PB:PB=ø:Z=ø:TB=ø:TP=ø:LS=
ø:EP=ø
4ø6 PRINT:PRINT"DO YOU WANT THIS
 OPTION  AGAIN"
411 PRINTTAB(12)"( Y/N )":INPUT
C$:IFC$="N"THEN 136 ELSE 416
416 TI(S)=ø:GOTO 136
421 CLS:PRINT@264,"now printing"
426 PRINT#-2,TAB(3ø)"OPTION ";S
431 IFS=3GOSUB851
436 IFS=4GOSUB856
441 PRINT#-2,"YEAR";TAB(6)"INT.P
AID";TAB(18)"PRINCIPAL PAID";TAB
(35)"TOTAL PAYMENT";TAB(51)"OUTS
TANDING BALANCE"
446 DP=3ø.41:Z=1:H=12:Y=1
451 DI=AI/365
456 IN=DI*CB*DP
461 P=NP-IN
466 PB=PB+P+EP
471 CB=CB-P
476 TP=TP+NP+EP
481 TI(S)=TI(S)+IN
486 CB=CB-EP
491 IFCB<=ØTHEN511ELSE496
496 IFZ=H THEN836ELSEZ=Z+1:GOTO4
56
5ø1 IFCB<ØTHENCB=ø
5ø6 PRINT#-2,Y;:PRINT#-2,TAB(6);
:PRINT#-2,USING"$######.##";TI(S
);:PRINT#-2,TAB(18);:PRINT#-2,US
ING"$######.##";PB;:PRINT#-2,TAB
(35);:PRINT#-2,USING"$######.##"
;TP;:PRINT#-2,TAB(51);:PRINT#-2,
USING"$######.##";CB:Y=Y+1:H=H+1
2:GOTO456
511 TI(S)=TI(S)-CB:PB=PB+CB
```

```
516 PRINT#-2,""
521 PRINT#-2,TAB(5)"YEARS";TAB(1
2)"TOTAL INTEREST";TAB(28)"TOTAL
 PRINCIPAL";TAB(45)"TOTAL PAYMEN
TS"
526 PRINT#-2,""
531 Z=Z/12
536 PRINT#-2,TAB(5):PRINT#-2,USI
NG"##.#";Z;:PRINT#-2,TAB(12);:PR
INT#-2,USING"$######.##";TI(S);:
PRINT#-2,TAB(28);:PRINT#-2,USING
"$######.##";PB;:PRINT#-2,TAB(45
);:PRINT#-2,USING"$######.##";TP
541 CB=PB:TP=Ø:TI=Ø:PB=Ø:EP=Ø:LS
=Ø:GOTO 136
546 CLS:PRINT@26Ø,"DO YOU WANT A
 PRINTOUT OF..."
551 PRINT@324,"YEARLY ANALYSIS..
."
556 PRINT:PRINTTAB(11)" ( Y/N )"
561 K$=INKEY$:IF K$=""THEN561
566 IFK$="Y"THEN716ELSEIFK$="N"T
HEN581
571 ''
576 ''
581 CLS:NP=NP/2:DP=14:Z=1:H=26:Y
=1
586 POKE65495,Ø
591 PRINT@196,"calculating total
s..."
596 PRINT@26Ø,"please stand by..
."
6Ø1 DI=AI/365
6Ø6 IN=DI*CB*DP
611 P=NP-IN
616 PB=PB+P
621 CB=CB-P
626 TP=TP+NP
631 TI(S)=TI(S)+IN
636 IFCB<=ØTHEN646ELSE641
641 Z=Z+1:GOTO6Ø6
646 TI(S)=TI(S)-CB:PB=PB+CB
651 CLS:PRINTTAB(11)"OPTION ";S
656 PRINT:PRINT"INTEREST PAID";:
PRINTTAB(21);:PRINTUSING"$######
.##";TI(S)
661 PRINT"PRINCIPAL PAID";:PRINT
TAB(21);:PRINTUSING"$######.##";
PB
666 PRINTTAB(21)STRING$(1Ø,"-")
671 PRINT"TOTAL PAID";:PRINTTAB(
21);:PRINTUSING"$######.##";TP
676 Z=Z/24
681 PRINT:PRINT"YEARS TO RETIRE
LOAN.. ";:PRINTUSING"##.#";Z
686 PRINT"1/2 NORMAL PAYMENT....
.";:PRINTUSING"$###.##";NP
691 PRINT"INTEREST RATE.........
";AI;"%"
696 POKE65494,Ø
7Ø1 C$=INKEY$:IFC$=""THEN 7Ø1
7Ø6 IFC$="P"THEN GOSUB 1381 ELSE
 711
711 NP=NP*2:TP=Ø:TI=Ø:CB=PB:PB=Ø
:GOTO 136
716 CLS:PRINT@264,"NOW PRINTING"
721 PRINT#-2,TAB(3Ø)"OPTION";S
726 PRINT#-2,"YEAR";TAB(6)"INT.P
AID";TAB(18)"PRINCIPAL PAID";TAB
(35)"TOTAL PAYMENTS";TAB(51)"OUT
STANDING BALANCE"
731 NP=NP/2:DP=14:Z=1:H=26:Y=1
736 DI=AI/365
741 IN=DI*CB*DP
746 P=NP-IN
751 PB=PB+P
756 CB=CB-P
761 TP=TP+NP
766 TI(S)=TI(S)+IN
771 IF CB<=ØTHEN 786ELSE776
776 IFZ-H=ØTHEN781ELSEZ=Z+1:GOTO
741
781 PRINT#-2,Y;:PRINT#-2,TAB(6);
:PRINT#-2,USING"$######.##";TI(S
);:PRINT#-2,TAB(18);:PRINT#-2,US
ING"$######.##";PB;:PRINT#-2,TAB
(35);:PRINT#-2,USING"$######.##"
;TP;:PRINT#-2,TAB(51);:PRINT#-2,
USING"$######.##";CB:Y=Y+1:Z=Z+1
:H=H+26:GOTO 741
786 TI(S)=TI(S)-CB:PB=PB+CB
791 PRINT#-2,"":PRINT#-2,"YEARS"
;TAB(7)"TOTAL INTEREST";TAB(23)"
TOTAL PRINCIPAL";TAB(4Ø)"TOTAL P
AYMENTS"
796 Z=Z/24
8Ø1 PRINT#-2,"":PRINT#-2,USING"#
#.#";Z;:PRINT#-2,TAB(7);:PRINT#-
2,USING"$######.##";TI(S);:PRINT
#-2,TAB(23);:PRINT#-2,USING"$###
###.##";PB;:PRINT#-2,TAB(4Ø);:PR
INT#-2,USING"$######.##";TP
8Ø6 NP=NP*2:CB=PB:TP=Ø:TI=Ø:PB=Ø
811 PRINT"HIT ANY KEY TO CONTINU
E"
816 K$=INKEY$:IFK$=""THEN816ELSE
136
821 CLS:PRINT@195,"amount of lum
p sum payment":M=12
826 INPUT LS:GOTO231
831 TP=TP+LS:CB=CB-LS:PB=PB+LS:Z
=Z+1:M=M+12:GOTO281
836 CB=CB-LS:PB=PB+LS:TP=TP+LS:Z
=Z+1:M=M+12:GOTO 5Ø1
841 CLS: PRINT@192,"amount of ex
cess payment":INPUT EP
846 GOTO 231
851 PRINT#-2,TAB(25)"LUMP SUM AM
OUNT ";:PRINT#-2,USING"$####.##"
;LS:RETURN
856 PRINT#-2,TAB(22)"AMOUNT OF E
XCESS PAYMENT ";:PRINT#-2,USING"
$####.##";EP:RETURN
861 CLS:PRINT@164,"how much will
 you finance":INPUTPV
866 CLS:PRINT@164,"how many mont
hs":INPUTM
871 CLS:PRINT@163,"what is the i
nterest rate":INPUTK
876 IFK<1THEN GOTO 951
881 K=K/12:K=K/1ØØ
886 C=(1+K)^M:C=C-1
891 D=(K+1)^M:D=D*K
896 C=C/D
9Ø1 IFPV=ØTHENGOTO956
9Ø6 A=PV/C
911 K=K*12:K=K*1ØØ
916 CLS:PRINT@96,"AMOUNT OF THE
LOAN":PRINT@118,USING"$######.##
";PV
921 PRINT@162,"NO. OF MONTHS":PR
INT@183,M
926 PRINT@226,"INTEREST RATE":PR
INT@246,K;"%"
931 PRINT@29Ø,"MONTHLY PAYMENTS"
:PRINT@3Ø9,USING"$####.##";A
936 S$=INKEY$:IFS$=""THEN936
941 IF S$="P"THENGOSUB 1381 ELSE
 996
946 GOTO 996
951 CLS:PRINT@228,"PLEASE STATE
INTEREST RATE      AS A VALUE GR
EATER THAN 1":INPUTK:GOTO881
956 CLS:PRINT@162,"what are the
monthly payments":INPUTA
961 PV=A*C:GOTO 911
966 GOSUB 1381
971 PRINT#-2,""
976 PRINT#-2,"AMOUNT OF LOAN";:P
RINT#-2,USING"$######.##";PV
981 PRINT#-2,"MONTHS REQUIRED TO
 RETIRE LOAN";:PRINT#-2,M
986 PRINT#-2,"INTEREST RATE";K;:
PRINT#-2,"%"
991 PRINT#-2,"MONTHLY PAYMENTS";
:PRINT#-2,USING"$####.##";A
996 CLS:PRINT@224,"DO YOU WANT A
NOTHER CALCULATION"
1ØØ1 PRINTTAB(11)"(Y/N)"
1ØØ6 S$=INKEY$:IFS$=""THEN1ØØ6
1Ø11 IFS$="Y" THEN 861 ELSE 1Ø16
1Ø16 CB=PV:K=K/1ØØ:AI=K:NP=A:GOT
O 81
1Ø21 CLS:PRINT@164,"DO YOU WANT
INSTRUCTIONS":PRINT@2Ø5,"( Y/N )
"
1Ø26 K$=INKEY$:IFK$=""THEN 1Ø26
1Ø31 IF K$="Y"THEN 1441 ELSE 115
6
1Ø36 CLS:PRINT"IF YOU ARE PLANNI
NG A HOME       PURCHASE..THE LOA
N CALCULATOR    CAN DETERMINE YOU
R MONTHLY       PAYMENTS...IF YOU
 KNOW WHAT YOU  CAN AFFORD PER MO
NTH,BUT DONT    KNOW THE AMOUNT Y
OU CAN FINANCE  TO ARRIVE AT THIS
 PAYMENT.... "
1Ø41 PRINT"THEN loan calculator
CAN HELP.."
1Ø46 PRINT:PRINT"HIT ANY KEY TO
CONTINUE"
1Ø51 K$=INKEY$:IFK$=""THEN1Ø51EL
SE 1Ø56
1Ø56 CLS:PRINT"WHEN YOU ARE ASKE
D HOW MUCH YOU WILL FINANCE...JU
ST HIT <ENTER> YOU WILL THEN BE
ASKED THE       AMOUNT OF MONTHLY
 PAYMENT.ENTER  WHAT YOU WANT TO
PAY PER MONTH. THE PROGRAM WILL
THEN RETURN THEAMOUNT YOU CAN FI
NANCE TO GIVE
1Ø61 PRINT"YOU THE PAYMENTS YOU
WANT"
1Ø66 PRINT:PRINT:PRINT"HIT ANY K
EY TO CONTINUE"
1Ø71 K$=INKEY$:IFK$=""THEN1Ø71EL
SE 1Ø76
1Ø76 CLS:PRINT"AFTER FINDING OUT
 THE AMOUNT YOUCAN FINANCE. RUN
THE"
1Ø81 PRINTTAB(7)"mortgage saving
s"
1Ø86 PRINT"PORTION OF THIS PROGR
AM."
1Ø91 PRINT"IT WILL SHOW THREE WA
YS YOU CAN SAVE THOUSANDS OF DOL
LARS AND        MANY YEARS OFF YOUR M
ORTGAGE."
1Ø96 PRINT"YOU CAN HAVE A YEARLY
 PRINTOUT       OF ANY OR ALL OPTIONS
.IT WILL        SHOW,BY YEAR,INTEREST
 PAID,TOTAL     PAYMENT,CURRENT BALA
NCE AND         YEARS PAID. IT WILL T
HEN GIVE        TOTALS OF ALL ITEMS.
11Ø1 PRINT:PRINT"HIT ANY KEY TO
CONTINUE"
11Ø6 K$=INKEY$:IFK$=""THEN 11Ø6E
LSE 1111
1111 CLS:PRINT"IN ORDER TO SEE A
N ON SCREEN     COMPARISON OF INT
EREST PAID      ON ALL OPTIONS, Y
OU MUST RUN     OPTION 4 LAST."
1116 PRINT:PRINT"WHEN THERE IS N
O CURSOR ON A     CALCULATED SCRE
EN YOU CAN GET A PRINTOUT BY PRE
SSING THE LETTER p ANY OTHER KEY
 WILL CONTINUE    THE PROGRAM"
1121 PRINT:PRINT"HIT ANY KEY TO
CONTINUE"
1126 K$=INKEY$:IFK$=""THEN1126EL
SE1131
1131 CLS:PRINT"THE AMOUNTS GIVEN
 IN THIS        PROGRAM SHOULD NO
T BE CONSTRUED TO BE EXACT AMOUN
TS YOU WILL PAYOR SAVE..... BUT
SHOULD BE USED ONLY AS A GUIDE T
O REPRESENT     YOUR PAYMENTS AND
 SAVINGS"
1136 PRINT"OPTION 2 OF THIS PROG
RAM MAY NOTBE ACCEPTABLE TO YOUR
 LENDER     PLEASE CHECK WITH THE
M BEFORE    ATTEMPTING THIS METHO
D.
1141 PRINT"IF YOU ARE CONSIDERIN
G A HOME    PURCHASE THIS IS AN A
TTRACTIVE   WAY TO SET UP YOUR LO
AN PAYOFF..... AS YOU WILL SEE.
1146 PRINT"HIT ANY KEY TO CONTIN
UE"
1151 K$=INKEY$:IFK$=""THEN1151EL
SE1156
1156 CLS:PRINT"DO YOU WANT TO GO
 TO THE LOAN    CALCULATOR OR TO
MORTGAGE        PLANNER"
1161 PRINT:PRINT"IF YOU CHOOSE T
HE CALCULATOR     PORTION OF THE
PROGRAM THE       LAST AMOUNTS US
ED WILL BE AUTO-  MATICALLY BE EN
```

```
TERED INTO THE   MORTGAGE PLANNE
R"
1166 PRINT:PRINT"PRESS <C> FOR C
ALCULATOR AND <M> FOR MORTGAGE P
LANNER"
1171 S$=INKEY$:IFS$=""THEN 1171
1176 IF S$="C"THEN 861 ELSE 186
1181 FORL=1Ø24TO1Ø55
1186 POKEL,191:NEXTL
1191 L=1Ø56
1196 POKEL,191
12Ø1 L=L+32
12Ø6 IFL=15Ø4+32THEN1211ELSE1196
1211 FORL=15Ø4TO1535
1216 POKEL,191:NEXTL
1221 L=1535
1226 POKEL,191
1231 L=L-32
1236 IFL=1Ø55 THEN1241ELSE 1226
1241 FORT=1TO7ØØ*2:NEXTT:RETURN
1246 CLS:PRINT"DO YOU WANT TO CO
MPARE INTEREST ON ALL FOUR OPTIO
NS"
1251 PRINT:PRINTTAB(11)"(Y/N)"
1256 K$=INKEY$:IFK$=""THEN1256
1261 IF K$="Y" THEN GOTO15Ø1ELSE
 RETURN
1266 CLS:S=1
1271 TI(S)=TI(S)*1ØØØØ
1276 PRINT:PRINT"OPTION";S;:PRIN
TUSING"$######.##";TI(S)
1281 S=S+1 :IF S=5 THEN 1286 ELS
E 1271
1286 IFTI(2)<=TI(3)THEN1291ELSE1
296
1291 IFTI(2)=ØTHEN1296 ELSE IFTI
(2)<=TI(4)THEN13Ø1ELSE1296
1296 IFTI(3)=ØTHEN 1311 ELSEIFTI
(3)<=TI(4) THEN 13Ø6 ELSE1311
13Ø1 TI(6)=TI(1)-TI(2):GOTO1316
13Ø6 TI(6)=TI(1)-TI(3):GOTO1316
1311 TI(6)=TI(1)-TI(4):GOTO1316
1316 PRINT:PRINT"YOU CAN SAVE ";
:PRINTUSING"$######.##";TI(6)
1321 K$=INKEY$:IFK$=""THEN1321
1326 IFK$="P"THEN GOSUB 1381 ELS
E 1331
1331 PRINT:PRINT"PRESS <R> TO RE
VIEW OPTIONS AND RUN PROGRAM AGA
IN"
1336 PRINT:PRINT"PRESS ANY OTHER
 KEY TO END"
1341 K$=INKEY$:IFK$=""THEN1341
1346 IFK$="R"THEN 1366 ELSE 1351
1351 CLS:PRINT@2ØØ,"happy saving
"
1356 END
1361 GOSUB 1181
1366 S=1
1371 TI(S)=Ø:PB=Ø:Z=Ø:TB=Ø:TP=Ø:
LS=Ø:EP=Ø:Y=Ø:S=S+1:C2=Ø
1376 IF S=4+1 THEN 1156 ELSE 137
1
1381 ZZ=Ø
1386 FORXX=1Ø24TO1535
1391 YY=PEEK(XX):ZZ=ZZ+1
1396 PP=YY AND 127
14Ø1 IF PP>95 THENPP=PP-64
14Ø6 PRINT#-2,CHR$(PP);
1411 IF ZZ=32 THEN PRINT#-2:ZZ=Ø
1416 NEXT XX
1421 RETURN
1426 PRINT"LUMP SUM AMOUNT.....
. ";:PRINTTAB(21);:PRINTUSING"$#
###.##";LS:RETURN
1431 PRINT"EXCESS PAYMENT.....
";:PRINTTAB(21);:PRINTUSING"$###
.##";EP:RETURN
1436 CLS
1441 CLS:PRINT"HERE'S THE BEST I
DEA TO COME    ALONG IN QUITE A
WHILE: PAY OFF YOUR MORTGAGE FAS
TER,IN ORDER   TO GET A QUICKER
EQUTIY BUILDUP IN YOUR HOUSE.
1446 PRINT"IT USED TO COST A LIT
TLE MORE    EACH MONTH TO REACH
THIS GOAL. BUT A NEW KIND OF MOR
TGAGE THAT IS JUST TAKING HOLD I
N THE U.S. CAN MAKE THE PROCESS
ALMOST     PAINLESS.
1451 PRINT"FASTER MORTGAGE PAYME
NTS WILL   SAVE YOU A FORTUNE IN
 INTEREST  RATES.
1456 PRINTTAB(1Ø)"HIT ANY KEY"
1461 K$=INKEY$:IFK$=""THEN1461 E
LSE 1466
1466 CLS:PRINT"THE NEW WAY TO FA
STER HOME OWNERSHIP IS THROUGH A
 BI-WEEKLY     PAYMENT PLAN. YOU
R LOAN IS      AMORTIZED AS IF I
T WERE GOING   TO LAST FOR 3Ø YE
ARS. BUT...    INSTEAD OF PAYING
 ONCE A MONTH, YOU MAKE 1/2  OF
THE MONTHLY"
1471 PRINT"PAYMENT EVERY TWO WEE
KS. THIS   SCHEDULE LEADS TO THE
 EQUIVALENTOF 13 MONTHLY PAYMENT
S EVERY    YEAR RATHER THAN THE
USUAL 12."
1476 PRINT"THIS MAY NOT SOUND LI
KE MUCH OF A CHANGE. BUT ITS EFF
ECT IN     CUTTING THE TIME AND
COST OF ANYMORTGAGE IS ASTOUNDIN
G."
1481 K$=INKEY$:IFK$=""THEN1481 E
LSE 1486
1486 CLS:PRINT"THIS METHOD OF RE
PAYMENT IS     option 2.YOU WILL
 SEE HOW MUCH  CAN BE SAVED WITH
 THIS OPTION   WHEN YOU COMPARE
INTEREST PAID  ON ALL OPTIONS."
1491 PRINT:PRINTTAB(1Ø)"HIT ANY
KEY"
1496 K$=INKEY$:IF K$=""THEN 1496
 ELSE 1Ø36
15Ø1 CLS
15Ø6 FOR L=1Ø24 TO 1Ø55
1511 POKE L,175:NEXT L
1516 L=1Ø56
1521 S=1:X=Ø
1526 X=49
1531 POKE L,X
1536 L=L+64
1541 X=X+1
1546 IF X=53 THEN 1551 ELSE 1531
1551 FOR L=128Ø TO 1311
1556 POKE L,175
1561 NEXT L
1566 FOR L=1Ø25 TO 128Ø STEP 32
1571 X=175
1576 POKE L,X:NEXT L
1581 PRINT@291,"X 1  5  1Ø  15
2Ø  25  3Ø"
1586 PRINT@32Ø,"INTEREST = X TIM
ES  $1Ø,ØØØ"
1591 S=1:X=Ø
1596 L=1Ø58+X
16Ø1 TI(S)=TI(S)/1ØØØØ
16Ø6 F=FIX(TI(S))
1611 FORL=L TO L+F
1616 POKEL,191
1621 NEXTL
1626 X=X+64
1631 S=S+1:IF S=4+1 THEN 1641 EL
SE 1596
1636 POKEL,191
1641 PRINT@384,"HIT <C> TO CONTI
NUE PROGRAM "
1646 PRINT@416,"ANY OTHER KEY WI
LL END PROGRAM"
1651 K$=INKEY$:IF K$=""THEN1651
1656 IFK$="C"THEN 1266ELSE 1351
```

continued from Page 17

6) List Month Actual vs. Budget

Provides a summarized listing of account versus budget amounts for a chosen month. A difference (variance) between actual and budget is shown. Credit (-) amounts in the variance column are unfavorable (over budget) while debit variances are favorable (under budget). See Exhibit 6.

Exhibit 8

ACT	<--D E S C R I P T I O N-->
1	HOUSE MORTGAGE
2	INSURANCE
3	BANK CARDS
4	ELECTRICITY
5	AUTO PAYMENT
6	GROCERIES
7	CLOTHING
8	MISCELLANEOUS

7) List YTD Actual vs. Budget

Provides the same information as Selection 5, except it reflects data through a specified month. See Exhibit 7.

8) List Chart of Accounts

Allows for a listing of your chart of accounts. This is a handy reference listing. See Exhibit 8.

9) Return to Main Program

This option returns you to the main program, *Exptrakr*, to continue other activities.

10) End Session

Select this option if you wish to terminate the program.

Although this set of programs offers various options for use in budgeting and tracking expenses, enhancements such as displaying all account detail expenditures for a year or allowing for more transactions in a month can be made. It is hoped that the programs are helpful and create an interest in expanding their usefulness. May all of your expenditures be small.

(Editor's Note: This program runs to six pages of code. In the interest of providing you with as much information as possible, we have decided to exclude the listing from the magazine.

The program of course is available on Australian Rainbow on Tape for this month. G.)

A Simple Technique for Creating Animation

By Joseph Kolar

Animation gives any CoCo graphics program a lot of pizazz with the illusion of movement. The technique of creating animation seems beyond the capability of the newcomer to CoColand. It is not the formidable project that it appears to be.

The good news is that the beginner need not be overwhelmed by animation creation. He can do it with a minimum of artistic talent. Today, we are going to make like we are Rembrandts, and do some simple but satisfying animation. Using artistic license, we will create a "stick bird." We will take this bird, set it in flight and give it a chance to soar on our screen. We will create various stick shapes to add variety and give the appearance of graceful flight.

Look at Listing 1 and key in lines 10 and 1000. PCLS3 gives us the background blue sky upon which the bird is highlighted. Key in lines 20 to 27, the eight forms of the bird we will use in our effort to animate. They are called by the variables assigned them. The reason we use various forms of the bird is to create the illusion of a change in the bird as it wheels, soars or just flaps its wings. If we use just one shape throughout (one variable), the bird looks stiff and its flight stilted.

Key in Line 30. All the birds are displayed using the DRAW statement. The color, horizontal and vertical locations are included within quote marks. The desired bird shape is selected by picking the desired variable and added with the good old concatenation marker, '+'. Now RUN and you should be suitably unimpressed to see what is supposed to be a bird. Press BREAK and, one at a time, substitute the other variables in Line 30 to see the so-called birds in our repertoire. After you have seen them all, replace the original variable, B$.

Keep in mind that there are many ways to develop animation. The following system is somewhat unwieldy, but lends itself admirably for the purpose of this tutorial.

Run the program and note that we place a bird, in color C2, at both a horizontal and vertical location of 10. The bird (B$ in this case), has the left wing "up" and the right wing in a horizontal plane. At this time, also note that in lines 40 through 350, the color (C2) is redundantly included in every DRAW statement. In Listing 1 only the first C2 in Line 30 is required to maintain the same color of the bird. CoCo knows that C2 is desired in all the subsequent DRAW lines. At a later stage in our artistic endeavors, it will be necessary to insert C2 in all of the DRAW lines in this listing. To save a lot of time and monotonous editing later on, we will put them in as we proceed.

At this stage, we will place the various-shaped birds at locations we deem either logical or interesting, one at a time, ever increasing the number of birds in the flight plan. Press BREAK, key in Line 40 and RUN. You will see a second bird form on the screen (A$), with both wings in the "up" position. We moved it over to the right five units, to 15 on the horizontal, and 10 units down on the vertical. We now have two birds on the screen.

For the purpose of this tutorial, we will move either zero, five or 10 units from a previous location. This creates a smooth transition from one location to the next.

Press BREAK, key in Line 50 and run. The same 'V' bird is flying to the right. Comparing lines 40 and 50 in the listing, you can verify that we moved the bird 10 units to the right. Press BREAK, key in Line 60 and run. You can see the same bird heading to the right.

Don't get excited if the birds overlap on the screen. It will all be sorted out later. At this stage, we are plotting the flight path and want to see every shape and its location as we create it. This way you can locate a desired shape at the location you feel is right. If you don't like the shape or location, it can be altered now without disturbing future additions to the flight plan. You won't have to dissect your program and wearily rearrange it later.

We are so creative that we plot our animation program directly on the screen. However, you may use graph paper if you wish to determine the shapes and locations of the birds.

Press BREAK, key in Line 70 and run. Here I promptly violated my own rule and dropped the bird, B$, 15 units. I had a down draft in mind, which I imagined might cause the bird to drop more than usual and change directions by going five units to the right.

Press BREAK, key in Line 80 and run. Here the bird, E$, is wheeling to the right and down; press BREAK again.

If you have difficulty viewing a shape that is superimposed over a previous shape, temporarily mask the previous line with a REM marker. RUN to see if it is what you intended, then remove the REM marker from the previous line. At this point, key in each program line one at a time, and check to see what shape you added where.

We have completed our first phase: creating, locating and displaying each bird. Now that we have created each bird in our tableau, we have to make them vanish. Beginning with the bird at Line 30 through the last one at Line 350, we will erase them. LIST 30 and add +5 to Line 30 to create the "erase" line. Thus, all program lines used to erase the birds will end in '5'. Key in 35, the line number, and copy the data appearing in Line 30 that you have on the screen changing only the digit (2) in C2 to '3'. Now run. C3 is the same color as PSCL3. The bird is still, invisible and effectively erased.

If you care to check this out, temporarily change PCLS3 to PCLS2 in Line 10 and RUN. See? Restore Line 10 to its

original state.

Press BREAK and LIST40. Create Line 45 and copy the scoop in Line 40, making the desired color change. Proceed line by line, every once in a while pausing to RUN and make sure you have erased all the birds. If some residue remains on the screen, you made a booboo in copying! When you check, you will be excited to see the vestiges of a bird in flight, if ever so fleetingly.

When you have finished the second phase and check out your work, you should get a fleeting glimpse of the bird in motion and end up with a blank, blue display.

Obviously, we must create pauses between the creation and disappearance of each bird so ordinary mortals can observe the flight. The third phase determines the length of time each bird is visible. We will use a pause routine to accomplish this feat: FOR Z= 1 TO X: NEXT, where 'X' is some value between 75 and 200. We will use increments of 25, so for 'X' we will use the following values: 75, 100, 125, 150, 175 and 200 to keep it simple and under control.

After you get the idea, you can substitute your values for the chosen ones in the listing to make it fly the way you want it to fly. First, let me give you the system we will use in this tutorial.

Type in LIST 30-40. We list two lines to see how far the bird moved. We note that the bird moved 10 units down and five units to the right. We compare either Line 30 or 35 to Line 40. We will place the pause line immediately following the creation line (Line 30). Each pause line will be numbered by incrementing the creation line by +1 and the erase line by +5. We will try 100 as the length of the pause. Key in 31 FOR Z=1TO100:NEXT and run. You can't see much! Press BREAK, LIST40-50 and let's make this pause line shorter in duration by using 75. Key in 41 FOR Z=1TO75:NEXT and run, then press BREAK and LIST50-60. We'll use 100. Key in 51 FOR Z=1TO100:NEXT.

Follow the same procedure using 150 in Line 61, 100 in Line 71 and 75 at both lines 81 and 91. RUN and observe the movement. Press BREAK and adjust it to suit yourself. It is your bird! Make it fly as you would imagine it should fly. Vary the time lapse, preferably a higher figure for a large location displacement and a shorter lapse for a small movement, but do it from one line to the next in a methodical manner, ensuring that all previously determined time pause lines are satisfactory. You may compare the pauses you chose with the ones in Listing 2.

You may want the movement to be quicker so it looks even more natural. If so, lower the value of each pause line by 25 or 50 units. For that matter, you may prefer to substitute other shapes (bird variables). Be my guest! When this phase of the program is completed and all the pauses are set, you may want to change Line 1000 (1000 GOTO10).

Since many of your pause lines are repetitious, this is an ideal occasion to use GOSUB. For instance, add 400 FOR Z=1TO75:NEXT:RETURN and change lines 41, 81, 91, 141, 221, 231, 241, 251, 261, 281, 291, 301, 311, 321 and 341 to GOSUB400. You will have to put in a line, 360 GOTO10 or 360 GOTO1000, to walk around the GOSUB routine. Ideally, the GOSUB should be at the end of the program, for example, Line 2000. However, using 400 instead of 2000 saves typing one extra zero and whatever error that third zero might generate due to typing mistakes. Naturally, you can make other GOSUB lines to accommodate frequently-used pause lines.

OK, what have we wrought? Nothing much! Just a bunch of lines flapping across the screen. But, you learned a lot quite painlessly. Let us recap:

1) A picture, design or shape must be created. It can be elaborate or as mindlessly simple as our bird. After it is created and put into a variable form, it can be called using DRAW. Alternate shapes should be created in anticipation of need, but they can be created as required and added to the list of shapes.

2) It must be located at the desired site on the screen in a color other than the background.

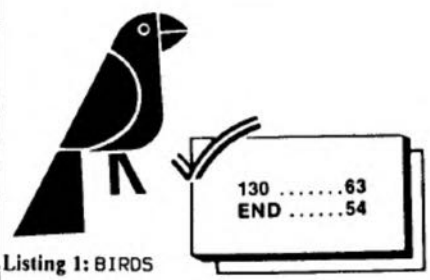
3) It must remain on the screen for a certain length of time.

4) It must be erased by creating the same design and in the exact location but using the background color so it appears to vanish.

5) The same picture or a variant, again created and called as a variable in a DRAW statement, can be placed in a newly selected location. Repeat steps two through four. Suppose you made a pastoral scene in the blank space reserved under the bird? Or the outline of a few buildings?

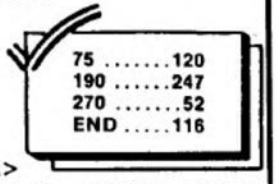
Now that you know how to make a bird fly around, you can use the same technique to produce your own creation.

As an added attraction, Listing 3 uses SOUND as a timer and has a skyline thrown in to show how to enhance the animation. You can modify your tutorial program by inserting lines 11-13, modify Line 1000 and change all pause lines to SOUND lines. If you don't care for my sounds, make up your own.



Listing 1: BIRDS

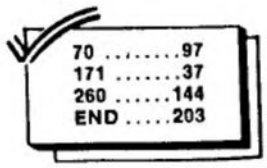
```
Ø 'LISTING1
1Ø PMODE3,1:PCLS3:SCREEN1,Ø
2Ø A$="F6E6"
21 B$="F6R6"
22 C$="R6E6"
23 D$="E6F6"
24 E$="F12"
25 F$="E12"
26 G$="R12"
27 H$="R6F6"
3Ø DRAW"C2BM1Ø,1Ø"+B$
4Ø DRAW"C2BM15,2Ø"+A$
5Ø DRAW"C2BM25,2Ø"+A$
6Ø DRAW"C2BM35,2Ø"+A$
7Ø DRAW"C2BM4Ø,35"+B$
8Ø DRAW"C2BM5Ø,4Ø"+E$
9Ø DRAW"C2BM6Ø,45"+B$
1ØØ DRAW"C2BM7Ø,45"+A$
11Ø DRAW"C2BM75,55"+A$
12Ø DRAW"C2BM85,6Ø"+A$
13Ø DRAW"C2BM9Ø,7Ø"+A$
14Ø DRAW"C2BM95,75"+C$
15Ø DRAW"C2BM11Ø,9Ø"+D$
16Ø DRAW"C2BM12Ø,9Ø"+D$
17Ø DRAW"C2BM13Ø,95"+D$
18Ø DRAW"C2BM14Ø,95"+C$
19Ø DRAW"C2BM15Ø,95"+F$
2ØØ DRAW"C2BM14Ø,8Ø"+A$
21Ø DRAW"C2BM13Ø,75"+A$
22Ø DRAW"C2BM12Ø,7Ø"+A$
23Ø DRAW"C2BM12Ø,6Ø"+A$
24Ø DRAW"C2BM12Ø,5Ø"+A$
25Ø DRAW"C2BM125,45"+B$
26Ø DRAW"C2BM12Ø,4Ø"+E$
27Ø DRAW"C2BM125,35"+E$
28Ø DRAW"C2BM13Ø,25"+A$
29Ø DRAW"C2BM135,2Ø"+A$
3ØØ DRAW"C2BM14Ø,15"+A$
31Ø DRAW"C2BM145,15"+C$
32Ø DRAW"C2BM145,1Ø"+G$
33Ø DRAW"C2BM15Ø,1Ø"+H$
34Ø DRAW"C2BM16Ø,1Ø"+D$
35Ø DRAW"C2BM16Ø,5"+D$
1ØØØ GOTO1ØØØ
```



Listing 2: FLIGHT 1

```
Ø '<LISTING2>
1 'CREATED BY J. KOLAR, 1985
1Ø PMODE3,1:PCLS3:SCREEN1,Ø
2Ø A$="F6E6"
21 B$="F6R6"
```

```
22 C$="R6E6"
23 D$="E6F6"
24 E$="F12"
25 F$="E12"
26 G$="R12"
27 H$="R6F6"
3Ø DRAW"C2BM1Ø,1Ø"+B$
31 FOR Z=1TO1ØØ:NEXT
35 DRAW"C3BM1Ø,1Ø"+B$
4Ø DRAW"C2BM15,2Ø"+A$
41 FOR Z=1TO75:NEXT
45 DRAW"C3BM15,2Ø"+A$
5Ø DRAW"C2BM25,2Ø"+A$
51 FOR Z=1 TO 1ØØ:NEXT
55 DRAW"C3BM25,2Ø"+A$
6Ø DRAW"C2BM35,2Ø"+A$
61 FOR Z=1 TO 15Ø:NEXT
65 DRAW"C3BM35,2Ø"+A$
7Ø DRAW"C2BM4Ø,35"+B$
71 FOR Z=1TO1ØØ:NEXT
75 DRAW"C3BM4Ø,35"+B$
8Ø DRAW"C2BM5Ø,4Ø"+E$
81 FORZ=1TO75:NEXT
85 DRAW"C3BM5Ø,4Ø"+E$
9Ø DRAW"C2BM6Ø,45"+B$
91 FOR Z=1TO 75:NEXT
95 DRAW"C3BM6Ø,45"+B$
1ØØ DRAW"C2BM7Ø,45"+A$
1Ø1 FOR Z=1TO15Ø:NEXT
1Ø5 DRAW"C3BM7Ø,45"+A$
11Ø DRAW"C2BM75,55"+A$
111 FOR Z=1TO125:NEXT
115 DRAW"C3BM75,55"+A$
12Ø DRAW"C2BM85,6Ø"+A$
121 FOR Z=1 TO 125:NEXT
125 DRAW"C3BM85,6Ø"+A$
13Ø DRAW"C2BM9Ø,7Ø"+A$
131 FOR Z=1TO75:NEXT
135 DRAW"C3BM9Ø,7Ø"+A$
14Ø DRAW"C2BM95,75"+C$
141 FOR Z=1TO175:NEXT
145 DRAW"C3BM95,75"+C$
15Ø DRAW"C2BM11Ø,9Ø"+D$
151 FOR Z=1TO1ØØ:NEXT
155 DRAW"C3BM11Ø,9Ø"+D$
16Ø DRAW"C2BM12Ø,9Ø"+D$
161 FOR Z=1TO1ØØ:NEXT
165 DRAW"C3BM12Ø,9Ø"+D$
17Ø DRAW"C2BM13Ø,95"+D$
171 FOR Z=1 TO 125:NEXT
175 DRAW"C3BM13Ø,95"+D$
18Ø DRAW"C2BM14Ø,95"+C$
181 FORZ=1TO125:NEXT
185 DRAW"C3BM14Ø,95"+C$
19Ø DRAW"C2BM15Ø,95"+F$
191 FOR Z=1TO 175:NEXT
195 DRAW"C3BM15Ø,95"+F$
2ØØ DRAW"C2BM14Ø,8Ø"+A$
2Ø1 FOR Z=1TO125:NEXT
2Ø5 DRAW"C3BM14Ø,8Ø"+A$
21Ø DRAW"C2BM13Ø,75"+A$
211 FOR Z=1TO1ØØ:NEXT
215 DRAW"C3BM13Ø,75"+A$
22Ø DRAW"C2BM12Ø,7Ø"+A$
221 FORZ=1TO75:NEXT
225 DRAW"C3BM12Ø,7Ø"+A$
23Ø DRAW"C2BM12Ø,6Ø"+A$
231 FOR Z=1TO75:NEXT
235 DRAW"C3BM12Ø,6Ø"+A$
24Ø DRAW"C2BM12Ø,5Ø"+A$
241 FOR Z=1TO75:NEXT
245 DRAW"C3BM12Ø,5Ø"+A$
25Ø DRAW"C2BM125,45"+B$
251 FOR Z=1TO75:NEXT
255 DRAW"C3BM125,45"+B$
26Ø DRAW"C2BM12Ø,4Ø"+E$
261 FOR Z=1TO75:NEXT
265 DRAW"C3BM12Ø,4Ø"+E$
27Ø DRAW"C2BM125,35"+E$
271 FOR Z=1 TO 1ØØ:NEXT
275 DRAW"C3BM125,35"+E$
28Ø DRAW"C2BM13Ø,25"+A$
281 FOR Z=1TO75:NEXT
285 DRAW"C3BM13Ø,25"+A$
29Ø DRAW"C2BM135,2Ø"+A$
291 FORZ=1TO75:NEXT
295 DRAW"C3BM135,2Ø"+A$
3ØØ DRAW"C2BM14Ø,15"+A$
3Ø1 FOR Z=1TO 75:NEXT
3Ø5 DRAW"C3BM14Ø,15"+A$
31Ø DRAW"C2BM145,15"+C$
311 FOR Z=1TO75:NEXT
315 DRAW"C3BM145,15"+C$
32Ø DRAW"C2BM145,1Ø"+G$
321 FOR Z=1TO 75:NEXT
325 DRAW"C3BM145,1Ø"+G$
33Ø DRAW"C2BM15Ø,1Ø"+H$
331 FOR Z=1TO125:NEXT
335 DRAW"C3BM15Ø,1Ø"+H$
34Ø DRAW"C2BM16Ø,1Ø"+D$
341 FOR Z=1TO 75:NEXT
345 DRAW"C3BM16Ø,1Ø"+D$
35Ø DRAW"C2BM16Ø,5"+D$
351 FOR Z=1TO2ØØ:NEXT
355 DRAW"C3BM16Ø,5"+D$
1ØØØ GOTO1Ø
```



Listing 3: FLIGHT 2

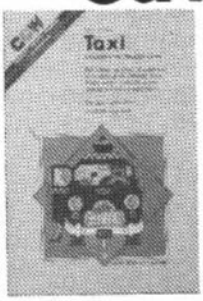
```
Ø '<LISTING3>
1 ' CREATED BY J. KOLAR, 1985
1Ø PMODE3,1:PCLS3:SCREEN1,Ø
11 DRAW"C1BMØ,14ØR4ØD1ØR1ØU9ØR1Ø
D3ØR5ØD5ØR4ØU4ØR1ØD4ØR1ØU9ØR4ØD1
ØØR1ØU1ØR1ØD1ØR1ØU4ØR2ØBD5ØL255"
12 PAINT(5,145),2,1
13 PAINT(5,19Ø),1,1
2Ø A$="F6E6"
21 B$="F6R6"
22 C$="R6E6"
23 D$="E6F6"
24 E$="F12"
25 F$="E12"
26 G$="R12"
27 H$="R6F6"
3Ø DRAW"C2BM1Ø,1Ø"+B$
31 SOUND125,3
35 DRAW"C3BM1Ø,1Ø"+B$
4Ø DRAW"C2BM15,2Ø"+A$
41 SOUND 89,2
45 DRAW"C3BM15,2Ø"+A$
5Ø DRAW"C2BM25,2Ø"+A$
51 SOUND125,3
55 DRAW"C3BM25,2Ø"+A$
6Ø DRAW"C2BM35,2Ø"+A$
61 SOUND159,2
65 DRAW"C3BM35,2Ø"+A$
7Ø DRAW"C2BM4Ø,35"+B$
71 SOUND125,3
75 DRAW"C3BM4Ø,35"+B$
8Ø DRAW"C2BM5Ø,4Ø"+E$
81 SOUND89,2
85 DRAW"C3BM5Ø,4Ø"+E$
9Ø DRAW"C2BM6Ø,45"+B$
91 SOUND89,3
95 DRAW"C3BM6Ø,45"+B$
1ØØ DRAW"C2BM7Ø,45"+A$
1Ø1 SOUND159,2
1Ø5 DRAW"C3BM7Ø,45"+A$
11Ø DRAW"C2BM75,55"+A$
111 SOUND133,2
115 DRAW"C3BM75,55"+A$
12Ø DRAW"C2BM85,6Ø"+A$
121 SOUND133,2
125 DRAW"C3BM85,6Ø"+A$
13Ø DRAW"C2BM9Ø,7Ø"+A$
131 SOUND89,2
135 DRAW"C3BM9Ø,7Ø"+A$
14Ø DRAW"C2BM95,75"+C$
141 SOUND89,2
145 DRAW"C3BM95,75"+C$
15Ø DRAW"C2BM11Ø,9Ø"+D$
151 SOUND125,2
155 DRAW"C3BM11Ø,9Ø"+D$
16Ø DRAW"C2BM12Ø,9Ø"+D$
161 SOUND125,2
165 DRAW"C3BM12Ø,9Ø"+D$
17Ø DRAW"C2BM13Ø,95"+D$
171 SOUND125,2
175 DRAW"C3BM13Ø,95"+D$
18Ø DRAW"C2BM14Ø,95"+C$
181 SOUND133,2
185 DRAW"C3BM14Ø,95"+C$
19Ø DRAW"C2BM15Ø,95"+F$
191 SOUND17Ø,2
195 DRAW"C3BM15Ø,95"+F$
2ØØ DRAW"C2BM14Ø,8Ø"+A$
2Ø1 SOUND133,2
2Ø5 DRAW"C3BM14Ø,8Ø"+A$
21Ø DRAW"C2BM13Ø,75"+A$
211 SOUND 125,2
215 DRAW"C3BM13Ø,75"+A$
22Ø DRAW"C2BM12Ø,7Ø"+A$
221 SOUND89,2
225 DRAW"C3BM12Ø,7Ø"+A$
23Ø DRAW"C2BM12Ø,6Ø"+A$
231 SOUND89,2
235 DRAW"C3BM12Ø,6Ø"+A$
24Ø DRAW"C2BM12Ø,5Ø"+A$
241 SOUND89,2
245 DRAW"C3BM12Ø,5Ø"+A$
25Ø DRAW"C2BM125,45"+B$
251 SOUND89,2
255 DRAW"C3BM125,45"+B$
26Ø DRAW"C2BM12Ø,4Ø"+E$
261 SOUND89,2
265 DRAW"C3BM12Ø,4Ø"+E$
27Ø DRAW"C2BM125,35"+E$
271 SOUND125,2
275 DRAW"C3BM125,35"+E$
28Ø DRAW"C2BM13Ø,25"+A$
281 SOUND89,2
285 DRAW"C3BM13Ø,25"+A$
29Ø DRAW"C2BM135,2Ø"+A$
291 SOUND89,2
295 DRAW"C3BM135,2Ø"+A$
3ØØ DRAW"C2BM14Ø,15"+A$
3Ø1 SOUND89,2
3Ø5 DRAW"C3BM14Ø,15"+A$
31Ø DRAW"C2BM145,15"+C$
311 SOUND 89,2
315 DRAW"C3BM145,15"+C$
32Ø DRAW"C2BM145,1Ø"+G$
321 SOUND89,2
```

continued on Page 41

Tandy ELECTRONICS

Color Computer Educational Software for Children

Save $10 on any of these!



Drive Your Own Taxi

Reg 34.95 **24.95**

Taxi 7 years and older. Choose city to drive in (London, New York etc.), pick up passengers, stop at traffic lights etc. Joysticks required. 26-2509

Makes Adding Up Fun

Reg 39.95 **29.95**

Grover's Number Rover 3-6 years 6 games-in-one! Help Grover pick up Twiddlebugs. Add, subtract, etc. A great learning aid. 26-2522

Teaches Children Shapes

Reg 39.95 **29.95**

Ernie's Magic Shapes Designed for ages 3-6. Choose difficulty level. Match color and shapes and then zap shapes away. 26-2524

Learn Letters Easily

Reg 39.95 **29.95**

Cookie Monster's Letter Crunch Ages 3-6. Bake cookies with the right letters on and he'll eat it. Turn letters into words. Requires joystick. 26-2526

Recognise Amimals, Plants

Reg 39.95 **29.95**

Big Bird's Special Delivery Ages 3-6. Match packages to shapes, take them to the right store. Animals, plants, food, instruments. 26-2525

Save $15 on any of these!

Let the Kids Use Keys

Reg 59.95 **44.95**

Kids on Keys Ideal introduction to the keyboard. Three games for location of letters and numbers. Four levels of difficulty. 26-3167

A Fun Way to Computers

Reg 59.95 **44.95**

Face Maker Educational program for younger users. Three games in one. Control the adding of eyes, nose, ears, hair, etc. on blank face. 26-3166

Playing with Sights & Noises

Reg 59.95 **44.95**

Kinder Comp Ages 3-8. Create colorful pictures, scribble on screen, make sounds, animation, find letters. A perfect way to learn. 26-3168

Learning Fractions Easily

Reg 59.95 **44.95**

Fraction Fever Ages 7-adult. Hop along on a pogo stick, checking for matching fractions, zap incorrect ones. Just like an arcade game! 26-3169

How to Spell & Have Fun

Reg 69.95 **54.95**

Alphabet Zoo Ages 3-8. Teaches relationship between letters and sounds. Race through maze, altering letters that form a picture. 26-3170

Ask about Our Special Order Educational Software for Color Computer and TANDY 1000

SALE ENDS: 30/4/86

VALUE! Color Computer External Expansion

Save $50

Reg **149.95**

99.95

Multi-Pak Interface allows you to connect up to four Program Pak cartridges to your Color Computer at once. It also makes it possible to connect devices such as a Color Computer disk drive and graphics tablet. When you're ready to change from one Program Pak to another, just move the selector switch on the front, or change between slots under program control. Upgrade your computer with a disk drive now, or just make it easier to change software programs. Attractive, durable and easy to use. AC operation. 26-3024

The Tandy Model 4 Desktop Computer

Save $500

$1499

Reg **1999.00**

A true price breakthrough! The **Tandy Model 4** is the perfect computer for busy managers, educators and home computer users. Boasts 64K memory, expandable to 128K, numeric keypad, RAM based Disk Drives, CP/M compatible, Parallel Printer interface. Huge ready-to-run software library. Comes with owner's manual and introductory booklet to get you started.

26-1069

No rainchecks on this item

WE SERVICE WHAT WE SELL!

Tandy ELECTRONICS

A DIVISION OF TANDY AUSTRALIA LIMITED INC IN N.S.W.

Nearly 350 Stores Australia-Wide

Available from 350 Stores Australiawide including Tandy Computer Centres

**Independent Tandy Dealers may not be participating in this ad or have every item advertised. Prices may also vary at individual Dealer Stores*

A Disk Tinkerer's Device

By Martin H. Goodman

There was an excellent article in the December 1985 RAINBOW, "Zapping with Confidence," Page 118 by Jeffry Dwight, that provided a well-designed "disk zap" utility. Now you can have an easy means to examine and modify disks. In this article I will try to aid such hardy tinkerers by discussing some aspects of just what you will see when you look at your disks. Some of this material is explained in the Radio Shack Disk Extended BASIC manual in Chapter 11. Some of the material, however, is not given there, especially the information on specific file structure.

As a bonus, I'll provide you with a utility. Called *Analyzer*, it automatically gathers up the widely separated information on just where given directory files are on your disk and prints it out in a neat fashion. *Analyzer* can be used in conjunction with any disk editor, such as the one mentioned above.

Note 1: There may be some confusion about what number (zero or one) is the first number in a given sequence. The first sector on a track is numbered one, yet the first track on the disk is numbered zero. The first byte in the directory entry is called "byte zero." The first granule is called granule number zero. These are arbitrary conventions. They are not all consistent with each other, and are a pain to remember . . . but remember them a hacker must!

Note 2: When referring to the data on the disk, I'll denote it in two different forms. When I say the first 11 bytes contain the filename and extension, I mean that the data is there in ASCII code. However, when I say the File Type flag byte will be 0, 1, 2 or 3, I am indicating the Hex value of that byte. When I later refer to the value of a byte in the Granule Allocation Table, I'll also be referring to its Hex value.

The letter 'A' in ASCII is represented as Hex 41. Most disk zappers offer the option of displaying a sector in either Hex or ASCII. The one published in the December RAINBOW had even more options (decimal and binary) for how to display the data from the disk. The best disk zappers use a technique to display at the same time both the ASCII value and the Hex value of at least a selected byte in the sector, if not some or all of the sector. Some disk zappers (*VIP*s, for example) display "screen code" values of the sector. This is vaguely like an ASCII display, but the data is represented somewhat differently. With such zappers, the ASCII/Hex options need to be used. In this article, when I make reference to text I'm talking ASCII, but when I specify to numeric information I am talking Hex.

Note 3: I will assume we are considering only normal Radio Shack/Microsoft Disk Extended BASIC files here. What follows is not relevant directly to OS-9, copy-protected material or to noncopy-protected, but also non standard format disks, such as some new Radio Shack games, the new Infocom games and *Graphicom* or *WEFAX* picture disks.

Note 4: I will assume you are familiar with the fundamental divisions of data on a disk: the 35 tracks and the 18 256-byte sectors that are standard for Radio Shack's Disk Operating System. The Radio Shack Disk Extended BASIC manual is quite clear on this matter. I will also assume you understand that a "granule" consists of nine sectors on the disk, thus is 2¼K in size and can occupy either the first or the last nine sectors (numbered 1 through 9 or 10 through 18) on a given track. Every track except Track 17 (the directory track) consists of two granules. The directory track is excluded from granule notation. Therefore, Track 0, Sector 1 is the first sector in granule number 0, Track 16, Sector 18 is the last sector in Granule 33 and Track 18, Sector 1 is in Granule 34.

Note 5: Most of the time I will use Hex notation, but sometimes I will use decimal. Thus, when I refer to Track 17, I am talking decimal. Whenever I use Hex, I'll specify it by writing the word "Hex" or by preceding the number with a dollar sign ($).

These notes may seem tedious, but hopefully they will help the novice get past some of the conventions that longtime hackers often accept and understand without thinking about them, yet are sometimes confusing to the newcomer.

Directory Structure

When you type DIR, you see a listing of filenames, extensions and then a number, a letter and another number. The first number displayed is the File Type, the letter is the ASCII flag and the last number is the number of granules in that file. Everything except for the information on how many granules there are in the file is directly recorded in the directory entry.

The directory starts on Track 17 (Hex $11), Sector 3. Each entry is 32 (Hex $20) bytes long, of which only the first 16 (Hex $10) bytes are used. The remaining 16 bytes are "reserved for future use" by Microsoft. Therefore, one sector can hold up to eight directory entries. If the directory has more than eight files on it, then more sectors (Track 17, sectors 4, 5, 6, etc.) are used. Let's look at Track 17, Sector 3.

Bytes $00 through $0A: Filename and Extension

As can be learned from reading Chapter 11 of the Radio Shack manual, filenames in the directory are placed on Track 17, Sector 3 and up. The first eight bytes of the entry are the filename, the next three are its extension. These 11 bytes normally contain ASCII characters. Files that were killed will have the first byte in their name changed to Hex 0.

Byte $0B: File Type Flag

The next one is the File Type flag. This byte equals 00 for tokenized BASIC

programs. It equals Hex 01 for what the Radio Shack manual calls "BASIC data files," or what you will encounter as "ASCII BASIC" files, or as with many word processors and editor/assemblers, ASCII text files. It equals Hex 02 for machine language programs. The manual says this byte equals Hex 03 for "text editor source files." This File Type is rarely encountered, except by users of *Color Scripsit*. To those, I suggest buying *Telewriter* and *Telepatch* or *VIP Writer* or *Elite Word*. You'll have a much better word processor and won't have to worry about text editor source files.

Byte $0C: ASCII Flag

The ASCII flag follows the File Type flag. This single byte is set to zero if the file is in binary format, and set to Hex FF if the file is in ASCII format. Tokenized BASIC is a kind of binary File Type; so is a machine language program. Thus, both of those tend to have the ASCII flag set to zero. ASCII text files (File Type flag = 1) have their ASCII flag set to Hex FF.

Byte $0D: First Granule

This is the number of the first granule used for the file.

Bytes $0E and $0F: Number of Bytes in Last Sector

Byte $0E is the high order byte, and is either zero or one — most of the time it is zero. If one, the next byte is zero, and 256 bytes in use (a full sector) are specified. Byte 15 varies from one through FF to signify from one through 255 bytes used in the last sector of the file. Bytes $10 through $1F are "reserved" from back in 1981 for "future use." They have never been used.

Note here that while bytes Hex 0D,0E and 0F provide some information about where the file resides on the disk and how far it extends, they leave out a lot! They give no clue, in and of themselves, as to how many granules there are in the file or of how many sectors are used in the last granule of the file. To find that out, you have to move over to Track 17, Sector 2, called the Granule Allocation Table, or GAT.

The GAT

The GAT occupies Track 17, Sector 2. Actually, only the first 68 bytes of Track 17, Sector 2 constitute the GAT. The disk manual incorrectly states that the remaining bytes in that sector will be zero. Anyone who's ever looked at a disk with a zapper knows this is not true. Indeed, due to some sloppy code in Disk BASIC, copies of pieces of the directory itself wind up in the space beyond the 68th byte of Track 17, Sector 2. This little idiosyncrasy had to be corrected by authors of Disk BASIC modifications who were implementing support for 40- and 80-track drives, but that's another story. Suffice it to say here that in a normal Disk BASIC disk, the first 68 bytes of Track 17, Sector 2 are the GAT and the remaining bytes are "garbage."

The first byte in the GAT is "byte number zero." Each byte in the GAT corresponds to the status of a given granule on the disk. That status is encoded as follows: If the GAT byte is equal to $FF, then the corresponding granule is avilable for new files. On a blank disk this is a blank (all $FF) granule; on an often-used disk, which has had files killed and other files written to it, that granule might contain some data from a previously killed file. In either case, the granule is flagged as available for new files.

If a byte in the GAT is equal to a number from zero through 67 ($0 through $43), it means that the granule is occupied by a given directory file, that this granule is not the last granule in the file, and the next granule in the file is the granule number corresponding to the number in that byte. As a result, if the directory entry says the first granule in a file is Granule $1E and Byte $1E in the GAT reads Hex 1F, that means granule $1F is the next granule in the file, and Byte $1F of the GAT must now be looked at to learn more about where the file resides.

If the byte in the GAT reads $C1 through $C9, it means the corresponding granule is the last granule in the file, and the number of sectors in the granule that actually belong to the file is the low order Hex digit of the number in the GAT byte. That is, if we look at Byte $1F from the example of the last paragraph and find it contains $C4, it means the file in question occupies a total of two granules, granules 1E and 1F, and Granule 1F actually has only the first four of its sectors used for the file (the remaining five would be wasted).

Note that the smallest file in Disk BASIC *must* occupy a whole granule, even if it is only one byte long. The rest of the granule in question is wasted. Note that if a GAT byte is equal to any number *besides* $FF, 00 through $43, or $C1 through $C9, it means the GAT itself has an error in it! The Disk BASIC manual alleges that $C0 is a valid code for a GAT byte, but I can see no use for that value. (If a reader can explain to me the significance of a $C0 GAT byte value, I'd appreciate it.) For now, I can only assume the Disk BASIC manual is in error on this matter.

To fully know exactly where the file ends, we now need to hop back to look at the directory entry for the number of bytes in the last sector of the file that

Sample Run

```
* INDICATES A KILLED FILE WHEN IN FRONT OF FILE NAME
* INDICATES INVALID GAT ENTRY IN GAT LISTING
KILLED FILES THAT ARE NOT LISTED AS 'SCRUNCHED GAT' HAVE MOST LIKELY
BEEN WRITTEN OVER BY A NEW FILE AND ARE REALLY LOST!


DIRECT1 /BAK BASIC data file     ASCII
00,01,02,03,04,05,06
 2  = # OF SECTORS  IN LAST GRAN
2 = # OF BYTES IN LAST SECTOR

DIRECT2 /BAK BASIC data file     ASCII
07,08,09,0A
 6  = # OF SECTORS  IN LAST GRAN
3B = # OF BYTES IN LAST SECTOR

DISKANAL/BAK BASIC program       BINARY
0B
 9  = # OF SECTORS  IN LAST GRAN
FF = # OF BYTES IN LAST SECTOR

DIRECT1 /TXT BASIC data file     ASCII
0C,0D,0E,0F,10,11,12
 2  = # OF SECTORS  IN LAST GRAN
2 = # OF BYTES IN LAST SECTOR

DIRECT2 /TXT BASIC data file     ASCII
13,14,15,16
 6  = # OF SECTORS  IN LAST GRAN
3B = # OF BYTES IN LAST SECTOR

DISKANAL/BAS BASIC program       BINARY
17
 9  = # OF SECTORS  IN LAST GRAN
FF = # OF BYTES IN LAST SECTOR
```

are actually used. You now can see that the specification of what bytes on the disk corrrespond to a given file is smeared out between the directory entry for that file and the GAT. In the GAT, the file size has to be determined by tracking down the file from GAT byte to GAT byte, until the end of what programmers call the "linked list" of bytes is reached. And finally, after finding the last granule and last sector in that granule, one has to go back to the directory entry to find where the last used byte is in the last sector.

What a mess! Why did they do it that way? Despite the mess, there is some method to this madness. Grouping all the information concerning which granules are used and which are not into one single block, they facilitate keeping track of available space on the disk and make killing of old files easier to do. Though there are some ways in which the scheme is needlessly complex, it actually makes more sense than it seems to upon first glance after you start considering how disk operating systems and file managers have to be written.

Killed Files

When you kill a file using Disk BASIC, the actual file data is not immediately destroyed. What happens is the first byte in the name of the file in the directory is set to zero, and all bytes in the Granule Allocation Table, which correspond to bytes in that file, are set to $FF (= available). Thus, if you kill a file, all data in the file remains on the disk. Only the information in the GAT needed to find such data is destroyed by the KILL command itself.

Of course, if you try to SAVE any new data to a disk after killing a file, you may end up writing over granules that were previously a part of the killed file, or even writing over the old killed directory entry as well. At that point, the file data in the killed file is completely destroyed. But, if you have merely killed a file and then want to restore it, such restoration is possible, though often tedious.

As you add files to a fresh, formatted disk, Disk BASIC is inclined to assign granules to each new file in a fashion that starts on one side of the dirctory and tends to alternate on either side of the directory track. Therefore, files under Disk BASIC tend to get assigned near the middle of the disk and grow towards both the center and the outer edge.

However, some disk utilities (such as Spectrum Projects' *Directory Utiltity*) assign granules sequentially from Granule 0 to Granule 68; disks that have had many files written to them, then erased, then others written to them, tend to have the granules that compose a given file scattered all over the disk. This can make reconstruction of a big killed file on such a disk very difficult.

File Structure

Now that you know how to find a given file, from its first to last byte on the disk, I'll explain what you can expect to see in the three most commonly encountered Disk BASIC files.

ASCII

ASCII text files ("BASIC text files") are the easiest of all to understand. These files have the File Type flag set to one and the ASCII flag set to $FF. They are almost totally "raw" data — just byte after byte of information, usually (though not necessarily) ASCII text. The only thing special about them is the last byte in the file is Hex 1A (control Z). This is the flag that marks the end of an ASCII text file. Within the file the bytes are typically less than a Hex value of $80, but are not required to be so. Thus, the only special "structure" such a file has is that it will not have any $1A's in it until the last byte of the file.

Tokenized BASIC

Tokenized BASIC files are a kind of binary file. They have a File Type flag of zero and an ASCII flag of zero as well. Looking at them in ASCII, you will be able to recognize all the text that is in the BASIC program, but all BASIC key words are encoded ("tokenized") into one or two bytes. Line numbers do not appear as ASCII, but as two Hex bytes.

For example, the line 257 PRINT "ABC" appears in the file as the following sequence of bytes: 00 (a line delimeter), 01 01 (the two-byte Hex value for 257 decimal) followed by Hex 80 (the BASIC token code for PRINT), then Hex 22 41 42 43 22, the ASCII codes for "ABC". Because no BASIC token is set to 00, and 00 is a nul (not used to encode ASCII letters and symbols), you will never find inside the tokenized BASIC file more than two 00 bytes in a row.

However, at the end of the file, you will find three 00 bytes. This is BASIC's "end of file flag." If you are in the process of reconstructing a disk after losing the GAT on it (an utterly thankless task . . . let me tell you!), your reconstruction of a given BASIC file is aided by your search for the sector with BASIC code and three 00 bytes.

Occasionally, you might encounter what appears to be a normal BASIC file that has two sets of three 00 bytes in it. This most likely is an especially prepared "end packed" BASIC file, made up by programmers to stuff machine language code invisibly at the end of a BASIC file. Such files are not normal BASIC files and have been "foxed with" by the programmer.

Machine Language Files

These are by far the the most complicated files of all. This is due to the provision Microsoft made for "segmented" binary files. That is, an ML file on a disk (unlike its counterpart on a tape) can consist of several segments that load in different areas of memory. Let's start with the description of a non-segmented ML file, then go on from there.

Non-segmented ML Files

The SAVEM command generates non-segmented ML files. Note that the SAVEM command cannot generate a segmented ML file; those are created using various editor/assemblers or by foxing with the file as it resides on the disk using a disk zapper. Such non-segmented ML files (actually they are segmented files that have only one segment) begin with a 00 byte. This is followed by two bytes that specify number of ML data bytes, then by two more bytes that specify where the ML data is to start loading into memory. This five-byte "header" is followed by the ML data itself. At the end of the file is a five-byte ending sequence, consisting of an $FF byte, two bytes of 00 each, then two bytes that specify the execute address of the file.

For example, if you made a file using SAVEM "TEST" &H4321, &H4324, &H4322, and if $4321 through $4324 contain the Hex values A1, B2, C3 and D4 at the time you save the file, it appears on the disk (in Hex) as follows: 00 00 04 43 21 (the five-byte header with the 00 flag byte, the length of the file as $0004 and the start address of $4321), followed by A1, B2, C3, D4 (the actual data in the file itself), followed by the end five bytes of FF, 00, 00, 43 and 22.

Note carefully that the end address is not specifically stored as such on the disk in the file header. It must be calculated from the start of load address and the file length. Also note that you must track the file down to its end before you can tell what its execute address is.

Segmented ML Files

Segmented ML files are very similar, but after the first segment, instead of

having an FF 00 00 (execute address) five-byte end flag, they have another header, specifying more data to be loaded elsewhere in memory. There is no limit (other than the memory of the CoCo and the size of the disk) on how many segments such a file can have, so it is possible to create an ML file that loads single bytes all over the memory of the CoCo. In these segmented files the end is recognized by the presence of the FF end flag followed by the 00 00 (execute address) five-byte final block. Thus, a segmented ML file can have lots of start and end addresses, though it can only have one execute address.

This segmented structure can be a bit confusing, but it is very convenient for assemblers! And, it is helpful when you need to make a program that loads stuff in differing and widely separated areas of memory. Such segmented files are easily created with *EDTASM* and with *Macro 80C* (and probably most other editor/assembler packages) using more than one ORG statement in the source code. Indeed, some assemblers that assemble directly to disk, like *Macro 80C*, create segmented files even when assembling source code that is not multiple ORGed. In such cases, the end address of one segment will be seen to be one less than the start address of the next.

References

Disk BASIC Unravelled, published by Spectral Associates, is the bible for ML disk hackers. Indeed, I'd go so far as to say that if you do any assembly language hacking using Disk BASIC, you need to buy the three-volume *BASIC Unravelled* (cost is about $50). This set has fully commented disassemblies of all versions of the CoCo ROMs, gobs of information about the RAM-base page and such, stuff on file formats, BASIC routine entry points, and the like. It is what Microsoft and Tandy should have published the day they released the CoCo on the market. Spectral Associates and the unnamed ML hacker(s) who compiled this set deserve the thanks of all CoCo users.

The *Disk Extended BASIC Manual* comes with your Radio Shack disk Drive 0. This manual can be ordered separately, although only the few pages in Chapter 11 are of relevance to what is written here.

The Disk Analyzer Program

The following simple BASIC program automatically searches out all the information needed to find every byte in a given valid file on a Disk BASIC disk. It dumps that information to a printer; you can also have it go to the screen by changing Line 50 from D=-2 to D=0. If you do this, you'll want to add some kind of pause feature as the data otherwise scrolls by too quickly to read. Just load the program, type RUN, put the disk you want to analyze in Drive 0, make your printer ready and press any key. All text is printed in ASCII characters and all numeric values are printed in Hex.

Analyzer prints four lines of information about each file on your disk.

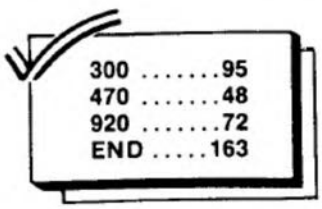
First line: Filename, extension, File Type flag byte status, ASCII flag status (an asterisk [*] precedes any killed files on your disk that this *Analyzer* will see and list).

Second line: The numbers of all the granules that compose the file, from the first to the last. If invalid granule numbers are detected, the program indicates this by a '*' and/or by printing in the next line "scrunched GAT!".

Third line: The number of sectors in the last granule.

Fourth line: The number of bytes in the last sector.

This program gathers together all the widely separated data into one table for you to refer to when you are wandering around your disk using a disk zapper. Note that *Analyzer* does some testing for messed up entries, but on a disk with blown directory entries it won't be of much use. □



The listing: ANALYZER

```
1Ø CLEAR 2ØØØ
2Ø DIM G(69)
25 CLS:PRINT"(C) MARTY GOODMAN 1
985":PRINT
3Ø PRINT"DIRECTORY ANALYZER":PRI
NT:PRINT"PREPARE PRINTER":PRINT:
PRINT"HIT ANY KEY TO CONTINUE"
4Ø IF INKEY$="" THEN GOTO 4Ø
5Ø D=-2
7Ø PRINT#D,"* INDICATES A KILLED
 FILE WHEN IN FRONT OF FILE NAME
"
8Ø PRINT#D,"* INDICATES INVALID
GAT ENTRY IN GAT LISTING"
9Ø PRINT#D,"KILLED FILES THAT AR
E NOT LISTED AS 'SCRUNCHED GAT'
HAVE MOST LIKELY"
92 PRINT#D,"BEEN WRITTEN OVER BY
 A NEW FILE AND ARE REALLY LOST!
"
95 PRINT#D,"":PRINT#D,""
1ØØ REM READ IN GAT
11Ø DSKI$ Ø,17,2,A$,B$
12Ø FOR N=1 TO 68
13Ø G(N)=ASC(MID$(A$,N,1))
14Ø NEXT N
2ØØ REM ANALYZE DIRECTORY
21Ø S=3
25Ø DSKI$ Ø,17,S,A$,B$
3ØØ REM MAIN LOOP
3Ø5 IF LEN(A$)=Ø THEN :GOTO 5ØØØ
3Ø7 K=Ø
31Ø T=1:GOSUB 2ØØØ:IF E$=CHR$(&H
FF) THEN GOTO 9ØØØ
32Ø IF ASC(E$)<>Ø THEN GOTO 35Ø
325 K=1
335 PRINT#D,"* ";
35Ø T=8:GOSUB 1ØØØ:PRINT#D,E$;:P
RINT#D,"/";
36Ø T=3:GOSUB 1ØØØ:PRINT#D,E$;
37Ø T=1:GOSUB 1ØØØ
375 E=ASC(E$)
38Ø IF E=Ø THEN GOTO 41Ø
385 IF E=1 THEN GOTO 42Ø
39Ø IF E=2 THEN GOTO 43Ø
395 IF E=3 THEN GOTO 44Ø
4ØØ PRINT#D," BAD FLAG BYTE  ";:
GOTO 45Ø
41Ø PRINT#D," BASIC program  ";:
GOTO 45Ø
42Ø PRINT#D," BASIC data file";:
GOTO 45Ø
43Ø PRINT#D," Mach Lang progm";:
GOTO 45Ø
44Ø PRINT#D," Txt Ed src file";
45Ø PRINT#D,"    ";:T=1:GOSUB 1ØØ
Ø:E=ASC(E$)
46Ø IF E=Ø THEN GOTO 48Ø
465 IF E=255 THEN GOTO 49Ø
47Ø PRINT#D," *BAD* ":GOTO 5ØØ
48Ø PRINT#D," BINARY":GOTO 5ØØ
49Ø PRINT#D," ASCII "
5ØØ REM ANALYZE GAT ENTRIES
51Ø T=1:GOSUB 1ØØØ:E=ASC(E$)
515 COMMA=Ø
52Ø REM GAT LOOP
525 GOSUB 3ØØØ
53Ø IF V=Ø THEN PRINT#D," *":GOT
O 89Ø
54Ø IF V=2 THEN GOTO 8ØØ
545 IF COMMA=Ø THEN GOTO 55Ø
547 PRINT#D,",";
55Ø G$="ØØØØ":G$=G$+HEX$(E):G$=R
IGHT$(G$,2):PRINT#D,G$;
56Ø COMMA =1
57Ø E=G(E+1):GOTO 52Ø
8ØØ REM CHECK SECTORS USED IN
LAST GRAN
81Ø PRINT#D,"":PRINT#D,E AND &HØ
F;" = # OF SECTORS  IN LAST GRAN
"
82Ø GOTO 9ØØ
89Ø PRINT#D,"SCRUNCHED GAT!"
9ØØ REM CHECK BYTE COUNT IN LAST
 SECTOR AND LOOP
91Ø GOSUB 1ØØØ
915 B=ASC(E$):IF B>1 THEN GOTO 9
8Ø
92Ø T=1:GOSUB 1ØØØ
925 BC=256*B+ASC(E$):IF BC>256 T
HEN GOTO 98Ø
93Ø PRINT#D,HEX$(BC);" = # OF BY
TES IN LAST SECTOR"
94Ø T=16:GOSUB 1ØØØ
945 PRINT#D,""
95Ø GOTO 3ØØ
98Ø PRINT#D,"BAD BYTE COUNT ENTR
Y!":GOTO 94Ø
1ØØØ REM TRIM OFF LEFT AND GET
SUBSTRING
1ØØ5 E$=LEFT$(A$,T)
1Ø1Ø A$=RIGHT$(A$,LEN(A$)-T):RET
URN
```

continued on Page 41

VARLIST:

A Quick and Easy Way to List Program Variables

By Hans Schulz

Varlist, a utility program that lists all variables, will lend some help with those pesky problem programs that you just can't get to work properly.

Let's look at an example. I once had a program with a variable called LNC (for "Line Count"); it also had another variable named LND (for "Line Description"). The program just didn't work as anticipated. Of course, in hindsight I realized the error of my ways: Only the first two characters are significant in a variable name. The program treated LNC and LND as the same variable!

Now, if I only had a utility tool that could tell me all the lines containing LNC and LND. I didn't have such a tool, and it took me an agonizingly long time to go through every line of the program to change one of the variable names so it was different from the other.

I decided then that someday I would write such a utility program. *Varlist* is the result, and I would like to share it with other CoCo friends. It lists all the numbers of the lines in which variables appear; in fact, if a variable appears twice, it is listed twice. In addition, it also highlights all the jump statements, i.e., GOTO, GOSUB, THEN and ELSE.

Do You Like BASIC's Beauty?

This program is written entirely in BASIC. As you can imagine, it does not exactly race through the target program, but it lets you know where it is at all times. In the interest of preserving some processing speed, I have kept the REMarks to a bare minimum and have eliminated unnecessary spaces wherever possible. A program line without spaces between BASIC statements and variable names may look strange at first sight, but you will get the drift of it.

Are You in a Hurry?

POKE 65495,0 will increase your processing speed, but if you are really impatient, insert lines 50001 and 50601 into *Varlist* for additional speed (see Listing 2). However, you lose the screen display during this speed up.

Do You have Enough Memory?

Varlist uses approximately 9,000 bytes (9K) of RAM. The program changes the standard PCLEAR4 when you first turn on the CoCo to PCLEAR1, which only reserves one page of graphics memory (1,536 bytes). If you have a lengthy target program you may have to free up some additional memory to fit both *Varlist* and your program into the available RAM space. To get at the extra 1,536 bytes, you have to perform the equivalent of a PCLEAR0, which, as you may know, is not a valid BASIC command. It can be summarized as follows: When you first power up your CoCo, type POKE &H19,6:NEW and press ENTER. In this case, you should also remove the PCLEAR1 statement from Line 50010.

How do You get the Program to Work?

First key in *Varlist* and CSAVE a copy of it, then make sure there are no typing errors by testing it with RUN 50000. It will list the variables in the test program (lines 10 through 90). Correct typing errors, if any, and CSAVE a corrected copy of *Varlist*. Now delete lines 10 through 90.

Make sure the program for which you want to produce the list of variables does not have any line numbers greater than 49999 and, if necessary, renumber it. Merge your program with *Varlist*. Now type RUN 50000 and press ENTER. The screen will display the line numbers of your program, which *Varlist* is scrutinizing as it steps through the program line by line.

How does *Varlist* Work?

Line 50010 reserves 1,500 bytes of memory for string variables and reserves space for 500 variable names and 500 line numbers; it also releases three pages of graphics memory. (When you turn on your CoCo it automatically PCLEARs four graphics pages.) Then the screen is cleared.

Line 50030 finds the starting address of your BASIC program in memory, regardless of whether you have a 16K or 32K CoCo. (This may be useful for future reference.)

Line 50040 initializes the variable PO (the pointer address of the beginning of the next line) and variable LI (the current line number being worked on). Line 50570, processed in the GOSUB statement, displays the line number being examined on the screen. If the line number is greater than 49999 then the program has reached *Varlist* and the end of your target program, in which case all variables have been found and the list will be displayed on the screen starting at Line 50600.

As an aid to the general understanding of *Varlist*, I let it generate a list of the variables used in the *Varlist* program (see Sample Output, Table 1). I have also prepared a shortened list of the variables, where each variable appears only once, and have sorted the variables in alphabetical order (see Table 2).

The GOSUB 50520 in Line 50060 reads the next character (in ASCII format) from the memory location where your

Table 1
Sample Output:
Variables and Jumps in *Varlist*

```
5ØØ1Ø   LA$(SUB)
5ØØ1Ø   LR(SUB)
5ØØ3Ø   N
5ØØ4Ø   gosub 5Ø55Ø
5ØØ4Ø   goto 5ØØ9Ø
5ØØ6Ø   gosub 5Ø52Ø
5ØØ7Ø   C
5ØØ7Ø   N
5ØØ7Ø   N
5ØØ7Ø   gosub 5Ø55Ø
5ØØ8Ø   gosub 5Ø52Ø
5ØØ8Ø   C
5ØØ8Ø   then 5ØØ8Ø
5ØØ9Ø   N
5ØØ9Ø   PO
5ØØ9Ø   C
5ØØ9Ø   PO
5ØØ9Ø   N
5ØØ9Ø   N
5ØØ9Ø   goto 5ØØ7Ø
5Ø1ØØ   C
5Ø1ØØ   N
5Ø1ØØ   then 5ØØ8Ø
5Ø11Ø   C
5Ø11Ø   N
5Ø11Ø   then 5ØØ8Ø
5Ø12Ø   C
5Ø12Ø   N
5Ø12Ø   then 5ØØ8Ø
5Ø13Ø   C
5Ø13Ø   N
5Ø13Ø   then 5ØØ8Ø
5Ø14Ø   C
5Ø14Ø   N
5Ø14Ø   then 5ØØ8Ø
5Ø15Ø   C
5Ø15Ø   C
5Ø15Ø   C
5Ø15Ø   C
5Ø15Ø   C
5Ø15Ø   thengosub 5Ø54Ø
5Ø15Ø   goto 5ØØ8Ø
5Ø16Ø   C
5Ø16Ø   N
5Ø16Ø   D$
5Ø16Ø   D$
5Ø16Ø   gosub 5Ø41Ø
5Ø16Ø   goto 5ØØ9Ø
5Ø17Ø   C
5Ø17Ø   N
5Ø17Ø   D$
5Ø17Ø   D$
5Ø17Ø   gosub 5Ø41Ø
5Ø17Ø   goto 5ØØ9Ø
5Ø18Ø   C
5Ø18Ø   D$
5Ø18Ø   gosub 5Ø41Ø
5Ø18Ø   goto 5ØØ9Ø
5Ø19Ø   C
5Ø19Ø   N
```

program is stored.

Line 50070 is reused again later, and if 'C' (the character being examined) is a zero, it indicates that the last byte of the program line has been reached. In that case, 'N' (the number of the storage location being read) must be decremented by one to update the address pointer (PO) of the next line in the GOSUB 50550.

In Line 50080 the next character is read (GOSUB 50520). If the character ('C') is a space, then the process is repeated until a non-blank character is found.

If in Line 50090 the address pointer of the next line is identified, then PO is updated again.

Lines 50100 through 50140 identify some BASIC statements with two-byte tokens and processing continues with reading the next character in Line 50080. Line 50100 identifies a PEEK; Line 50110, a USR statement; Line 50120, an ABS function; Line 50130, an ATN function; and Line 50140, an SQR function.

Line 50150 determines when processing can skip to the next line without reading to the end of the current line. That can be done when a DATA statement is encountered (token 134) or a REM (token 130), or its equivalent, the apostrophe ('), which is tokenized as 131, and also the LLIST statement (token 155) and LIST (token 148).

Table 2
Short List of Variables
in Alphabetical Order

```
AA$
C
D$
I
I$
IM
IX
LA
LA$(SUB)
LI
LL
LR(SUB)
MN$
MX$
N
PL
PO
Q
R
S
T
Z
```

GOSUB 50540 accomplishes the skip to the next line, after which processing continues at 50080 by reading the next character.

Lines 50160 through 50200 are somewhat self-explanatory: They deal with the jump statements (GOTO, GOSUB, THEN and ELSE). Only Line 50190 seems a little out of place — it identifies the two-byte token for RND (255 132). The RND token is not of significance to the *Varlist* logic and the program reads the next character by branching back to 50080. Having disposed of the RND (255 132) token, it can now be deduced in Line 50200 that if the current character is a token 132, it is part of the two-byte token (58 132) representing the statement ELSE. (Does 58 seem familiar? It is the ASCII code for the colon [:], which is used by BASIC to separate statements on the same line. Quite clever, those Microsoft people, using the colon as part of the ELSE logic!)

Now, back to the jump statements. In each case a descriptor (D$) is being built. This string may contain, for example, the word "then" or "else" and may conceivably have the word "goto" or "gosub" added to it. At this point, a subroutine is performed (GOSUB 50410) that obtains the line reference number, i.e., the line number to which the jump statement has been programmed to jump. Upon return from the subroutine, with the next byte already read, processing branches back to 50090 to determine what to do with this character.

Line 50210 looks at the letter 'M' (ASCII code = 77). This is not an ordinary 'M' though, such as an 'M' that may be part of a variable name. It is the 'M' in CLOADM. The token for CLOAD is 151, and if the byte following it is the letter 'M', then GOSUB 50520 reads it and, immediately afterwards, branches back to 50080 to read the next character. At this point Line 50220 discards any further tokenized BASIC statements, i.e., ASCII codes greater than 127, and branches back to 50080.

Line 50230 finds out if the character read is alphabetic, i.e., if the ASCII code is in the range from 65 to 90, representing the capital letters A to Z. Finally, the program does some real work after all the sifting and discarding up to this point: The subroutine at 50300 assembles the variable name, starting with the character ('C') just read, then adds to it, one byte at a time, until the variable name is completely assembled. After return from the subroutine, *Varlist* branches back again to 50080 to read the next character.

Line 50240 tests to see whether the character read is a quote ("), which is

represented by ASCII code 34. Anything enclosed in quotes is of no interest in this program. Therefore, the subroutine at Line 50270 keeps on reading and discarding characters until it finds the second of a pair of quotes. The main body of this program ends at Line 50250, where processing loops back to read the next character at 50080.

What do the Subroutines do?

Varlist contains the following subroutines, which are each described here.

1) Skip between Quotes — Line 50260
2) Build the Variable Name — Line 50300
3) Build the Line Number Reference — Line 50400
4) Peek at the next ASCII Character — Line 50510
5) Skip to the next Line — Line 50530
6) Print the List of Variables — Line 50590

Skip between Quotes — Line 50260

As described earlier, this subroutine keeps reading and discarding characters until it finds the second of a pair of quotes (ASCII code 34). The subroutine also checks for reasonable length of the string between the pair of quotes. I felt anything in excess of five lines of 32 characters (a total of 160 characters) is probably in error and designed the program to stop in such a case. If this does not apply in your program, simply change Line 50280 accordingly.

Build the Variable Name — Line 50300

When powering up CoCo the subscript (LA) used to identify the labeled variable (LA$) has a value of zero. On each trip through the subroutine, that is, every time a new variable name is stored, the subscript is incremented by one in Line 50310. In Line 50320 the first character of the variable is stored; in 50330 the next character is read.

In Line 50340 the character value of zero indicates the end of the current line has been reached and it is now time to store the current line number (LI) in the array LR(LA). This array is used for later printing to indicate where each variable appears in the target program. The program then branches back to the beginning of the main routine of the program.

Line 50350 tests to see if the byte currently under scrutiny is numeric (ASCII codes 48 to 57) or if it is alphabetic (ASCII codes 65 to 90). If it is alphanumeric, the byte is appended to the array LA$(LA) and processing loops back within the subroutine to 50330 to read the next byte. If the character being examined in Line 50360 is a '$' sign (ASCII code = 36), it is added to the variable name and processing resumes at 50330, getting the next byte.

In Line 50370, if the character is an ASCII code 40, i.e., the opening bracket '(', then the literal "(SUB)" is appended to the variable name to show that the variable is subscripted. In other words, the variable is an array.

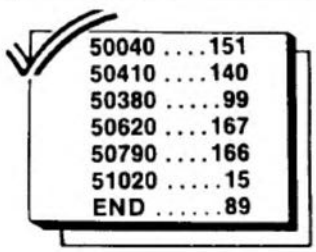
Line 50380 stores the current line number being worked on (and presently being held in "LI") in the Line Reference array, "LR(LA)."

Build the Line Number Reference — Line 50400

In Line 50410, which is similar to Line 50310, the subscript LA is incremented by one to store the Line Reference on each pass through the subroutine.

In Line 50420 the line number LI presently being processed is stored in the Line Reference array, "LR(LA)."

Line 50430 obtains the next character, and if it happens to be a blank (ASCII = 32), the program immediately gets the next byte.



Listing 1: VARLIST

```
1Ø REM**LINES 1Ø-9Ø REPRESENT A
TEST PROGRAM FOR DEMONSTRATION.
15 FORO=1TOP:IFQ=RT THENGOSUB123
456789ELSE1Ø
2Ø S3=T3+U:V(W)=X3
25 NEXT Y:  REM Q$
3Ø IFZ=5 THEN1ØELSEGOTO13498
35 A3(56)=1:B345=2:C4444=3
4Ø IF PEEK(DT)=4 THEN POKE ER,1
45 '!!!
5Ø IFF=3THENG=9ELSEGOTO1Ø
55 IF H=I THENGOSUB1Ø:GOTO 1Ø 'R
EAD & END
6Ø PRINT "A=";J;"   A3=";K3;"
";L(M);"=ARRAY R":RETURN
65 READ N$:END
7Ø DATA A,A3,B
9Ø LIST
5ØØØØ '************************
      *        VARLIST        *
      *(LIST OF ALL VARIABLES)*
      *FOR THE COLOR COMPUTER *
      * BY HANS SCHULZ (1984) *
      ************************
5ØØ1Ø CLEAR15ØØ:PCLEAR1:DIM LA$(
5ØØ),LR(5ØØ):CLS
5ØØ2Ø '***INITIALIZE***
5ØØ3Ø N=PEEK(25)*256+PEEK(26)-1
5ØØ4Ø GOSUB5Ø55Ø:GOTO5ØØ9Ø
5ØØ5Ø '***READ THE FILE***
5ØØ6Ø GOSUB5Ø52Ø
5ØØ7Ø IFC=ØTHENN=N-1:GOSUB5Ø55Ø
5ØØ8Ø GOSUB5Ø52Ø:IFC=32THEN5ØØ8Ø
5ØØ9Ø IFN=PO ANDC=ØTHENPO=PEEK(N
+1)+PEEK(N)*256:GOTO5ØØ7Ø
5Ø1ØØ IF(C=134ANDPEEK(N-2)=255)T
HEN5ØØ8Ø
5Ø11Ø IF(C=131ANDPEEK(N-2)=255)T
HEN5ØØ8Ø
5Ø12Ø IF(C=13ØANDPEEK(N-2)=255)T
HEN5ØØ8Ø
5Ø13Ø IF(C=148ANDPEEK(N-2)=255)T
HEN5ØØ8Ø
5Ø14Ø IF(C=155ANDPEEK(N-2)=255)T
HEN5ØØ8Ø
5Ø15Ø IFC=134ORC=13ØORC=131ORC=1
55ORC=148THENGOSUB5Ø54Ø:GOTO5ØØ8
Ø
5Ø16Ø IF(C=165ANDPEEK(N-2)=129)T
HEND$=D$+"goto":GOSUB5Ø41Ø:GOTO5
ØØ9Ø
5Ø17Ø IF(C=166ANDPEEK(N-2)=129)T
HEND$=D$+"gosub":GOSUB5Ø41Ø:GOTO
5ØØ9Ø
5Ø18Ø IFC=167THEND$="then":GOSUB
5Ø41Ø:GOTO5ØØ9Ø
5Ø19Ø IF(C=132ANDPEEK(N-2)=255)T
HEN5ØØ8Ø
5Ø2ØØ IFC=132THEND$="else":GOSUB
5Ø41Ø:GOTO5ØØ9Ø
5Ø21Ø IF(C=151ANDPEEK(N)=77)THEN
GOSUB5Ø52Ø:GOTO5ØØ8Ø
5Ø22Ø IFC>127THEN5ØØ8Ø
5Ø23Ø IF(C>64ANDC<91)THENGOSUB5Ø
31Ø:GOTO5ØØ8Ø
5Ø24Ø IFC=34THENGOSUB5Ø27Ø:Q=Ø:G
OTO5ØØ8Ø
5Ø25Ø GOTO5ØØ8Ø
5Ø26Ø '***SKIP BETWEEN QUOTES***
5Ø27Ø GOSUB5Ø52Ø:Q=Q+1:IFC<>34TH
EN5Ø27Ø
5Ø28Ø IFQ>16ØTHENPRINT"CKECK FOR
 PAIRED QUOTES":STOP
5Ø29Ø RETURN
5Ø3ØØ '*** GET VARIABLE NAME ***
5Ø31Ø LA=LA+1
5Ø32Ø LA$(LA)=CHR$(C)
5Ø33Ø GOSUB5Ø52Ø
5Ø34Ø IFC=ØTHENLR(LA)=LI:GOTO5ØØ
7Ø
5Ø35Ø IF(C>47ANDC<58)OR(C>64ANDC
<91)THENLA$(LA)=LA$(LA)+CHR$(C):
GOTO5Ø33Ø
5Ø36Ø IFC=36THENLA$(LA)=LA$(LA)+
CHR$(C):GOTO5Ø33Ø
5Ø37Ø IFC=4ØTHENLA$(LA)=LA$(LA)+
"(SUB)"
5Ø38Ø LR(LA)=LI
5Ø39Ø RETURN
5Ø4ØØ '***GET LINE # REFERENCE**
5Ø41Ø LA=LA+1
5Ø42Ø LR(LA)=LI
5Ø43Ø GOSUB5Ø52Ø:IFC=32THEN5Ø43Ø
5Ø44Ø IFD$="then"ANDPEEK(N-1)>12
9THEND$=""
5Ø45Ø IFC>127THENLR(LA)=Ø:GOTO5Ø
Ø9Ø
5Ø46Ø IF(C<48ORC>57)THENLR(LA)=Ø
:D$="":GOSUB5Ø32Ø:GOTO5ØØ9Ø
5Ø47Ø LL=VAL(CHR$(C))
5Ø48Ø GOSUB5Ø52Ø:IF(C>47ANDC<58)
THENLL=LL*1Ø+VAL(CHR$(C)):GOTO5Ø
48Ø
5Ø49Ø LA$(LA)=D$+STR$(LL)
5Ø5ØØ D$="":RETURN
5Ø51Ø '***GET ASCII FOR NEXT C**
*
5Ø52Ø C=PEEK(N):N=N+1:RETURN
5Ø53Ø '***SKIP TO NEXT LINE ***
5Ø54Ø N=PO-1
5Ø55Ø PO=PEEK(N+2)+PEEK(N+1)*256
5Ø56Ø LI=PEEK(N+4)+PEEK(N+3)*256
5Ø57Ø IFLI>49999THEN5Ø6ØØELSEN=N
+5:PRINT"... LINE #";LI
5Ø58Ø RETURN
5Ø59Ø '***PRINT THE LIST***
5Ø6ØØ PRINT:PRINT"   PROCESSING
COMPLETE ...":PRINT
5Ø61Ø PL=1
5Ø62Ø IFLR(PL)>ØTHENPRINTUSING"#
####";LR(PL);:PRINT"  ";
5Ø63Ø IFLA$(PL)<>""THENPRINTLA$(
PL)
5Ø64Ø PL=PL+1
```

```
5Ø65Ø IFPL=FIX(PL/12)*12ANDPL<=L
A THENPRINT@471,"MORE ...":PRINT
:GOTO5Ø67Ø
5Ø66Ø IFPL<=LA THEN5Ø62ØELSE5Ø68
Ø
5Ø67Ø IFINKEY$<>""THEN5Ø62ØELSE5
Ø67Ø
5Ø68Ø PRINT"***END OF LIST***":P
RINT@481,"WHAT WOULD YOU LIKE NO
W? ..."
5Ø69Ø IFINKEY$<>""THEN5Ø7ØØELSE5
Ø69Ø
5Ø7ØØ CLS:PRINTTAB(25)"ENTER"
5Ø71Ø PRINTTAB(3)"PRINT ON YOUR
PRINTER..P"
5Ø72Ø PRINTTAB(3)"DISPLAY THE LI
ST ......D"
5Ø73Ø PRINTTAB(3)"SHORT LIST ...
.........S"
5Ø74Ø PRINTTAB(3)"END THE PROGRA
M .......E"
5Ø75Ø PRINTTAB(22)"==> ";
5Ø76Ø I$=INKEY$:IFI$=""THEN5Ø76Ø
5Ø77Ø PRINTI$;" <=="
5Ø78Ø IFI$="P"THEN5Ø83Ø
5Ø79Ø IFI$="D"THEN5Ø61Ø
5Ø8ØØ IFI$="E"THENEND
5Ø81Ø IFI$="S"THEN5Ø88Ø
5Ø82Ø GOTO5Ø7ØØ
5Ø83Ø PL=1
5Ø84Ø IFLR(PL)>ØTHENPRINT#-2,USI
NG"#####";LR(PL);:PRINT#-2,"  ";
5Ø85Ø IFLA$(PL)<>""THENPRINT#-2,
LA$(PL)
5Ø86Ø PL=PL+1
5Ø87Ø IFPL<=LA THEN5Ø84ØELSE5Ø68
Ø
5Ø88Ø '*** SHORT LIST ***
5Ø89Ø PRINT"ONE MOMENT PLEASE ..
.":PRINT"   short list":FORS=1TO
LA:FORT=1TOLA
5Ø9ØØ IFLA$(S)=""THENLA$(S)=" ":
GOTO5Ø93Ø
5Ø91Ø IFASC(LA$(S))>96THENLA$(S)
=" ":GOTO5Ø93Ø
5Ø92Ø IF LA$(S)=LA$(T)ANDS<>T TH
ENLA$(S)=" "
5Ø93Ø NEXTT:NEXTS
5Ø94Ø FORR=1TOLA
5Ø95Ø IFLA$(R)<>" "THENPRINTLA$(
R);":";:Z=Z+1:LA$(Z)=LA$(R)
5Ø96Ø NEXTR
5Ø99Ø '***ALPHASORT THE LIST***
51ØØØ N=Z:S=1
51Ø1Ø PRINT:PRINT:PRINT"      NO
W SORTING ..."
51Ø2Ø MN$=LA$(S):IM=S:MX$=MN$:IX
=S
51Ø3Ø FORI=S TO N
51Ø4Ø IFLA$(I)>MX$ THENMX$=LA$(I
):IX=I
51Ø5Ø IFLA$(I)<MN$ THENMN$=LA$(I
):IM=I
51Ø6Ø NEXT
51Ø7Ø IFIM=N THENIM=IX
51Ø8Ø AA$=LA$(N):LA$(N)=LA$(IX):
LA$(IX)=AA$:N=N-1
51Ø9Ø AA$=LA$(S):LA$(S)=LA$(IM):
LA$(IM)=AA$:S=S+1
511Ø0 IFN>S THEN51Ø2Ø
5111Ø FORI=1TOZ:PRINTLA$(I);"*";
:NEXT
5112Ø END
```

Listing 2: SPEEDUP

```
5ØØØ1 POKE65497,Ø'HIGH SPEED
5Ø6Ø1 POKE65496,Ø:POKE65494,Ø:SO
UND128,2Ø:SOUND128,2Ø 'RESETTING
 TO NORMAL SPEED
```

continued from Page 37

```
2ØØØ REM GET LEFT STRING
2Ø1Ø E$=LEFT$(A$,1):RETURN
3ØØØ REM CHECK FOR VALID GAT ENT
RY
3ØØ5 V=1
3Ø1Ø IF E<68 THEN RETURN
3Ø2Ø IF E>&HBF GOTO 3Ø4Ø
3Ø3Ø V=Ø:RETURN
3Ø4Ø IF E>&HC9 THEN GOTO 3Ø3Ø
3Ø5Ø V=2:RETURN
5ØØØ REM CHECK FOR MORE DATA IN
B$ OR IN NEXT SECTOR
5Ø1Ø IF B$<>"" THEN A$=B$:B$="":
GOTO 3ØØ
5Ø2Ø S=S+1:GOTO 25Ø
9ØØØ REM ALL DONE
9Ø1Ø SOUND 1ØØ,2Ø:CLS:PRINT@27Ø,
"DONE"
9Ø2Ø END
1ØØØØ REM GAT DISPLAY
1ØØ1Ø PRINT#D,:PRINT#D,:PRINT#D,
"   GRANULE ALLOCATION TABLE LIS
TING"
```

continued from Page 31

```
325 DRAW"C3BM145,1Ø"+G$
33Ø DRAW"C2BM15Ø,1Ø"+H$
331 SOUND133,2
335 DRAW"C3BM15Ø,1Ø"+H$
34Ø DRAW"C2BM16Ø,1Ø"+D$
341 SOUND89,2
345 DRAW"C3BM16Ø,1Ø"+D$
35Ø DRAW"C2BM16Ø,5"+D$
351 SOUND 176,2
355 DRAW"C3BM16Ø,5"+D$
1ØØØ FOR Z=1 TO 1ØØØ:NEXT:GOTO1Ø
```

Listing 4: BATS

```
Ø '<BATS>
1 ' (C) 1985, J. KOLAR
1Ø PMODE3,1:PCLS3:SCREEN1,Ø
11 DRAW"C1BMØ,135R44D1ØR6U9ØR1ØD
3ØR5ØD5ØR4ØU4ØR1ØD4ØR1ØU9ØR4ØD1Ø
ØR1ØU1ØR1ØD1ØR1ØU4ØR2ØBD54L255"
12 PAINT(5,145),2,1
13 PAINT(5,19Ø),1,1
2Ø A$="F6E6"
23 D$="E6F6"
3Ø DRAW"C2BM=X;,=Y;"+A$
32 GOSUB1ØØ
35 DRAW"C3BM=X;,=Y;"+A$
4Ø DRAW"C2BM=X ;,=Y ;"+D$
42 GOSUB1ØØ
45 DRAW"C3BM=X;,=Y;"+D$
5Ø DRAW"C2BM=X;,=Y;"+A$
52 GOSUB1ØØ
55 DRAW"C3BM=X;,=Y;"+A$
6Ø DRAW"C2BM=X;,=Y;"+D$
62 GOSUB1ØØ
65 DRAW"C3BM=X;,=Y;"+D$
7Ø X=RND(25)*1Ø-1Ø:Y=RND(15)*1Ø
71 GOTO3Ø
1ØØ FOR Z=1TO1Ø:NEXT:RETURN
```

Hint . . .

Get the Sound Out

You can send sound from your 80C to any amplifier simply by soldering a couple of connections from the RF modulator.

Pin 3 from the RF modulator and any PC Board ground will give you audio output that you can send to any outside amplifier.

Incidentally, Pin 3 is the third pin back from the rear of the 80C on the RF modulator.

You should remember that opening the computer case will void your warranty.

Hint . . .

What's Your ROM Version?

With all the talk about new ROMs, you may be wondering exactly which ROM you have. If you have an older CoCo with Extended BASIC, just read the version number of your Extended BASIC at the top of the screen on power up. Then, to see which Color BASIC ROM you have, type EXEC 41175 and press ENTER.

If you have the new ROMs, Extended BASIC will be Version 1.1 and Color BASIC will be Version 1.2.

On the CoCo 2, Color BASIC will always be Version 1.2 or 1.3 (which are functionally identical).

The third installment of the "beginner's hardware course"

An Introduction to Timing

By Tony DiStefano

Continuing our journey into the CoCo, this month I will look into the heart of this and any computer — timing. All the hardware of the computer is controlled by timing. The most important part of the timing is to keep the CPU in step. What is a CPU, anyway? Well, the letters CPU stand for Central Processing Unit. The CPU inside the CoCo is the MC6809. The CPU, in a way, does all the work. It can move data from one part of memory to another, compare two values and act according to the result, add and subtract values and so forth. In fact, without the CPU, the rest of the hardware that makes up a computer would be worthless. The CPU is a very complex chip. It has data lines, address lines, interrupt lines, status lines and more. The timing that goes with the CPU is also important. OK, let's get into it. It is a prerequisite to understanding how a CPU works.

Up till now, when I talked about zeros and ones and the change from one to the other, it was considered to be instantaneous. There was no mention of how long it took to change from one state to another. In fact, we are dealing with real life, not just theory. Situations in theory rarely work in real life the way you want or expect them to. Welcome to the real world of delays. Ever caught an on-time airline flight? Ha!

The first fact of the real world is propagational delay. Take, for instance, a simple inverter. Figure 1a shows an inverter. When there is a '1' at the input there is a '0' at the output. A '0' input will give a '1' output. But when the input changes from one state to another, there is a short delay before the output changes. This delay is called the propagational delay, which means the amount of time it takes an electrical signal to go through a logic element or wire.

Figure 1b shows a graph of the input and the output of an inverter. The X-axis (from left to right) shows the passing of time. This can be in seconds, hundredths of seconds, thousandths of seconds and even millionths of seconds. When no time base is given, then time factor is not relevant. Typical delay times for the TTL family (more on chip families in later articles) is from five to 30 ns (ns = nanoseconds). The Y-axis usually shows the binary level of '0' and '1'. When two or more signals are shown that are related to each other, they are shown on top of each other with the left-to-right passing of time common to each.

Getting back to Figure 1b, we see the passing of time and the relation of the input to the output. There is no delay shown in this diagram. To show the delays of each signal for a given complex gate would confuse the diagram. Instead, an overall delay is given for the gate. But, in order to get used to the idea of delays, Figure 1c shows the time delays of a typical inverter. Along with the delay of the signal there is also the *rise* and *fall* time. The rise time of a signal is described as the time it takes for a given signal to reach 90 percent of maximum voltage from the 10 percent voltage level. The fall time of a signal is described as the time it takes for a given signal to drop to 10 percent voltage from the 90 percent voltage level. In the case of the CoCo, the voltage considered a logical level of '1'

Figure 1a

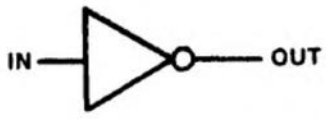


Figure 1b

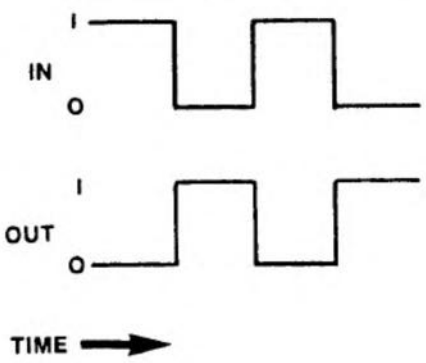


Figure 1c

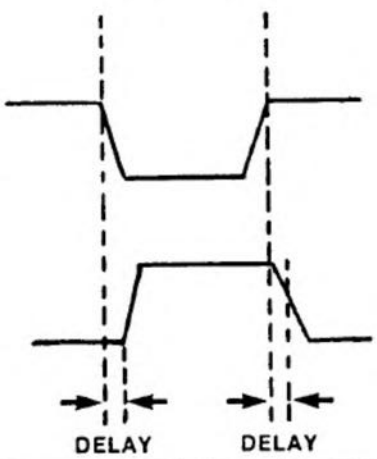


Figure 2a

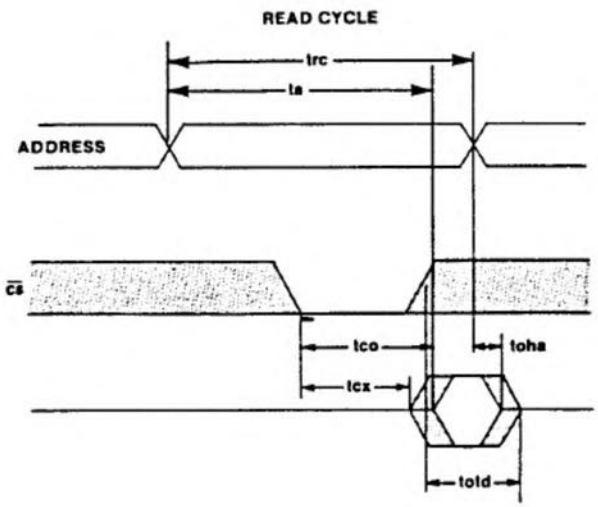
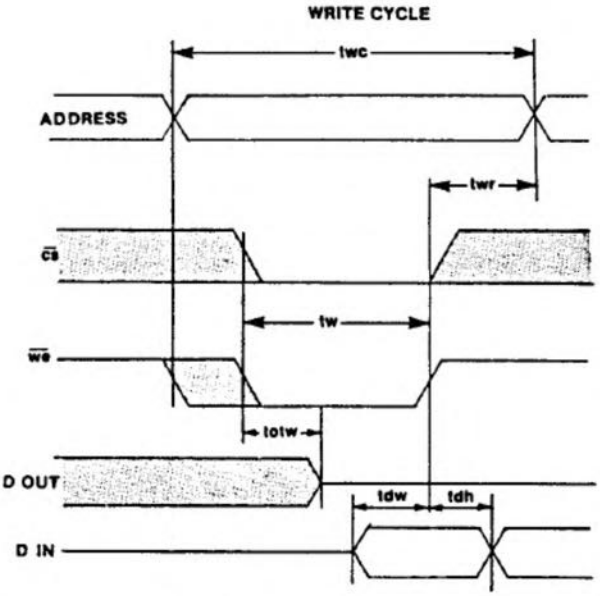


Figure 2b



(or HI) is five volts. The logical level '0' (or LO) is, of course, zero volts. The actual working voltages may be slightly different.

Delay, rise and fall times are important mainly to the designer of the system. When an engineer designs a computer he must know these timings and make sure that all operations are within the given limits. For example, two signals go to one gate, but one goes through several gates first. Each time the signal travels from one gate to another there is more delay. If the signal is delayed enough, an improper signal output results.

It sounds like I'm making a big deal of delays. While it is important, it is not a major concern to computer hackers (or should I use the term hobbyist?) and even less to end users. More important to us is another kind of delay. It is known as "access time," which means the mean time between the request for memory and the actual valid data.

Let us look at a typical memory chip. There are thousands of gates and transistors inside this chip. All of these gates inside the chip cause a significant delay between the time when the address to the chip is valid and the time when the data output appears on the data bus. This is known as access time. When talking about memory, an important parameter is access time. These access times can range from super-fast static memory at about 10 ns to very slow dynamic memory at 450 ns and slower. It is this limitation that controls and limits the speed of CPUs. Figure 2a shows the read cycle timing diagram of a memory chip. Figure 2b shows the write cycle for the same chip. What follows is a description of what each line on the diagram means.

Address — These are the address lines that select what byte is to be accessed. It is shown with two lines, one high and one low. It is shown this way because there are usually several lines and since the timing is the same no matter what byte you access, it is not relevant which address line is high or which line is low. The two lines (one on top and one on the bottom) represent any given address within the chip. Where the lines crisscross means a change of address. That is when the CPU is finished with that byte and requests another by putting another address on the bus. Access times are always measured with respect to the address change from the CPU. Actually, it starts when the address is stable, better known as a "valid address."

Chip Select (CS) — Remember the *CS line on memory chips in past articles? It is used to select or activate the chip. From the diagram of the read cycle, we can now see the relation between when the address is valid, the *CS line and when the data is valid.

Data out — This, of course, is the data that the CPU requested. Notice the data valid area. That is the time when the data that appears on the bus is the data that is held in that memory location. Notice the top and bottom dual line display. It has the same description as address lines, some are ones and some are zeros. The line in front of the data valid section is halfway between zero and one. That means the data lines are tri-state and no valid data is input or output. The shaded area on both sides of the data valid window is the transition time between tri-state and data valid. In this area, data lines are changing to their proper values. A read in this area will not yield valid data.

Read/Write — The *R/W line is used to select a read cycle or a write cycle. Straightforward, no problems there. In the CoCo this line is logical '1' to read and '0' to write.

The following is a description of all the relevant parameters used in Figures 2a and 2b.

t(rc) — Read Cycle Time: the time it takes for a complete read cycle given in ns.

t(a) — Access Time: the delay between a valid address and data valid.

t(co) — Chip Select to Output Valid: the delay between when the *CS is active and the data is valid. This is only true with a valid and stable address.

t(cx) — Chip Select to Output Active: same as t(co) but not to data valid; to when the data lines start changing from tri-state to output. Usually of minor importance.

t(otd) — Output Tri-state from Deselection: the time that the data stays valid after the *CE goes inactive or deselects.

t(oha) — Output Hold from Address Change: the time that the data stays valid after an address change is detected.

t(wc) — Write Cycle Time: same as the t(rc) except for a write cycle.

t(w) — Write Time: the minimum time the write line has to remain low.

t(wr) — Write Release Time: time between the *WE line deselects and a change of address.

t(otw) — Output Tri-state from Write: the time it takes the data lines to go to tri-state from a write request.

t(dw) — Data to Write Time Overlap: the time data must be stable before the *WE line deselescts.

t(dh) — Data Hold from Write Time: the time data must be stable after the *WE line deselects.

Figures 2a and 2b show the read and write cycle paramaters for a typical memory chip. Though these are not the memory chips inside the CoCo, the timing and paramaters are quite similar.

Now with no further *delays*, it is time to look into the CPU . . . well, sort of! There is one more thing we must look into; it is CPU related, though. We are getting closer. It is the master clock, which is a master reference wave form used to synchronize all of the logic in a system.

The master clock is usually the highest frequency in the computer. All other timings are derived (divided) from this clock. The CPU clock is the speed or frequency at which all instructions and data are retrieved and stored to memory. Depending on the system design, the CPU clock can be equal to the master clock, or any division thereof. In the case of the CoCo, the master clock frequency is 14.31818 MHz (mega-hertz or million hertz) and the CPU clock frequency is 1/16 that of the master clock at 0.8948 MHz. Well, there are two clock speeds in the CoCo. Under special conditions, the CPU can work at 1.8 MHz

Figure 3

READ CYCLE

SYMBOL	PARAMETER	MIN	MAX	UNIT
tpc	READ CYCLE TIME	250		NS
ta	ACCESS TIME		250	NS
tco	CHIP SELECT TO OUTPUT		85	NS
tcx	CHIP SELECT TO OUTPUT	10		NS
totd.	OUTPUT TRI-STATE FROM DESELECTION	15		NS
toha	OUTPUT HOLD FROM ADDRESS CHANGE		20	NS

WRITE CYCLE

SYMBOL	PARAMETER	MIN	MAX	UNIT
twc	WRITE CYCLE TIME	250		NS
tw	WRITE TIME	135		NS
twr	WRITE RELEASE TIME	0		NS
totw	OUTPUT 3-STATE FROM WRITE		60	NS
tdw	DATA TO WRITE TIME OVERLAP	135		NS
tdh	DATA HOLD FROM WRITE TIME	0		NS

Now you might say, "Wow, my CoCo has a clock rate of only .894 MHz!" Compared to that of the 4 MHz of other computers, that may or may not be slower. You see, it gets more complicated. The CPU clock does not always mean the net speed of the computer. There are some other factors involved, such as synchronous I/O, as opposed to asynchronous I/O.

Let's look at synchronous I/O first. As the word implies, synchronous I/O means that any memory, read or write, is synchronized. Synchronized to what? The CPU clock, of course. On any given clock cycle, the CPU can do one I/O. You know exactly when the CPU will need the bus. It corresponds to the clock cycle. In an asynchronous situation, the CPU requires more than one clock cycle to do a read or write. Asynchronous I/O requires either three or four cycles depending on what kind of I/O it is doing. On this type of CPU, signals are required to tell memory or other devices that an I/O has started.

Just about now, a little bit of math is required. Given that the clock frequency of the CoCo is 894886 hertz or 0.894 MHz, one clock cycle is 1117 nanoseconds. The way I did this is to transfer from frequency to time period. The equation used is:

$$T = 1/F$$

where 'T' is in seconds and 'F' (frequency) is in hertz. So the frequency of 0.894 MHz is a time period of .000001117 seconds, or 1117 nanoseconds, or 1.117 microseconds. Now, when we talk about speed, we can say that the CoCo can do about one I/O per microsecond — a much more accurate way to measure the effective speed of a CPU.

I hope these articles about the hardware of the CoCo are informative to you. Also, I hope I am not going too fast; it is hard for me to judge what audience I am writing for. If you have some comments to make, a direction to take or something you don't understand, write to me through RAINBOW and I'll try to answer the interesting and common ones here in this column. Next time, we'll look deeper into the heart of the CoCo.

CoCoConnection

To utilise this program, you will require a CoCoConnection - available from ourselves, or from Blaxland Computer Services.

This unit allows you to connect your CoCo to the outside world - ie, to models, Burgular Alarms, Printers, Synthesisers - and so on.

BITMAN

by Peter Feldtmann

(Sorry 'bout the trains project - time flies when you've got a magazine to run! I hope to be able to give you further details on that project NEXT month! G.)

This month Peter Feldtmann has given us this routine which enables a multitude of operations from simple keyboard inputs to CoCo.

All the programs occuring in this section of the magazine are supplied with the CoCoConnection when you purchase it.

Our thanks to Blaxland Computer Services for passing on Peter's work. You can see the CoCoConnection operating at Blaxland.

The Listing:

```
1 'BITMAN for the CoCoConnection
   This program enables sequence
s of bit combinations to be set
up including delays, by simple k
eyboard input.
   Prog by Peter Feldtmann
 C/o Blaxland Computer Services.
2 GOTO10
3 SAVE"BITMAN:2":STOP
10 REM INITIALIZATION ROUTINE
20 REM INITIALIZE PORT 2B AS OUT
PUTS
30 POKE 65415,4:POKE 65414,255:P
OKE 65415,0:POKE 65414,255:POKE
65415,4
40 REM INITIALIZE PORT 2A AS INP
UTS
50 POKE 65413,0:POKE 65412,0:POK
E 65413,4
60 CLS:PRINT "PORT 2A INITIALIZE
D AS INPUTS"
70 PRINT
80 PRINT "PORT 2B INITIALIZED AS
 OUTPUTS"
90 GOTO 5000
100 CLS:Q = 65414
110 INPUT"PLEASE ENTER NUMBER OF
 SETTINGS";COUNT
120 DIM NPT(COUNT),NUM$(COUNT),D
(COUNT)
125 PRINT " "
126 PRINT " "
130 FORJ=1 TO COUNT
135 PRINT " "
136 PRINT " "
140 GOSUB 1000
150 GOSUB 2000
160 NEXT J
170 FOR J = 1 TO COUNT
180 PRINT USING"####";NTP(J)
190 NEXT J
200 PRINT :PRINT"PRESS ANY KEY T
O CONTINUE"
205 A$=INKEY$:IF A$="" THEN 205
210 FOR I = 1 TO COUNT
220 POKE Q,NTP(I)
230 FOR J = 1 TO D(I):NEXT J
240 NEXT I
250 PRINT :PRINT"PRESS ANY KEY T
O CONTINUE"
255 A$=INKEY$:IF A$="" THEN 255
260 GOTO 3000
1000 PRINT:INPUT"ENTER OUTPUT NU
MBERS TO SET";NUM$(J)
1010 FOR I =1 TO LEN(NUM$(J))
1020 N$=MID$(NUM$(J),I,1)
1030 A=INSTR("87654321",N$)-1
1040 TMP =2^A:NTP(J) = NTP(J) +
TMP
1050 NEXT I
1060 RETURN
2000 INPUT"PLEASE ENTER DELAY";D
(J)
2010 RETURN
3000 PRINT "WANT TO SAVE THIS SE
QUENCE (Y/N) ";
3010 A$=INKEY$:IF A$="" THEN 301
0
3020 IF A$="Y" THEN 4000
3030 IF A$<>"N" THEN PLAY "GFEDC
":GOTO 3010
3040 RUN
4000 INPUT "ENTER FILENAME ";NAM
$
4010 OPEN "O",-1,NAM$
4015 PRINT #-1,COUNT
4020 FOR I=1 TO COUNT
4030 PRINT #-1,NTP(I),D(I)
4040 NEXT I
4050 CLOSE
4060 RUN
5000 PRINT "LOAD SEQUENCE FROM T
APE (Y/N) ";
5010 A$=INKEY$:IF A$="" THEN 501
0
5020 IF A$="Y" THEN 6000
5030 IF A$<>"N" THEN PLAY"GFEDC"
:GOTO 5000
5040 GOTO 100
6000 INPUT "ENTER FILENAME TO LO
AD ";NAM$
6010 OPEN "I",-1,NAM$
6020 INPUT #-1,COUNT
6025 DIM NTP(COUNT),D(COUNT)
6030 FOR I=1 TO COUNT
6040 INPUT #-1,NTP(I),D(I)
6050 NEXT
6060 CLOSE
6070 GOTO 170
10000 POKE150,41:PRINT#-2,CHR$(2
7);CHR$(29);CHR$(27);CHR$(82);CH
R$(32)
```

BARDEN'S BUFFER

Listening to Your CoCo with Assembly Language

By William Barden, Jr.

One of the nice things about assembly language is that it gives you access to parts of the computer that just can't be handled through BASIC. Take sound effects and music, for example. Sure, you can use SOUND in Extended BASIC to sound a tone for a certain length of time. You can also use the PLAY command in Extended BASIC to play musical notes. However, with BASIC you are limited to these short, simple tones. Assembly language, on the other hand, allows you to create a variety of complex sounds. Want a car crash, a phaser blast, or the sound of an Apple IIe being dropped from the top floor of One Tandy Center? Assembly language is the only way to go. Would you believe that these sounds are already programmed into your CoCo? I'll show you how to unleash the CoCo's sounds in this column. Actually, there'll be two major themes this month. First, we discuss assembly language sounds. Secondly, I'll show you how to plan and use a *Sound* program, for those of you who are still a little shaky about using *EDTASM+* or Disk *EDTASM*. As I mentioned last month, you OS-9 users can still benefit from the column, but the examples will be in *EDTASM* format.

Color Computer Sounds

Sounds on the Color Computer are generated quite differently from sounds on the Tandy 1000 or other systems. Many other systems contain a sound synthesizer chip. This is an integrated circuit similar in appearance to many of the chips you'll see inside the CoCo. Internally, though, a sound synthesizer chip contains logic to generate square waves or sine waves and to create different envelopes that determine the wave shape.

The Color Computer does not use a sound synthesizer chip. Instead, it creates sounds by electronic logic that makes up a digital-to-analog converter. I'll call this logic a DAC for short. The CoCo DAC is a "six-bit" DAC, meaning that it will convert a digital value of zero through 63 into 64 different voltage levels. The CoCo DAC uses the upper six bits of a byte in the conversion. Here are the results we'll get with a range of values:

Digital Value	Voltage Output
00000000	.23 volts
00000100	.30
00001000	.37
00001100	.44
00010000	.52
00010100	.59
00011000	.66
.	.
.	.
.	.
11111000	4.69
11111100	4.76

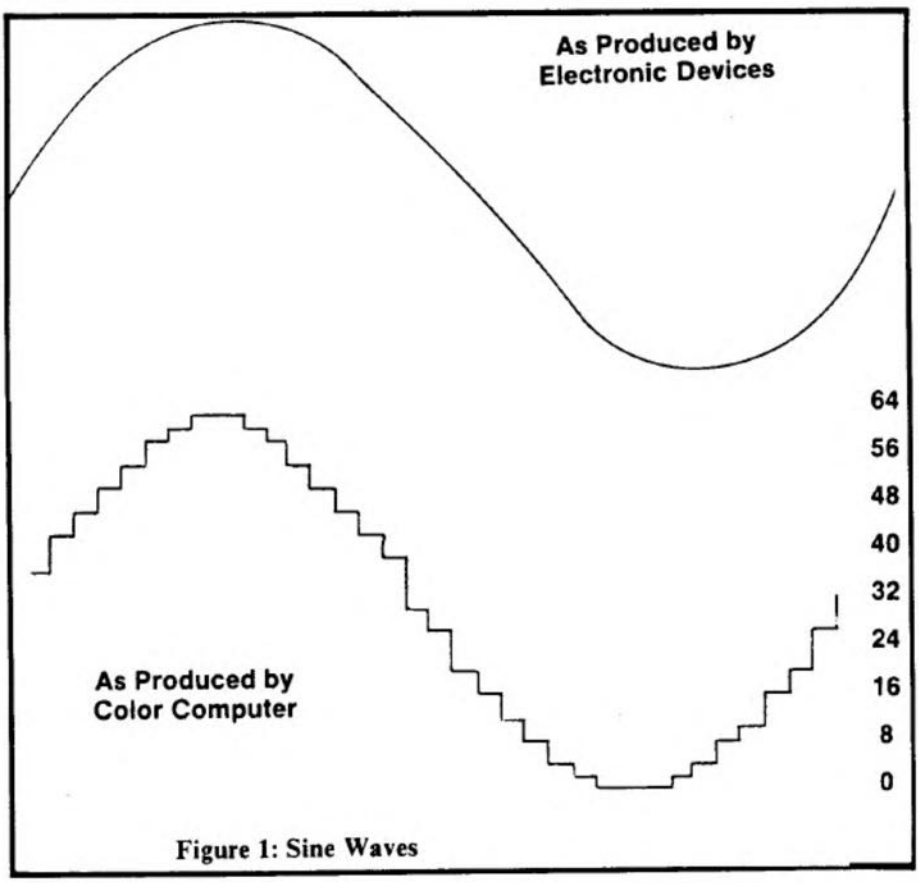


Figure 1: Sine Waves

Note that the the lower two bits of the digital value are always zero — only the upper six bits change. You can also see that the step size of the voltage output is constant. There is always about 0.07 volts between one digital value and the next.

All well and good, but how does the DAC create sounds? Let's take an example. The purest sound is a sine wave, shown in Figure 1. The second part of the figure shows a comparable sine wave generated by the DAC. When fed into an audio amplifier, the result will be a relatively pure musical tone, similar to that produced by an electronic doorbell, or a Dolby test tone.

If you look closely at the DAC sine wave, you can see that it's made up of a series of discrete voltage levels, giving it a "staircase step" appearance. The closer the interval is between steps, the smoother the sine wave becomes, as shown in Figure 2.

This sine wave was generated by the CoCo DAC from a table in ROM. Believe it or not, this is the way the CoCo generates the 1200 and 2400 hertz (cycles per second) tones used for cassette tape output! (I'll tell you where to find the table later.)

The distance from crest to crest, or from trough to trough of the sine wave is called the *period* of the sound. The reciprocal of the period is the *frequency* of the sound. A 600 hertz tone, then, has a period of 1/600 seconds or about 1.66 milliseconds (1.66 thousandths of a second).

More Complex Sounds

Imagine tuning in MTV and listening to a new heavy metal band playing sine wave synthesizers. It would drive the viewers to Mozart! Most natural and instrumentation sounds are made up of a combination of frequencies, as shown in Figure 3. Random sounds, such as surf or crowd noise, are made up of an even combination of all frequencies, giving a hissing effect. These are the sounds that BASIC cannot create on the CoCo with simply the SOUND and PLAY commands.

A music synthesizer not only provides the capability to generate the complex sound of strings or a flute, but it also allows the user to define an envelope for the sound. The envelope describes how the sound varies in loudness and is sometimes called an ADSR, for attack, decay, sustain, release, as shown in Figure 4. A musical instrument such as a piano has a sustained sound, while an instrument such as a snare drum has a much shorter duration sound. Both envelopes are different, as shown in the figure. Synthesizers also provide the capability to create envelopes not produced by musical instruments, such as a sound that starts off at the minumum loudness and builds to a maximum, producing a sound like a musical tone played backwards on a tape recorder.

Many complex sounds can be generated by the CoCo, at the expense of building a table of values that define the wave shape of the sound. Another alternative is to use the patterns found in the CoCo's ROM, selecting those that produce the sounds you're looking for. A short section of BASIC ROM code from $A9EB through $A9FF on my systems, for example, produces the wave shape shown in Figure 5 when the upper six bits are considered.

Any small section of code can be repeated over and over, and the interval between outputs to the DAC can be varied by timing loops within the assembly language program to produce differ-

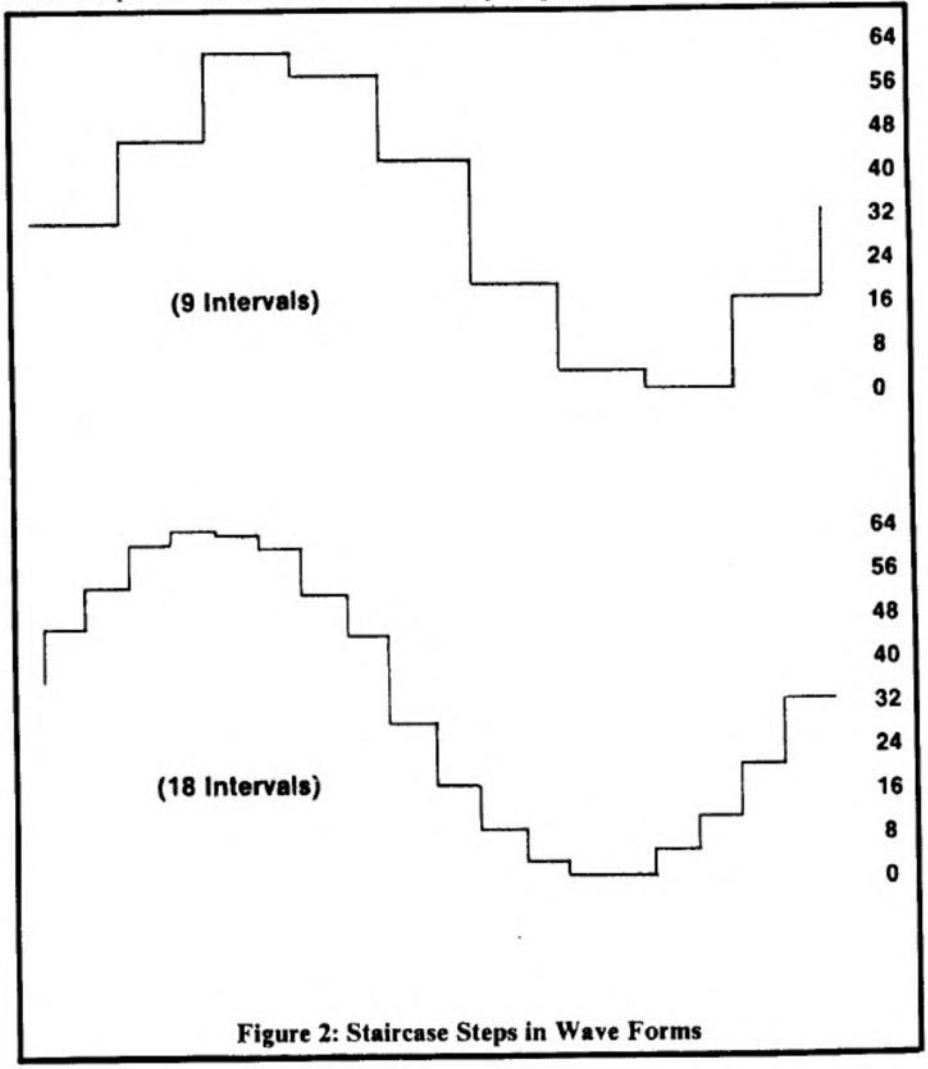


Figure 2: Staircase Steps in Wave Forms

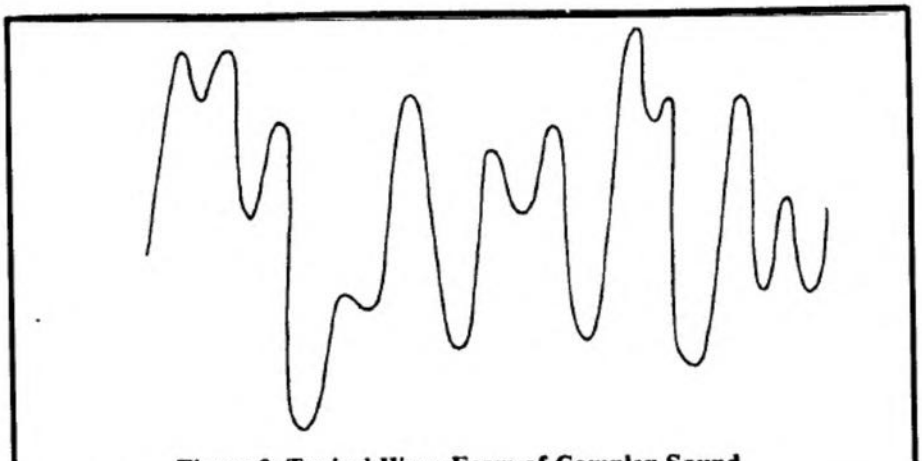


Figure 3: Typical Wave Form of Complex Sound

ent frequencies. As the shortest timing loop can produce periods that are about 10 microseconds wide, the highest frequencies that can be produced in assembly language are 100,000 hertz radio waves (!), far above the 6,000 hertz sound waves that can be passed through the CoCo electronics. That's the beauty of assembly language here — there's plenty of time left over.

Talking to the PIA of the CoCo

The output of the DAC goes both to the cassette output and to a device that routes the DAC output to the television sound channel, as shown in Figure 6. The MC14529 routes the DAC output to the TV sound channel by two "select" bits set by the following BASIC commands:

```
180 POKE &HFF01,PEEK(&HFF01)
    AND &HF7 'select bit 0
190 POKE &HFF03,PEEK(&HFF03)
    AND &HF7 'select bit 1
200 POKE &HFF23,PEEK(&HFF23)
    OR 8 'set 6-bit sound
```

The third POKE here sets six-bit sound as opposed to a single-bit "on/off" sound that can also be used.

Once these commands are given, they need not be output again — the DAC is routed to the TV sound channel for the duration of the program.

The six inputs to the DAC are controlled by six signals from another source, as shown in the figure. The source here, as in the case of the two select signals, is a PIA, or peripheral interface adapter. The CoCo uses a number of PIAs to provide programmable signals to control color graphics, sound, cassette operations, and RS-232-C operations, to name a few. In this case, the PIA acts as a simple memory device, holding whatever six bits have been sent to it until another six bits are sent. In BASIC the six PIA to DAC outputs are set by

```
1000 POKE &HFF20, VALUE*4
     'VALUE is 0 - 63
```

In assembly language, the instructions are very similar:

```
LDA #VALUE     value is VVVVVV00
STA $FF20      outputs value to DAC
```

And that's about all there is to producing sounds on the Color Computer — route the DAC output to the television channel and then send out the proper patterns to the PIA/DAC, spaced at even intervals, repeating the patterns if necessary.

Putting Together a Sound Assembly Language Program

Now that we know enough about the sound capabilities of the Color Computer, we can put together a short program to play a variety of sounds, natural and unnatural. What we're looking for is a program that will route the DAC output to the television sound channel and then output a series of digital values to the DAC, spaced at regular intervals. We also need the capability of repeating a series of values for a certain number of

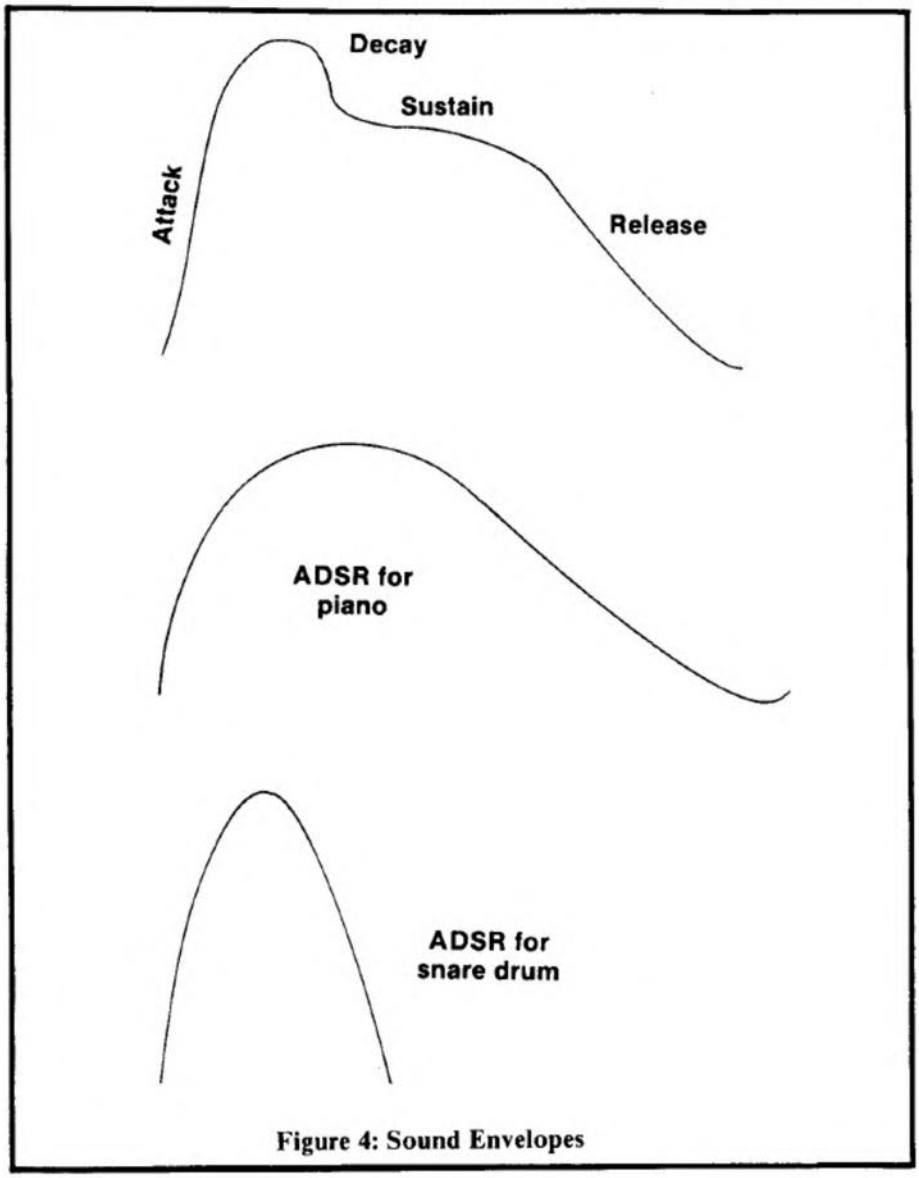


Figure 4: Sound Envelopes

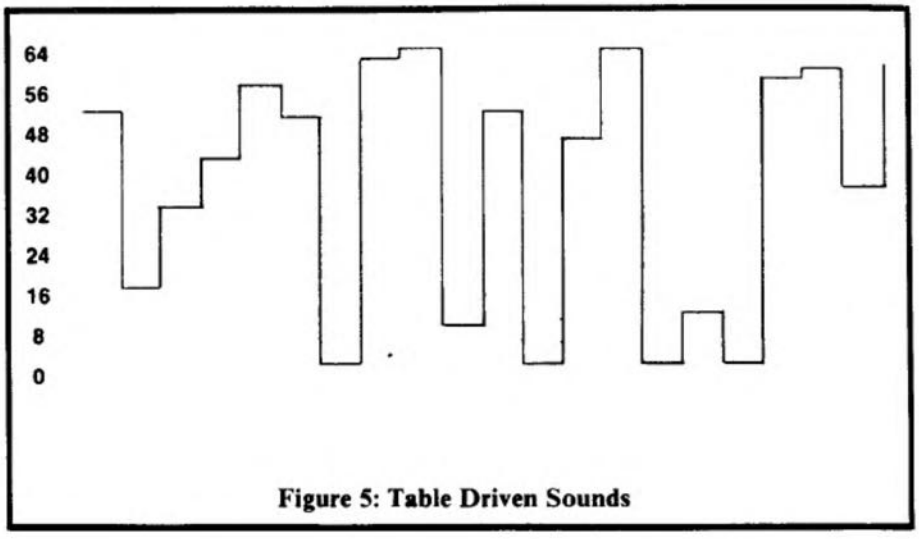


Figure 5: Table Driven Sounds

times.

The data that creates the sounds will be held in a table in memory, either a table of values that already exist, such as ROM values, or a table that we will create. Since we want to make the program handle a table of varying length, we'll need to specify a table length. An alternative to this is to use a "terminating value" at the end of the table to mark the end. However, we'd like to use ROM data for some of the sounds, and it's awfully difficult to write data to ROM (although one of my CoCos tries this on occasion).

To make the table values easier to generate, we'll also let the program shift the data so that it's aligned in the upper six bits. That way we can put values of zero through 63 in memory bytes without having to worry about what the values would be in their shifted form. Of course another approach is to "pack" the data into consecutive six bits, but this would present a real chore in creating and maintaining the table of data values.

What we have so far, then, is a program that will read a table of values starting from some given memory location and ending at another memory location, with each byte in the table representing an output value of zero through 63. Such a table is shown in Figure 7 — it's the encoded form of a simple bell sound using a square wave frequency.

Another thing that we need to specify to the program is the interval between DAC outputs. Remember, the smaller the increment, the less rough the final wave shape will be. What is a reasonable increment to implement? We know that we'll have to have a timing loop in the program to count off the time between DAC outputs. (Another alternative, however, would be reading in a PIA bit that shows the sync clock for video, appearing every 63.5 microseconds.) Even with assembly language, instructions take a finite time and we can't define a small enough interval by the time the program is coded. We'll let the interval be specified by a count parameter to the assembly language program and see what the minimum interval turns out to be in the final result.

A final parameter that must be specified to the program is the number of times the table must be repeated. A repeat capability is handy to have to generate wave forms that are periodic, such as the sine waves and square waves mentioned above. We may want to repeat thousands of times with short tables of values to get a sound that is seconds long.

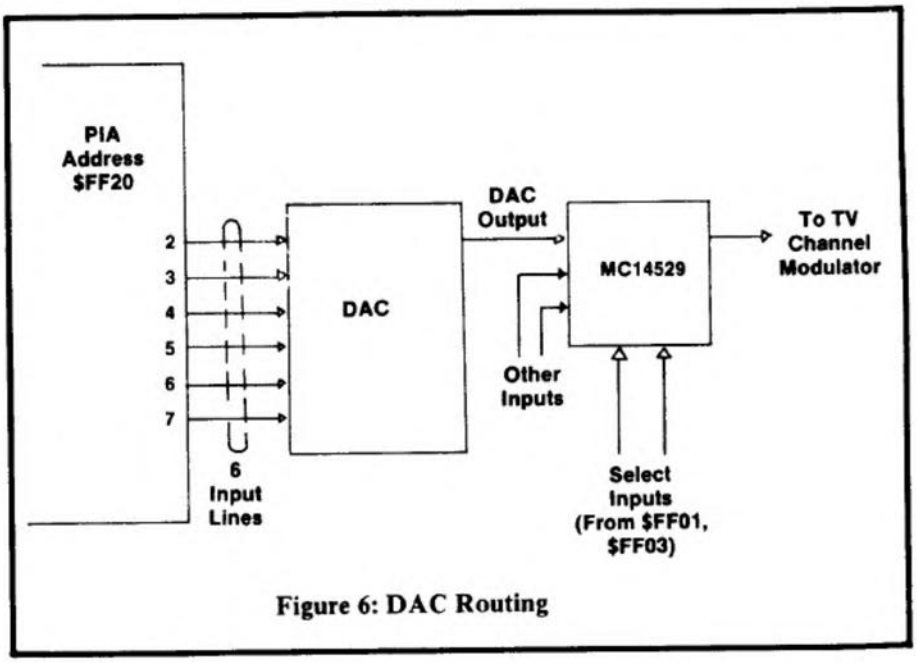


Figure 6: DAC Routing

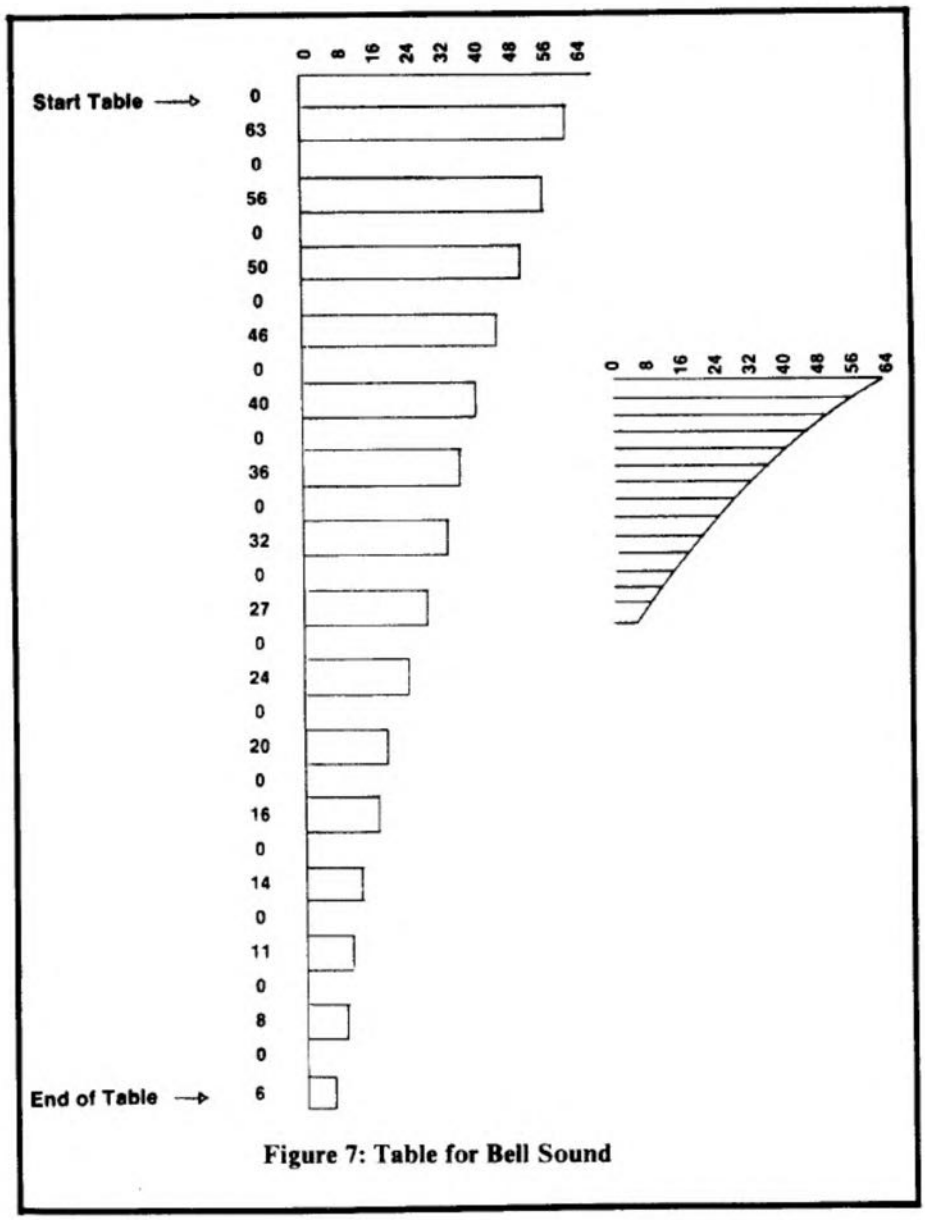


Figure 7: Table for Bell Sound

At this point we have these parameters that must be passed to the assembly language program:

- A 16-bit address that specifies the start of the table.
- A 16-bit address that specifies the end of the table.
- A 16-bit delay count that determines the time delay between outputs to the DAC.
- A 16-bit repeat count that determines the number of times that the table data is to be repeated.

Program Design Considerations

Before we start coding the program we need to make several more decisions about the basic design:

- Is this to be a program or a subroutine?
- Where in memory will the program be?
- Where in memory will the table be?

Programs Versus Subroutines

We could make the *Sound* program a full fledged program that could be loaded by LOADM and executed by EXEC (or from cassette by CLOADM and EXEC). However, this doesn't make too much sense, as the program isn't really a full-blown program, but simply a short piece of code that can be used to (presumably) generate short snatches of sounds. For that reason, it makes better sense to design it as an assembly language subroutine that can be called by BASIC (or other languages). That way we can use the convenience of BASIC to do all of the housekeeping and just call the assembly language subroutine when a sound is required.

The commands that Extended BASIC uses to interface to assembly language are DEFUSR and USR. DEFUSR tells BASIC where the assembly language code is located, while USR actually transfers control to the assembly language subroutine. The assembly language subroutine must always end with an RTS instruction, a ReTurn from Subroutine. The RTS acts just as a BASIC statement does, returning control back to the BASIC statement after the USR. The typical call to our yet-uncoded sound subroutine would look like this:

```
110 DEFUSR0 = &H3E08  done only
    once in BASIC
  .
  .
330 A = USR0(0)  'call assembly
    language sub
340 . . .              'return here
```

The dots between statements 110 and 2100 represent other BASIC statements that are executed. One thing that must be done before the subroutine is executed, of course, is to set up a table of data in memory that the subroutine will use to generate sounds, or to point to the table if it already exists (such as the ROM sine wave table).

Where in Memory Will the Subroutine Be?

The DEFUSR statement defines where the assembly language subroutine is in memory. But just where should it be? There are many places it could be, but the overriding rule is to *put it out of the way of BASIC*. BASIC is constantly changing memory by adding variables, manipulating strings, and using a stack area, and any assembly language code must be put into an area that BASIC cannot touch.

One of the best ways to do this is to use a protected area of high memory. The CLEAR statement in BASIC is specifically designed to protect memory so that assembly language subroutines can be put there. Doing a `100 CLEAR 400,&H3DFF` for example, protects all memory from locations &H3E00 on up to the top of your RAM memory (512 bytes less than 16K to the top of memory). We'll use this area for the *Sound* program, and you might keep it in mind for your own programs. Of course, if you have a 64K system, you could use `100 CLEAR 400,&H7DFF`.

The 400 value, by the way, establishes the size of the BASIC string storage area. Use a larger value if you have many string manipulations in your program or if you have more than 64K.

How does the assembly language

Figure 8:

```
1ØØ CLEAR 4ØØ,&H3DFF    'done at beginning of BASIC
11Ø DEFUSRØ = &H3EØ8    'done only once in BASIC
12Ø DATA &HXX, &HXX, ... &HXX  'machine language form
121 DATA &HXX, &HXX, ... &HXX  'of assembly language
14Ø FOR I = &H3EØØ TO &H3E2A   'subroutine
15Ø READ ML             'move the machine language to
16Ø POKE I,ML           'the &H3EØØ area - done once
17Ø NEXT I
18Ø POKE &HFFØ1,PEEK(&HFFØ1) AND &HF7  'select bit Ø
19Ø POKE &HFFØ3,PEEK(&HFFØ3) AND &HF7  'select bit 1
2ØØ POKE &HFF23,PEEK(&HFF23) OR 8      'set 6-bit sound
    .
    .
33Ø A = USRØ(Ø)         'call assembly language sub
34Ø ...                 'return here
```

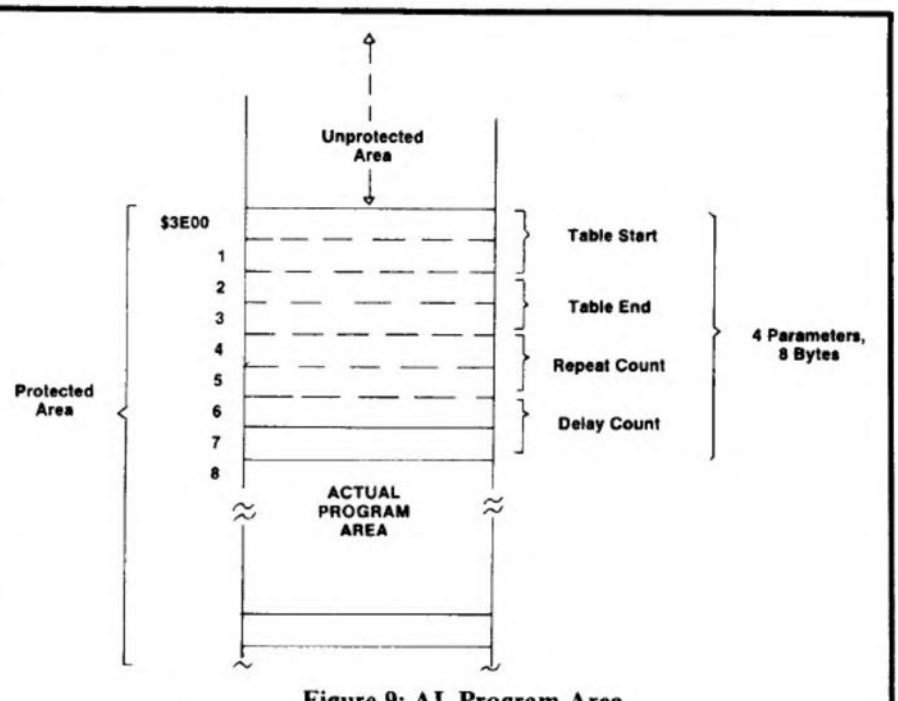


Figure 9: AL Program Area

code actually get into the &H3E00 area? We could load it in by a LOADM (or CLOADM), but it's a real convenience for a short subroutine like this to encode it into BASIC DATA statements and include it in the BASIC program. That way we have everything in one neat little package. The DATA statements use values taken directly from the assembly language listing and look like this:

```
120 DATA &HXX, &HXX, . . . &HXX
121 DATA &HXX, &HXX, . . . &HXX
```

where the XXs stand for machine language values.

To move the machine language values in the DATA statements, a simple FOR/NEXT loop in BASIC is used:

```
140 FOR I = &H3E00 TO &H3E2A
150 READ ML
160 POKE I,ML
170 NEXT I
```

The skeleton form of our BASIC call now looks like this (including signal routine to the TV channel): See Figure 8.

Where's the Table?

The next question to resolve before actually coding the program is to determine where the table of data is to be and how to tell the assembly language program where it is. We mentioned before that we'd like to explore some of the areas of ROM to see what kinds of sounds could be generated from the relatively random data found there. For that reason, we can't just use a preassigned area of memory as a table. The table address, therefore, will not be fixed.

In addition to the table address, we have to consider how the other parameters will be passed to the assembly language program. The USR statement allows for passing one 16-bit parameter to the assembly language program. This would only take care of one parameter, though, and we have four — table start, table end, delay count, and repeat count.

There are a number of methods that can be used to pass more than one parameter, but we'll choose a simple one for this program — we'll put the parameters in memory right before the program at &H3E00. The assembly language program area will now look like Figure 9, a block of four parameters in eight bytes, followed by the (still) undefined assembly language code. The parameters can easily be poked into the &H3E00 area from BASIC, and we can change all of them at any time by doing a series of POKEs.

Coding the Program

Whew! We're finally at the point at which we can start coding the *Sound* program. Everything we've done up to this point has been program design. Generally, the more time spent in design, the fewer changes we'll have in coding and debugging the program.

Before coding this program, I gave some thought about what registers could be used to hold the parameters. In such a short program, it is possible to dedicate registers for specific functions. In longer programs, of course, the registers can't be dedicated to any specific thing, but handle all kinds of tasks.

In *Sound*, the registers are set up this way:

- Register A is used as the main "working" register, holding the values to be sent to the DAC and other results.
- Register X holds the 16-bit delay count.
- Register U holds the table start initially, but is incremented by one to point to successively higher values in the table.
- Register Y holds the 16-bit repeat count.

Having as many things as possible in registers speeds up *Sound* considerably.

The complete listing of *Sound* is shown in Listing 1. Here's a short discussion of how it works. First, the repeat count is loaded into 'Y' from the parameter block (it's four bytes from the address in 'U'). This repeat count will be decremented down to zero through the program. When it reaches zero, the program has repeated the table values the number of times specified in the repeat count. Notice that this is a "Program Counter Relative" instruction that does not specify an absolute address. This and other instructions in *Sound* are relocatable, meaning that the machine language code can be moved anywhere in memory without having to reassemble the program.

Next, Register X is loaded with the delay count and Register U is loaded with the table address count from the parameter block.

Listing 1: *Sound* Program in Assembly Language

```
3EØØ                ØØ1ØØ          ORG     $3EØØ
3EØØ                ØØ11Ø BLK      RMB     8          PARAMETER BLOCK
3EØ8 1ØAE 8C F8     ØØ12Ø START    LDY     BLK+4,PCR GET REPEAT COUNT
3EØC AE   8C F7     ØØ13Ø          LDX     BLK+6,PCR GET DELAY COUNT
3EØF EE   8C EE     ØØ14Ø DACØØ5   LDU     BLK,PCR     GET TABLE POINTER
3E12 A6   CØ        ØØ15Ø DACØ1Ø   LDA     ,U+           GET VALUE
3E14 48             ØØ16Ø          LSLA                  ALIGN TO 6 BITS LEFT
3E15 48             ØØ17Ø          LSLA                  BY TWO SHIFTS
3E16 B7   FF2Ø      ØØ18Ø          STA     $FF2Ø         SEND TO DAC
3E19 1F   1Ø        ØØ19Ø          TFR     X,D           GET DELAY COUNT
3E1B 83   ØØØ1      ØØ2ØØ DACØ2Ø   SUBD    #1              DECREMENT BY ONE
3E1E 26   FB        ØØ21Ø          BNE     DACØ2Ø          GO IF NOT DOWN TO Ø
3E2Ø 11A3 8C DE     ØØ22Ø          CMPU    BLK+2,PCR     TEST FOR END
3E24 23   EC        ØØ23Ø          BLS     DACØ1Ø        GO IF NOT END OF TABLE
3E26 31   3F        ØØ24Ø          LEAY    -1,Y        DECR REPEAT COUNT BY ONE
3E28 26   E5        ØØ25Ø          BNE     DACØØ5      GO IF NOT DOWN TO Ø
3E2A 39             ØØ26Ø          RTS               RETURN TO BASIC
          ØØØØ      ØØ27Ø          END
ØØØØØ TOTAL ERRORS
```

Listing 2: SOUND

```
9Ø '**************************
91 '*SOUND PROGRAM EXERCISER *
92 '*RELOCATES AND CALLS AL  *
93 '*SOUND PROGRAM.          *
94 '**************************

1ØØ CLEAR 4ØØ,&H3DFF
11Ø DEFUSRØ=&H3EØ8
12Ø DATA &HØØ,&HØØ,&HØØ,&HØØ
121 DATA &HØØ,&HØØ,&HØØ,&HØØ
122 DATA &H1Ø,&HAE,&H8C,&HF8
123 DATA &HAE,&H8C,&HF7,&HEE
124 DATA &H8C,&HEE,&HA6,&HCØ
125 DATA &H48,&H48,&HB7,&HFF
126 DATA &H2Ø,&H1F,&H1Ø,&H83
127 DATA &HØØ,&HØ1,&H26,&HFB
128 DATA &H11,&HA3,&H8C,&HDE
129 DATA &H23,&HEC,&H31,&H3F
13Ø DATA &H26,&HE5,&H39
14Ø FOR I=&H3EØØ TO &H3E2A
15Ø READ ML
16Ø POKE I,ML
17Ø NEXT I
18Ø POKE &HFFØ1,PEEK(&HFFØ1) AND
    &HF7
19Ø POKE &HFFØ3,PEEK(&HFFØ3) AND
    &HF7
2ØØ POKE &HFF23,PEEK(&HFF23) OR8
21Ø INPUT "TABLE START";TS
22Ø INPUT "TABLE END";TE
23Ø INPUT "REPEAT CNT";RC
24Ø INPUT "DELAY CNT";DC
25Ø POKE &H3EØØ,INT(TS/256)
26Ø POKE &H3EØ1,TS-INT(TS/256)*
    256
27Ø POKE &H3EØ2,INT(TE/256)
28Ø POKE &H3EØ3,TE-INT(TE/256)*
    256
29Ø POKE &H3EØ4,INT(RC/256)
3ØØ POKE &H3EØ5,RC-INT(RC/256)*
    256
31Ø POKE &H3EØ6,INT(DC/256)
32Ø POKE &H3EØ7,DC-INT(DC/256)*
    256
33Ø A=USRØ(Ø)
34Ø GOTO 21Ø
```

The code in DAC020 and the next instruction is the innermost loop in the program. Before this loop, a DAC value has been output by the STA $FF20 and the delay count has been loaded into Register D from Register X by the TFR X,D instruction. The SUBD decrements this count and the BNE instruction causes a loop back to DAC020 while the count in 'D' is not equal to zero.

The code from DAC010 through five instructions from the end (BNE DAC010) is the next innermost loop. It makes a complete pass through the table of values, sending each value out to the DAC and delaying with the inner loop just discussed. The two LSLA instructions shift the data from the table to align it in the six high bits of 'A'. The table data is pointed to by 'U', which is used to load the data into 'A'. The LDA ,U+ also increments the pointer value in 'U' by one after the load is done. The CMPU BLK+2,PCR at the end of the loop continually compares the pointer value in 'U' to the end of the table value at two bytes from the parameter block start.

The outermost loop is from DAC005 through the BNE DAC005. It repeats a pass through the table values for the number of times equal to the repeat count. The repeat count in 'Y' is decremented by the LEAY -1,Y each time a complete set of table values has been sent out. When this value has been decremented down to zero, the BNE DAC005 is *not* done, and the program returns to BASIC by the RTS.

How to Use Sound

You can assemble *Sound* yourself, or simply use the BASIC version shown in Listing 2. The BASIC version will ask you for the Table Start value, Table End value, Repeat Count, and Delay. From here on in you're on your own as to what you specify for these parameters, but here are some suggestions:

To hear how your computer's memory data sounds, specify a start of zero and an end of &HFFFF with a repeat count of 1. You should vary the delay to get higher or lower frequencies as well.

Try the sine wave table at &HA85C through &HA87F to hear the sine wave used in the cassette output. (However, before you do, change Line 125 to read 125 DATA &H12,&H12,&HB7,&HFF to change the LSLA instructions to "no operations." The sine wave table data is already aligned to the left, ready to be sent out to the DAC.) Specifying different delay counts will create higher and lower frequencies and specifying repeat counts of other than '1' will sound the sine wave for longer periods of time.

Try short sections of code repeated many times to create "real-world" noises like car crashes, phaser blasts, and the like.

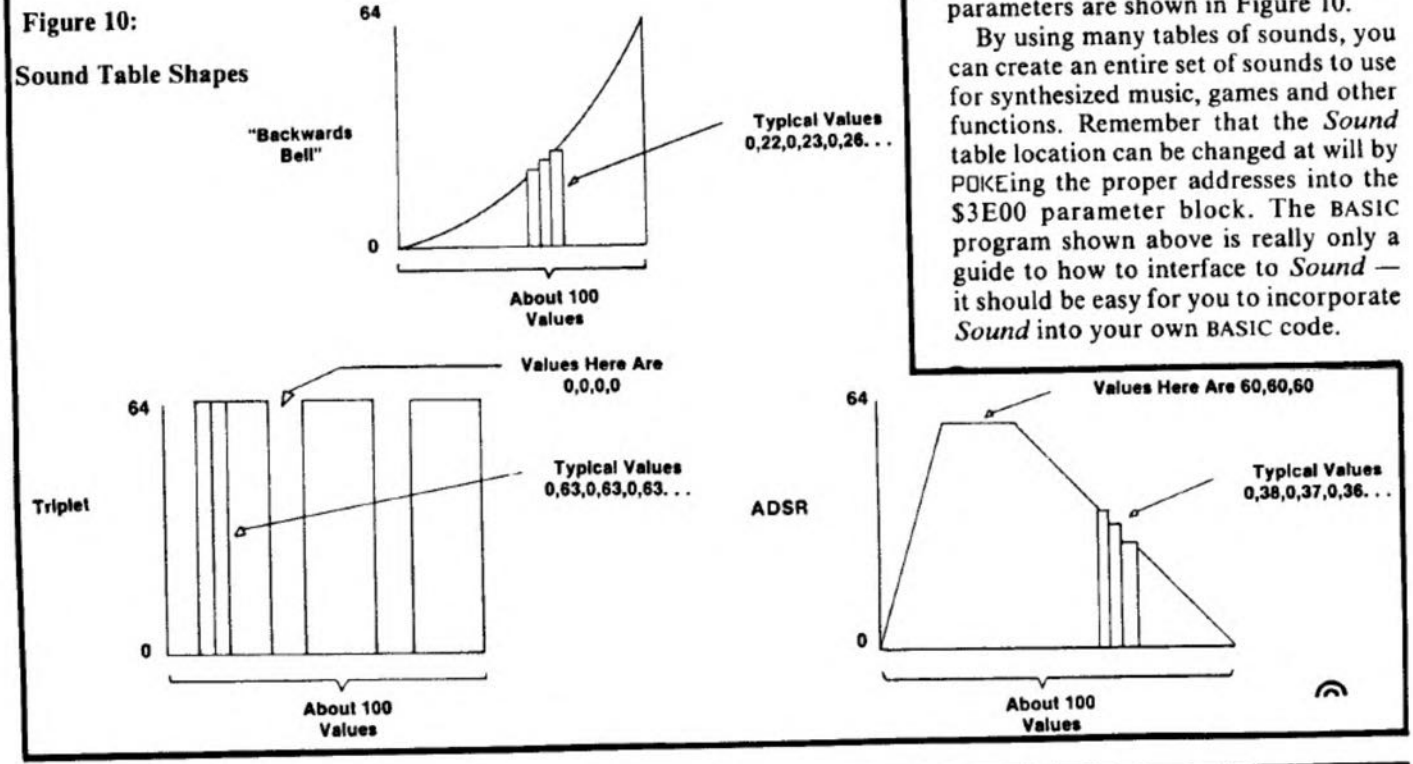
Constructing Your Own Sound Tables

Just as the sine wave table provides a sine wave sound output, you can construct your own tables of sounds. There's plenty of room to do this in the area beyond the end of the program at &H3E2B through &H3FFF. POKE the values by DATA statements and READ loops just as in moving the machine language code. Some suggestions with parameters are shown in Figure 10.

By using many tables of sounds, you can create an entire set of sounds to use for synthesized music, games and other functions. Remember that the *Sound* table location can be changed at will by POKEing the proper addresses into the $3E00 parameter block. The BASIC program shown above is really only a guide to how to interface to *Sound* — it should be easy for you to incorporate *Sound* into your own BASIC code.

Figure 10:

Sound Table Shapes



CLEARLY SAID, CLEANLY DONE

by John Redmond

Most of the Color Computers around seem to have Extended Color Basic and most of the programs that appear seem to assume this. But why? It's pretty much because of the graphics capabilities - the PMODEs, LINE, CIRCLE, etc. This month, we start an orderly implementation of no less than eleven display modes (ECB allows only five), together with Cartesian capabilities (like LINE, CIRCLE and simple user extensions to SQUARE, TRIANGLE). Then we'll extend things to Turtle Graphics and String Graphics. AND we will do this using absolutely standard Forth and NO extended Basic.

Because the Forth is squeaky-clean standard, it is, with a few patchings at the lowest level, portable to other computers. Furthermore, a from-the-ground-up graphics implementation gives us a chance to design an applications language, which we will call XGAL (eXtended Graphics Applications Language), with a sensible, rememberable, syntax. Look at the parameters for ECB's CIRCLE statement. They are a good example of what to avoid. Microsoft have simply tried to do too much with a single statement. In the language we are about to design, we want to say things in a straight-forward way, like:

```
FINE COLOUR GRAPHICS
MEDIUM SEMI GRAPHICS
RED INK
GREEN PAPER
NEW DISPLAY, etc.
```

The mechanics of implementation of the graphics modes are dealt with in Section IV of the Color Basic manual. All the essential information is there, but it is easy to be put off by the complexity of all the pokes to the VDG, control register and SAM. To make things worse, the mapping of Cartesian coordinates onto the graphics screen is messy for displays other than those in the highest-resolution two-colour mode. It is tempting to be satisfied with an imcomplete graphics package, but let's stick to our guns and make sure that we include all the best modes, including three best semi-graphics modes. Two rather dubious modes have been left out, but I don't think that we'll miss 'em.

There are no less than ten important parameters for each of the eleven modes that we will consider. These are tabulated in GRAPHICSARRAY (see Listing). So far so good, but we have to find some way of allocating the correct set for the mode that we want to use and some way of relating this set to the appropriate set of colour bytes and mask bytes in the COLOURSETS and MASKSETS arrays. How?

The clue is in the syntax. The adjectives COARSE ... FINEST are simply numbers which are placed on the stack for use as an index into GRAPHICSARRAY (look at their definitions). RESOLUTION, COLOUR and SEMI check and modify this number ready for GRAPHICS. GRAPHICS is a really smart word (one that looks like a noun and acts like a verb!). It makes sure that the number on the stack is not too large (why?), then multiplies it by the array width (to get an offset) and adds this result to the array base address. On the top of the stack we now have the address of the start of the correct set of graphics parameters.

The first byte of the set is an offset that COLOURSETS and MASKSETS need to know about. OFFSET therefore pokes this byte into the reserved bytes (initialized as 0 during compilation) at the start of each of these arrays. The rest of the set is then copied into the current attributes array, DESCRIPTION, where it can be accessed by the named pointer words. Finally, a named variable, P/BYTE, is initialized.

Everything is now ready for the two words that give visible results - DISPLAY and PLOT. PLOT will have to wait until next month, but let's look at DISPLAY. A display can be NEW or OLD and/or NORMAL or ALTERNATE. Look at the functions of these adjectives (verbs?) and note how we have permitted an ALTERNATE display ONLY for the true graphics modes and not for the SEMI graphics.

DISPLAY uses values in DESCRIPTION to determine the high-memory pokes carried out by the word REGISTERS (not a noun, but a verb!). This word, in turn, is used by !VDG and VIDEOSTART. I recommend that you set yourself the task of working out how REGISTERS works. It is a good example of Forth's low-level power, but you will need a hint or two:

0< returns a true flag (i.e., 1) if a number is negative. But when is it negative? When the highest bit is 1, of course. Now, if the highest bit IS 1, we have to add 1 to an even address to determine the correct poke location. NOW (wait for it!) the one we add is, in fact, the very logical flag that 0< produced. If the flag had been false, the flag value would have been 0 which, when added to an even address, gives the same even address. This sort of logic arithmetic is discussed in Brodie's 'Thinking Forth'. Finally, 2* is a very efficient way of doing a left shift to get the next binary bit into the top position for checking, if necessary, with 0<. End of lesson.

The background colour for a normal screen clear is held in DESCRIPTION, in a named location called (PAPER). The foreground colour is held in (INK). Either of these can be altered at any time by PAPER or INK, respectively, prefixed by a colour word. The colours are defined using a special defining word, HUE, which uses COLOURED to access the correct element of COLOURSETS using both the ofset passed by HUE and the byte value poked in by GRAPHICS (see above). (Defining words are one of the advanced topics dealt with in Brodie's 'Starting Forth'. They are dazzlingly elegant and powerful and will be discussed in detail in a later article.)

It is unlikely that all this will be grasped at the first reading. Look through the listing methodically, looking at the highest-level (i.e., last) words first. You may, of course, not be even

slightly interested in how it all happens. That's OK. All the work has been done and all you need to do is load these definitions from tape or disk and use them. If you say, from the keyboard or within a program, MEDIUM COLOUR GRAPHICS NEW DISPLAY, you will get just that.

Do you still think that postfix notation looks funny? Next month, drawing lines and circles in all graphics modes, using sensible commands like:

```
: \ 13 WORD DROP ;
\ XGAL - EXTENDED GRAPHICS
\ APPLICATIONS LANGUAGE,
\ VERSION 1.05,
\ (C) JOHN REDMOND, 1986.
\ PART 1.

\ CODE WRITTEN SPECIFICALLY
\ FOR TANDY COLOR COMPUTER
\ (ALL REVISIONS).
\ EXTENDED BASIC NOT REQUIRED.


HEX
\ ATTRIBUTE ARRAYS, CONTAINING:
\ OFFSET INTO COL/MASK ARRAYS
\ TRUE GRAPHICS FLAG
\ VDGBYTE, CONTROLBYTE
\ PIXEL WIDTH & HEIGHT MARKERS
\ SCREEN SIZE IN BYTES
\ NO OF BYTES PER ROW
\ INITIAL B/GROUND COLOUR

CREATE GRAPHICSARRAY
\ 2-COLOUR MODES
001 , 2090 , 403 , 400 , 1000 ,
001 , 60B0 , 402 , 600 , 1000 ,
001 , A0D0 , 400 , C00 , 1000 ,
001 , C0F0 , 300 , 1800 , 2000 ,
\ 4-COLOUR MODES
801 , 2080 , 403 , 400 , 1000 ,
801 , 40A0 , 303 , 800 , 2000 ,
801 , 80C0 , 302 , C00 , 2000 ,
801 , C0E0 , 300 , 1800 , 2000 ,
\ 8-COLOUR SG MODES
1000 , 4000 , 303 , 800 , 2080 ,
1000 , 8000 , 302 , C00 , 2080 ,
1000 , C000 , 300 , 1800 , 2080 ,

HERE GRAPHICSARRAY - 0B /
CONSTANT ARRAYWIDTH

CREATE COLOURSETS 0 C,
FFFF , FFFF , FFFF , FFFF ,
0055 , AAFF , 0055 , AAFF ,
8F9F , AFBF , CFDF , EFFF ,

CREATE MASKSETS 0 C,
8040 , 2010 , 804 , 201 ,
C0C0 , 3030 , C0C , 303 ,
FAFA , FAFA , F5F5 , F5F5 ,

VARIABLE P/BYTE
VARIABLE (INK)
VARIABLE (ALTERNATE)
6800 CONSTANT VBASE

\ CURRENT GRAPHICS ATTRIBUTES.
\ THE ARRAY IS ACCESSED BY
\ SPECIAL (NAMED) POINTERS.

CREATE DESCRIPTION
ARRAYWIDTH 1- ALLOT
DESCRIPTION 3 + CONSTANT WIDTH
DESCRIPTION 4 + CONSTANT HEIGHT
DESCRIPTION 5 + CONSTANT /SCREEN
DESCRIPTION 7 + CONSTANT B/LINE
DESCRIPTION 8 + CONSTANT (PAPER)

\ COLOUR DEFINITIONS

: COLOURED ( #)
0 MAX 7 MIN
COLOURSETS COUNT + + C@ ;
: HUE CREATE C, DOES>
C@ COLOURED ;
0 HUE GREEN
1 HUE YELLOW
2 HUE BLUE
3 HUE RED
4 HUE BUFF
5 HUE CYAN
6 HUE MAGENTA
7 HUE ORANGE
0 HUE PLAIN
80 HUE BLACK

: INK (INK) ! ;
: PAPER (PAPER) C! ;

\ RESOLUTION TYPES
\ AND ASSOCIATED WORDS

0 CONSTANT COARSE
1 CONSTANT MEDIUM
2 CONSTANT FINE
3 CONSTANT FINEST

: RESOLUTION 0 MAX 3 MIN
0 (ALTERNATE) !
RED INK ;
: COLOUR RESOLUTION 4 + ;
: SEMI 0 (ALTERNATE) !
0 MAX 2 MIN 8 +
YELLOW INK ;

: OFFSETS C@ DUP
COLOURSETS C!
MASKSETS C! ;

: GRAPHICS ( #)
0A MIN
ARRAYWIDTH * GRAPHICSARRAY +
DUP OFFSETS
1+ DESCRIPTION
ARRAYWIDTH 1- CMOVE
WIDTH C@ 3 >
IF 0F ELSE 7 THEN P/BYTE ! ;

\ SPECIAL WORDS TO POKE THE
\ ODD OR EVEN ADDRESSES.

: REGISTERS ( BYTE,ADDR,#REGS)
2* NEGATE OVER + SWAP
DO I OVER 0< +
0 SWAP C! 2*
-2 +LOOP DROP ;

: !CONTROL ( MASK)
FF22 C! ;

: !VDG ( BYTE)
8 << FFC4 3 REGISTERS ;

: CLEARSCREEN VBASE /SCREEN @
(PAPER) C@ FILL ;

: VIDEOSTART ( VBASE)
FFD2 7 REGISTERS ;

\ COLOUR SET WORDS.

: NORMAL ;
: ALTERNATE
DESCRIPTION C@
(ALTERNATE) ! ;

0 CONSTANT OLD
1 CONSTANT NEW
: DISPLAY ( T/F)
( CONTROLBYTE,VDGBYTE)
IF CLEARSCREEN THEN
VBASE VIDEOSTART
DESCRIPTION 1+ DUP
C@ !VDG
1+ C@ (ALTERNATE) @
IF 8 OR THEN
!CONTROL ;

\ EXAMPLES OF USE:

\ GREEN INK RED PAPER

\ MEDIUM SEMI GRAPHICS
\  NEW DISPLAY

\ FINEST COLOUR GRAPHICS
\ ALTERNATE NEW DISPLAY

\ COARSE RESOLUTION GRAPHICS
\ NORMAL OLD DISPLAY

\ 6800 VIDEOSTART
```

FORTH FORUM

by John Poxon

Before we begin this month's Forth Forum, I have about 60 c30 audio mastering tapes for sale, at $1.20 each, plus postage. All are relatively new, fault free and bulk erased. Most have been recorded on once only.

Last month's article introduced Reverse Polish Notation as the modus operandi of FORTH arithmetical operations. The concept of placing the operator (+,-,* and /) after the numbers to be operated on is fundamental to FORTH and Reverse Polish Notation (R.P.N.).

R.P.N. is not a creation of or for FORTH, but originated in considerations of how arithmetical operations could be more logically and efficiently performed.

Fundamental to our considerations of FORTH arithmetical operations is the storage of numbers during these processes. To understand R.P.N. better we need to understand the stack, which may be regarded as the most important number reservoir.

To begin our analysis of the stack, imagine that you have typed in the following line.

1 2 3 4 <ENTER>

The numbers have been entered into memory in a last in first out form, which may be regarded as a column of numbers (or a stack), thus:

Top	4
	3
	2
Bottom	1

You can prove the stack contents at anytime by inserting the command .S in your calculations at any appropriate point. I give you the definition of .S without comment, since we haven't yet covered the concepts supporting it. Just type in the following line correctly and press enter. You can then use .S.

```
: .S CR 'S S0 @ 2- DO I @ . -2 +LOOP ;
```

The line came from page 50 of Starting FORTH.

Numbers at the top of the stack are most accessible for arithmetic operations and those at the bottom least accessible.

The stack can have, in principle, any depth, but in practice depth is limited. The stack can contain sufficient numbers for almost any legitimate purpose, so don't worry about overfilling it! This is not true of course, if you generate excess numbers which pile up on the bottom of the stack until a "stack overflow" error occurs. FORTH demands that you provide any necessary protection against stack overflow. Incidentally, to empty a stack full of unwanted data, type some meaningless word, e.g. sss <ENTER> and the stack will be cleared.

If you were next to enter an operator, it would operate on the top two numbers and place the result on the top of the stack. For example, typing * <ENTER> would result in the product of 4 and 3 being placed on the top of the stack. The stack would then "look" like this:

12
2
1

You may of course have a need to operate on numbers lower down in the stack while leaving the upper numbers intact. There are a certain special FORTH words which permit manipulation of the stack, that is, number positions in the stack may be altered to facilitate calculations. This means, of course, that the required numbers are brought to the top of the stack.

Some useful words are DUP, ROT, SWAP, OVER and DROP. These are not all of the stack manipulation words, but are most of the single length stack operators. We'll worry about double length operators later in this series. The words are easily understood. Let me list their functions below:

DUP — DUPlicates the top number on the stack so that two copies exist at the top of the stack. For example, if we commenced with a blank stack and typed 2 DUP <ENTER> the stack would contain two 2s.

ROT — ROTates the third number down in the stack to the top, e.g.

3		1
2	becomes	3
1		2

SWAP — SWAPs the top two numbers. e.g.

3		2
2	becomes	3
1		1

OVER Copies the second stack number and puts the copy on top of the stack e.g.

```
3                 2
2      becomes    3
1                 2
                  1
```

DROP DROPs the top number on the stack, e.g.

```
2                 3
3      becomes    2
2                 1
1
```

With these thoughts in mind we can try one or two simple calculations. Remember throughout that this is integer arithmetic (no decimal parts to numbers).

Let's try $3*(10)^2$. The line

```
3 10 DUP * * .
```

will do the job nicely. Stack contents look like this as the calculation progresses. I have written the operation preceding the stack condition above the gap between the stack listings

```
3 10     DUP     *       *

        10       10     100    300    300
         3       10       3           on
                  3                  screen
```

We would do well to remember that as simple single length numbers (16 bit) the stack values or intermediate results of calculations cannot exceed the range +32767 to -32768. I sense you recoiling in horror at such a profound limitation. Calm down, it's not as bad as it first appears.

FORTH provides ways of getting around these difficulties. Let's begin to talk about them by discussing the word */ (pronounced star-slash). This very useful word permits a single length calculation to have a double length (32 bit) intermediary value. It first multiplies and then divides. Like certain other division operators in FORTH the / part of */ leaves only the quotient. Care should therefore be taken when deciding the process of a calculation to avoid large errors due to dropped remainders.

Let's try an example to illustrate what we've learned so far. Calculate

$(14^2 + 12^2)^2 36$

The value of $(14^2 + 12^2)^2$ is more than 16 bits can handle, so the */ will come in useful. Entering the numbers in reverse order, the stack looks like this:

```
14
12
36
```

To make the calculation, do DUP * SWAP DUP * + DUP */ . . The stack contents will progressively look as follows.

```
   DUP      *      SWAP     DUP      *       +

14      14     196      12      12     144
12      14      12     196      12     196
36      12      36      36     196      36
        36                              36
```

```
   DUP      ROT     */

340     340      36     3211       3211
 36     340     340                on screen
         36     340
```

Another example. Find the effective resistance of three resistors (10, 20 and 30 ohms) in parallel. The formula for this is

$$R_T = (R1^{-1} + R2^{-1} + R3^{-1})^{-1}$$

We might expect the value of this expression, using these numbers to be a about half the lowest value of resistor (10 ohms). In fact the correct answer is 5.45 ohms. lets see how FORTH does using integer arithmetic.

Since it doesn't matter which order we process the resistance values we can enter them in any order, thus: 10 20 30 <ENTER>. But how to process them? If we simply divide each value into 1 to achieve a reciprocal we will be confounded by the loss of the quotient refered to earlier.

A solution is to scale each value and ultimately descale the answer. One has to be very careful doing this in problems involving multiplication or exponentiation.

If I divide a resistance value into 1000, the result will be 1000 times greater than otherwise. I'm not trying to insult your intelligence here; Graham would, I know, prefer these tutes written on the K.I.S.S. principle.

From here on there are a couple of ways to tackle the problem. One involves entering the scaling factor of 1000 as required: the other involves setting a constant equal to 1000 and calling up the constant when needed. Since we will tackle constants later in this series, I will restrict myself to the first option.

We have already entered the resistance values. The operations on the stack proceed as follows: 1000 SWAP / 1000 ROT / ROT 1000 SWAP / + + 1000 SWAP / . . The last period is of course the sentence ending. The stacks go as follows.

```
   /     ROT     1000     SWAP     /     + +

  20     50      30      1000     30     33
1000    100      50        30   1000     50
 100     30     100        50     50    100
  30                          100          100
```

```
   1000     SWAP     /

183     1000     183   5     Prints on
         183     1000         screen
                               5
```

continued on Page 57

WHAT'S NEW

THE PROBE

The Probe is our latest piece of equipment for use with either the Colour Computer or the T1000.

Attached to a CoCo, The Probe uses either the right hand joystick port or CoCoMax to provide temperature monitoring.

On a T1000, The Probe connects to the right hand joystick port.

The Probe consists of a cable and a protected transducer, which is inserted into the substance being monitored.

The Probe is not able to cope with highly acidic solutions or temperatures in excess of 100 degrees C.

Temperature readings however, can be taken between -40 and 100 degrees Celcius, with an accuracy of +/- 2 degrees using the T1000 or CoCoMax, +/- 8 degrees read directly through the CoCo joystick port, or +/- 2 degrees C through the CoCo joystick port with an optional amplifier (price to be announced).

The maximum specified thermal response time of The Probe in still air is 3 minutes (time taken to reach final temp value). However preliminary testing shows that this varies from 50 to 180 seconds.

The Probe is supplied with three programs on tape or disk (please specify) for CoCo or with a disk containing three programs for the T1000.

These programs are meant as thought starters for your own projects, but never the less provide some interesting information.

"BARTEMP" * is a program which provides you with a simple thermometer on screen. As the temperature of the substance being checked changes, so the reading on screen changes.

"LOGGER" * samples temperature over a specified period and provides a print out of the result.

The information below shows the result of a test we made, when we immersed The Probe in a glass of ice water, and after 220 seconds, took it out again.

The third of these programs, "GRAPH", # samples temperature over a set number of periods, for whatever length of time you require, and draws a graph of the results.

* 16K CoCo's and T1000 128K+
# 16K ECB CoCo's and T1000 128K+

There are a number of uses for this interesting piece of equipment.

It can be used to demonstrate heating and cooling rates to students; it can be used to monitor temperatures in controlled environments, eg hot houses, chicken sheds, fire alarms; and it can be used by experimenters.

Owners of The CoCoConnection will find The Probe of special interest, for it adds to your arsenal of control equipment.

The price is $49.95 for the versions without an amplifier. The version with amp is likely to be $59.95.

Graham.

TIME= 0 SECS	TEMP= 34	TIME= 170 SECS	TEMP= 6	TIME= 340 SECS	TEMP= 30
TIME= 10 SECS	TEMP= 34	TIME= 180 SECS	TEMP= 6	TIME= 350 SECS	TEMP= 30
TIME= 20 SECS	TEMP= 36	TIME= 190 SECS	TEMP= 6	TIME= 360 SECS	TEMP= 30
TIME= 30 SECS	TEMP= 34	TIME= 200 SECS	TEMP= 6	TIME= 370 SECS	TEMP= 32
TIME= 40 SECS	TEMP= 18	TIME= 210 SECS	TEMP= 6	TIME= 380 SECS	TEMP= 30
TIME= 50 SECS	TEMP= 12	TIME= 220 SECS	TEMP= 6	TIME= 390 SECS	TEMP= 32
TIME= 60 SECS	TEMP= 10	TIME= 230 SECS	TEMP= 6	TIME= 400 SECS	TEMP= 32
TIME= 70 SECS	TEMP= 10	TIME= 240 SECS	TEMP= 10	TIME= 410 SECS	TEMP= 32
TIME= 80 SECS	TEMP= 8	TIME= 250 SECS	TEMP= 16	TIME= 420 SECS	TEMP= 32
TIME= 90 SECS	TEMP= 8	TIME= 260 SECS	TEMP= 18	TIME= 430 SECS	TEMP= 34
TIME= 100 SECS	TEMP= 6	TIME= 270 SECS	TEMP= 24	TIME= 440 SECS	TEMP= 36
TIME= 110 SECS	TEMP= 6	TIME= 280 SECS	TEMP= 28	TIME= 450 SECS	TEMP= 38
TIME= 120 SECS	TEMP= 6	TIME= 290 SECS	TEMP= 28	TIME= 460 SECS	TEMP= 38
TIME= 130 SECS	TEMP= 6	TIME= 300 SECS	TEMP= 28	TIME= 470 SECS	TEMP= 38
TIME= 140 SECS	TEMP= 6	TIME= 310 SECS	TEMP= 30	TIME= 480 SECS	TEMP= 38
TIME= 150 SECS	TEMP= 6	TIME= 320 SECS	TEMP= 30	TIME= 490 SECS	TEMP= 36
TIME= 160 SECS	TEMP= 6	TIME= 330 SECS	TEMP= 30		

continued from Page 55

The answer, 5 ohms, is about 92 % of the correct answer. This is within the normal production tolerance of commercial resistors. If one required a more accurate answer, using a scale factor of 10 000 would be O.K..

Did you see where I did the descaling? It was of course in the final factor of 1000.

There are many ways of doing this calculation better than this; I hope the FORTH experts out there will not shudder too much. My intention was merely to provide a sample of some stack manipulation. Later in the series I will investigate some of those more efficient methods; in the meantime you would be best served by reading Starting FORTH. There you will see a wide variety of commands which I cannot, for sake of space and time, discuss here.

One point of re-assurance that I find comforting is that its O.K. to experiment on the computer. A learning experience, successful or not costs no more than a little time and a few cents of power. Your investigations of FORTH are therefore nothing more than a low-cost investment in your understanding of another aspect of programming.

Finally, remember - if you would like to talk about FORTH, feel free to call me on (07) 208 7820.

Getting Started with BASIC09

By Richard A. White

You have been reading about spreadsheets in this column for over six months. I know from letters that many of you have been following the articles and are using spreadsheets now. However, broad as the topic may be, there are other interests to be served.

Over the past few months, I have been learning and programming in BASIC09. Dale Puckett is right on target in praising BASIC09. I hope the tutorials I provide in the coming months will bring you to a similar conclusion.

Under OS-9, there is a wealth of programming languages. assembly language is the same as under Disk BASIC, except for the program/machine interface. Brian Lantz has recently covered Assembly language under OS-9 in THE RAINBOW. A series on PASCAL by Dan Eastham graced the pages of "RainbowTech" some time ago. While the very good DEFT Compiler was highlighted, Radio Shack's Microware PASCAL Compiler is likewise a powerful compiler under OS-9. Another programming language available under OS-9 from Radio Shack is Microware's C Compiler, possibly one of the very best C compilers for a microcomputer.

With these options, why program in BASIC09? BASIC09 is designed for structured programming, but so are PASCAL and C. C is very powerful and terse, although it is not particularly easy to learn. If you have done some assembly language programming and want higher programming productivity while staying close to the machine, C is probably for you. I know one programmer who held out for years before trying C and then wondered why he had been so bull-headed for so long.

Then there are the rest of us, many who have been programming under BASIC and want to do more under OS-9. BASIC09 provides a nice transition language since the key words you have learned under BASIC have the same meanings under BASIC09. Some of the pains in BASIC, like line numbers, are gone (you can still use them, but it's like shooting oneself in the foot). The automatic handling of variables you are used to is gone, but it is replaced by powerful structuring options that provide economy, clarity and speed.

BASIC09 provides a rich selection of control structures, including LOOP . . . ENDLOOP, REPEAT . . . UNTIL, WHILE . . . DO . . . ENDWHILE and EXITIF . . . THEN . . . ENDEXIT, in addition to FOR . . . TO . . . NEXT and IF . . . THEN . . . ENDIF. Both the IF and ENDIF constructions permit the use of ELSE. For those who have no previous experience with some of these structures, take my word that they are indeed nice to have.

If you are a fan of disk files, you will find BASIC09 different from the Microsoft BASIC in the CoCo, perhaps a tad more complex, but also more powerful. With one statement such as PUT #PATH, myfile, you can save a whole 10,000-byte data structure to disk as a machine code dump, which is fast and easy. Load it back with GET #PATH, myfile, which is as equally easy and fast.

BASIC09 is designed to encourage writing program modules. Subroutines are named (no more wondering what GOSUB 33 means), stored separately from other modules and run by name. This can save memory since only those procedures being used need be in memory. Such is not the case with PASCAL, where procedures must be defined and be in memory before they can be used. Variable names are local to modules, that is, you can use the same variable name in a number of modules to mean similar things. For example, "count" in the main program does not change when "count" in a subroutine is incremented, unless you specifically write the program so that happens. Long variable names are significant so that "count1" is different from "count2."

Perhaps I have left the status of PASCAL as a fuzzy choice. In fact, the choice between PASCAL and BASIC09 under OS-9 is not easy. One reason to choose PASCAL is that its programs can be compiled to machine language. Those who intend to take high school or college programming courses should opt for PASCAL simply because most institutions encourage beginning students to take PASCAL. Further advanced placement tests are standardized on PASCAL.

For the rest of us, the transition from BASIC to BASIC09 is easier than the transition to PASCAL. And, "byteheads" will probably take the next step to PASCAL as well.

To get started, buy the BASIC09 Compiler ($99.95) from your Radio Shack store. Also, buy *The Official BASIC09 Tour Guide* by Dale Puckett, which is now available from Radio Shack. Then, follow the rest of this article step by step.

When BASIC09 first appeared, the cry went up, "It won't load!" It was a directory problem. Under OS-9, if you just type the procedure name, that procedure must be in memory or in your CMDS directory to load and execute. The BASIC09 disk comes without a CMDS directory. Therefore, the very first order of business is to copy BASIC09 from the distribution disk to the CMDS directory on a system disk or to the CMDS directory on another disk if you are running two or more drives. Unlike C, BASIC09 does not require more than one disk drive.

There are two filenames on the distribution disk, *BASIC09* and *RUNB*. *BASIC09* includes the interpreter, an editor that is nice for a line editor, which as a class, ranges

from poor to horrible, and a debugger. *RUNB* is a special interpreter that runs packed BASIC09 modules. *RUNB* is half the size of *BASIC09*, and packed BASIC09 modules are a quarter to a third smaller than their source code. Packed modules run faster, though unpacked ones are fast compared to Color BASIC. Downstream, you will need to have *RUNB* in the CMDS directory along with packed BASIC09 modules.

I am back to a practice I follow under Disk BASIC. I make up a single disk for an application that includes the program modules and my working files. This way, I stick the disk in, boot the application, load any work files and go. This works even better under OS-9 where I have a special start-up file that automatically boots the application, setting a large memory buffer automatically.

I make an OS-9 system disk that includes *OS9boot* and its various utility procedures in the CMDS directory. Some of the procedures that come with OS-9 are never used with BASIC09 and should be discarded while making your BASIC09 system disk. There are a number of ways to approach this project, depending on how fancy you want to get. We will show an easy way, then refer to some more complex approaches that utilize more of OS-9's power.

Making a Single Drive BASIC09 Disk

First, format a disk. Type FORMAT /D0 at the OS-9 prompt. OS-9 loads the FORMAT modules and asks if you are ready. Now remove your system disk from Drive 0 and insert a fresh disk. Press 'Y' and that disk will be formatted.

The second step is to back up a system disk onto the newly formatted one. Exact examples of how to do this are shown on pages 63 and 64 of your red *OS-9 Commands Manual* for either two-drive or single-drive systems, so I won't duplicate them here. At this point, put your original system disk safely away and work with the backup. Type FREE and you will see that of 630 sectors: 84 are free and the largest block is 81 sectors (when using Version 01.01 of OS-9). This is on a 35-track disk.

Though 84 sectors is not much to work with (particularly when we have yet to copy *BASIC09* and *RUNB* onto the disk), there is quite a bit on the disk you will never need with BASIC09, so housecleaning is in order. We can start with the CMDS directory, which starts out like this.

```
Directory of /DØ/CMDS 11:36:4Ø

asm       attr      backup
binex     build     cmp
cobbler   copy      date
dcheck    debug     del
deldir    dir       display
dsave     dump      echo
edit      exbin     format
free      ident     link
list      load      login
makdir    mdir      merge
mfree     os9gen    printerr
procs     pwd       pxd
rename    save      setime
shell     sleep     tee
tmode     tsmon     unlink
verify    xmode
```

ASM is the assembler, which you won't need, so take it out. BINEX and EXBIN are worthless here, too, so pitch them. CMP is useful to see if files are identical, but also not worth keeping here. DCHECK and DEBUG can both go, and you can dump DUMP. LOGIN and TSMON are for use with an attached terminal intended for the multi-user mode, so take them both out; TEE can go, too.

Here is the easy way to clean house. Know that when deleting files from your data directory, you only need to specify the filename and not the whole pathlist. All the files listed here are in the CMDS directory. Therefore, make the CMDS directory the data directory by typing CHD /D0/CMDS. Now just a few typed lines will clean things up.

```
OS9>DEL ASM BINEX CMP DCHECK DEBUG DUMP
OS9>DEL LOGIN TEE TSMON
OS9>CHD /D0
```

After some OS-9 commands, a number of filenames may be entered up to the input buffer limit. The advantage of this is both reduced typing and speed since DEL is loaded only once to work on a number of entries.

Next we turn our attention to the DEFS files. These are only used with the assembler we just deleted, so we do the same with the DEFS files and their directory.

```
OS9>CHD /D0/DEFS
OS9>DEL OS9Defs RBFDefs SCFDefs SysType defsfile
OS9>DELDIR DEFS
```

At this point, OS-9 asks a cogent question to which I answer 'Y' and the directory goes away as well. Don't forget to do a CHD /D0 to return your data directory to D0.

Now we can copy *BASIC09* and *RUNB* to our commands directory. I suggest you type LOAD COPY to save a bit of time. Single drive copy works like single drive backup in terms of disk swapping and ready messages. The command lines should look like this:

```
OS9>COPY /D0/basic09 /D0/CMDS/basic09 -s #30K
OS9>COPY /D0/runb /D0/CMDS/runb -s #30K
```

The "-s" tells OS-9 you want a single drive copy and "#30K" allocates 30K bytes of memory buffer for the job. The larger the memory buffer, the fewer times you need to swap disks.

Now our CMDS directory looks like this:

```
Directory of /d1/cmds   21:39:33
BASICØ9   attr      backup
runb      build     cobbler
copy      date      del
deldir    dir       display
dsave     echo      edit
format    free      ident
link      list      load
makdir    mdir      merge
mfree     os9gen    printerr
procs     pwd       pxd
rename    save      setime
shell     sleep     tee
tmode     unlink    verify
xmode
```

And we still have some room on the disk, as FREE will show.

continued on P 61

VERY BASICALLY OS9

by Jack Fricker

OK. This month we are going to get back to the basics of OS-9 for those who are new to OS-9 and try to answer the questions most commonly asked.

In this article I am going to assume that you have just been to TANDY and bought a copy of OS-9 and BASIC09 and have just arrived home with them. You open the boxes look at the 3 disks and 4 manuals and say to yourself where do we start. You may even think that this is going to be TOO HARD to learn so why bother. You probably thought much the same when you first bought your computer home but you managed to learn how to use it didn't you? If you answered no you should go back and learn the basics of how to use it before you start on OS-9.

Well if you haven't been discouraged and have read this far that was the end of the hard sell. We are now going to get back to the point when you have opened the box .

The first thing to do is back up the master disk. First place a write protect tab over the write protect notch. The write protect notch is the small square missing from the side of the diskette. If you cover this with a piece of tape or a sticker then nothing can be written onto the disk.

Before you can backup the disk you must first get OS-9 up and running. If your Disk Basic has the DOS command (OS9 on some) all that you need to do is type DOS or OS9 when you have inserted the System Master Diskette into drive 0.

If your Disk Basic doesn't have either of these fear not, you have also been catered for. There should be a diskette marked Boot Disk in the package with the System Disk. If you do a DIR of this disk there should be two files on there, the one called "*" is the one that we want. Insert this in drive 0 and type RUN"*" and press the enter key. The screen will clear and ask you if you wish to check the speed of the drives or if you wish to boot OS-9. Answer OS-9 and you are on your way.

The loading and executing of OS-9 takes a little time. The screen should clear and the words OS9 boot should appear in the centre of the screen.

The next message that should appear will look something like this. It will be different for each version of OS-9.

```
OS-9 Level One
RS VERSION 01.01.00
COPYRIGHT 1980 BY MOTOROLLA
AND MICROWARE SYSTEMS CORP.
REPRODUCED UNDER LICENSE
TO TANDY CORP.
ALL RIGHTS RESERVED

yy/mm/dd hh:mm:ss
TIME ?
```

The prompt is asking you to enter the date and time. When you enter this it will start the SYSTEM CLOCK and print the date and time at various times. It will also set the date on any files so that you know for instance the last time a file was edited or when it was created. The CLOCK is more important than you might think because what we are about to do will use the date of the system clock.

The next thing to do is to make 2 backup copies of the master system disk. Why 2 you may ask? Its because you are going to modify one of them and if you make a mistake you will need the other disk to boot OS-9 again.

The first step in making a backup system disk is to format a blank disk (that makes sense doesn't it?). I am going to take you step by step through the procedure to format and backup using single or twin disk drives.

Although OS-9 will work quite well on a single drive two drives are better and will make these procedures easier and faster. Anyway, the first step in formatting a blank disk in drive 0 is to type "format /d0". Leave the distribution master disk in drive 0 while you are typing this. This is the reason that I said to place a write protect tab on the disk. After pressing enter the following will appear.

```
Format /d0 y/n
```

At this point you remove the distribution disk and insert the blank unformatted disk into drive 0 and press Y for yes. Just to make sure you really wish to do this it will ask you if you are ready. If you again type Y then it is too late to back out, that disk is about to be erased and formatted. The reason that you are asked twice is because formatting will erase any data on the disk that you may not wish erased.

After a short time you will be asked for a volume name. This name can be up to 32 characters long. This name will be written to the disk, but since we will be backing up onto this disk you can type anything you like in answer to this question. Backup will write the name of the disk that is being copied over the name you type in.

After you type your response & press enter you will see a number of HEXADECIMAL numbers on the screen. These numbers are the sectors being put on the disk. At the end of these numbers you will see the total number of sectors is $276. Repeat this procedure for the other disk.

Now that we have a couple of blank formatted disks maybe we should put something on them.

If you only have one disk drive you must re-insert your master system disk into drive 0 & type BACKUP /d0. The Computer will respond with Ready to backup /d0. If you type 'y' you will be asked to insert the destination disk in drive 0. Then you will be asked if disk so & so is the one you want.

If you have 2 disk drives insert the distribution system disk in drive0 and the new disk in drive 1, Type BACKUP & the prompt will read "Ready to backup from /d0 to /d1 y/n". When you type 'y' you will be asked if disk name(the name you entered before) is the one you want.

The reason that you are asked if you really wish to backup onto the disk is simple. When a backup of one disk onto another is done any and all information on the disk being copied to is writtem over including the name and date of creation. This is the reason that I said earlier when we created the disk not to worry about giving the disk a proper name. The creation date is the only thing that is not copied over from the original disk, it is given the current date instead.

OK. Now that you have 2 backups of your Master Distribution disk. From now on we'll call them your System disk and Backup System disk. Put your distribution disk away in a safe place because we should never need to use it again.

Instead you will use your newly created system disk. Put the system disk in drive 0 and type CHD /D0. Chd means CHange Data directory and /d0 means drive 0 (of course). Next type CHX /D0/CMDS. Chx means CHange eXecution directory. /cmds is the commands directory which is where a list of all the commands and where on the disk they are stored.

The chd and chx are commands but you will not find them in the commands directory because they are part of OS-9 and are in memory.
The /d0 and /d0/cmds are called pathlists. Pathlist is a word you will have to get used to when using OS-9 because I will be using it and the Manuals make constant use of it.

Parameter is another word that has to be learned. What it means is something that has to be passed from one thing to another. In this case pathlist is a parameter that is passed to the calling program chx or chd.

All these are mentioned in the manuals so I have mentioned them to help you when you read them. They contain all the things that you need to know so read them very carefully and fully.

If you have any questions about the user groups or the public domain software you may contact me c/- the Australian Rainbow or P.O.Box 306 Clayfield 4011.

continued from P 59

"COLOR COMPUTER DISK" created on: 85/11/17
Capacity: 630 sectors (1-sector clusters)
193 Free sectors, largest block 129 sectors

One last item to attend to is the start-up file.

```
TMODE .1 -UPC
XMODE /P BAUD=4 -UPC
PRINTERR
SETIME </TERM
LOAD BASIC09
EX BASIC09 #12K </TERM
```

TMODE .1 -UPC sets the terminal for upper- and lowercase. My experiments indicate this must be in all start-up files when lowercase is desired. It can be issued from the keyboard as well. Lowercase is used in BASIC09.

XMODE /P BAUD=4 -UPC sets the printer output to 2400 Baud. Use '2' for 600 Baud and '3' for 1200 Baud. PRINTERR enables OS-9 to get error messages from the disk replacing numbers; it's nice to have. SETIME </TERM gets the date and time entries from the keyboard. Without </TERM, SETIME looks for input from the disk and the system locks up. This is an important point that some of us have learned through much suffering. A start-up file is set up to interact with the disk drive unless instructed otherwise.

We load *BASIC09* (the program) so we can go back and forth between OS-9 and BASIC09 (the language) without having to load *BASIC09* each time. Finally, *BASIC09* is executed using EX, which saves forming another shell. >/TERM must be included or BASIC09 looks for its input from disk.

At this point you have one disk that automatically boots into BASIC09 and has space for some program files. However, we have not optimized memory. On booting, OS-9 loads a standard set of modules. A few of these won't be used and can be tossed, making room for either larger program files in memory or providing space for other modules such as DIR, which you might want in memory. Elsewhere in this issue, Donald Dollberg discusses optimizing your OS-9 boot to tailor in-memory modules to just those you want. His article, "Creating OS-9 System Disks," starts on Page 224.

When you have finished customizing your BASIC09 working disk, keep it as a master disk and make backups for working disks. When you start a new programming project, it's a good idea to start with a new working disk, as well.

Next month, we will pop our completely customized BASIC09 disk in the drive and start programming.

JUNK FILTER

by Ross McKay

(Many people appreciated seeing Rosko's program in the magazine last month. These simple, easy to follow programs are a great starting point for new OS-9'ers, and we'd encourage anyone who is not an expert to forward for publication their 'little' OS-9 progs.

OS-9 is proving to be a very enjoyable environment to work in. Many of you are a bit daunted at the prospect of starting something new - especially when it looks a bit complex.

But if you give it a go, you'll like it - so get into it! G.)

I've already had some calls from some people following the program last month - thanks, it was nice to hear from you!

Here is a little itility written in Basic 09 which will filter out 'junk' characters from text files.

These include line feeds and any graphics and control characters.

I wrote this to filter my BBS dumps, which ran to many pages with the extra line feeds.

```
PROCEDURE filt
(* FILT  1.0
(* by --- Rosko ! ---

(* usage: <run> filt("filename")
(* will remove non-cr's outside
(* range chr$(32)-chr$(127)
(* requires OS-9 Rename

PARAM file1:STRING
DIM infile,outfile:INTEGER
DIM char,lf:STRING[1]
lf:=CHR$(10)

OPEN #infile,file1:READ
CREATE #outfile,"temp":WRITE
WHILE NOT(EOF(#infile)) DO
GET #infile,char
IF char>CHR$(31) AND char<CHR$(128) OR char=CHR$(13) THEN
PUT #outfile,char
PUT #1,char
ENDIF
IF char=CHR$(13) THEN
PUT #1,lf
ENDIF
ENDWHILE
CLOSE #infile
CLOSE #outfile
DELETE file1
SHELL "rename temp "+file1
```

Hint . . .

Formatting Diskettes

Some CoCo users have reported problems with formatting more than one disk in succession. The CoCo uses a technique called "write precompensation" on the more critical inner tracks of a disk; for some reason, Disk BASIC doesn't turn off the feature after a DSKINI is completed. If you need to format more than one disk at a time, enter POKE 113,0 and press the Reset button after each disk.

Hint . . .

Slow Scrolling through Orange

Here's a powerful little POKE that slows your scrolling by creating a horizontal LIST. Type POKE 359,60 and you'll see what we mean. Add a colon (:) and SCREEN0,1 and you'll be slow-scrolling across an orange screen. To return to the green screen at full tilt, just type POKE 359,126.

AUSTRALIAN RAINBOW

index

The Best of CoCoOz:

#1 . . . EDUCATION programs. Fourteen programs for the teacher or parent.
#2 . . . Part 1 . . . 16K GAMES. (Mainly ECB). Sixteen programs to keep you on your toes!
Part 2 . . . 32K GAMES. Adventures, simulations and arcade games for everyone!
#3 . . . UTILITIES. Programs to make your use of the computer easier. Spoolers, Reverse Video, Disk utilities and more.
#4 . . . BUSINESS programs. Invocing, Accounts, Creditors and more.

Per Tape	$10.00
Per Disk	21.95
Two Disks or more each	$16.00
#3 on Disk	$16.00
Any two tapes	$17.00

What's on:

Best of CoCoOz #1. Education.

ROADQUIZ . . . . ROB WEBB
HANGMAN . . . . ALEPH DELTA
AUSTGEOG . . . . P. THOMAS
SPELL . . . . IAN LOBLEY
FRACTUT . . . . ROBBIE DALZELL
ICOSA . . . . BOB WALTERS
TAXMAN . . . . TONY PARFITT
MARKET . . . . ALEPH DELTA
TOWNQUIZ . . . . ROB WEBB
ALFABETA . . . . RON WEBB
TANK ADDITION . . . . DEAN HODGSON
TABLES . . . . BARRIE GERRAND
KIDSTUFF . . . . JOHANNA VAGG
FLAGQUIZ . . . . ROB WEBB

Best of CoCoOz #2 part 1. 16K Games.

TREASURE . . . . DAVISON & GANS
MASTERMIND . . . . GRAHAM JORDAN
ANESTHESIA . . . . MIKE MARTYN
OREGON TRAIL . . . . DEAN HODGSON
ADVENTURE . . . . STUART RAYNER
SHOOTING GALLERY . . . . . TOM DYKEMA
GARDEN . . . . DAVE BLUHDORN
YAHTZEE . . . . KEVIN GOWAN
BATTLESHIP . . . . CHRIS SIMPSON
ANDROMIDA . . . . MAX BETTRIDGE

Best of CoCoOz #2 part 2. 32K Games.

LE-PAS . . . . Wrongsoft
COCOMIND . . . . STEVE COLEMAN
OILSLICK . . . . JEREMY GANS
CCMETEOR . . . . BOB THOMSON
BATTACK . . . . JEREMY GANS
PROBDICE . . . . BOB DELBOURGO
CHECKERS . . . . J & J GANS
PYTHON . . . . ?
POKERMCH . . . . . GRAHAM & MATTHEWS
SPEEDMATH . . . . DEAN HODGSON
LNDATTCK . . . . ALDO DEBERNARDIS
INVADERS . . . . DEAN HODGSON
RALLY . . . . TONY PARFITT
FOURDRAW . . . . JOHANNA VAGG

Best of CoCoOz #3. Utilities.

PAGER . . . . ?
HI . . . . ALEX. HARTMANN
SPOOL64K . . . . WARREN WARNE
CREATITL . . . . BRIAN FERGUSON
FASTEXT . . . . OZ-WIZ
DATAGEN . . . . ROBIN BROWN
SPEEDCTR . . . . PAUL HUMPHREYS
PRNTSORT . . . . PAUL HUMPHREYS
BIGREMS . . . . BOB T
DIR . . . . PAUL HUMPHREYS
COPYDIR . . . . THOMAS SZULCHA
LABELLER . . . . J.D. RAY
SCRPRT . . . . TOM DYKEMA
MONITOR + ... BRIAN FERGUSON
BEAUTY . . . . BOB T
PCOPY . . . . B. DOUGAN
RAMTEST . . . . TOM DYKLEMA
DISKFILE . . . . B. DOUGAN
LABEL . . . . F. BISSELING

Best of CoCoOz #4. Business.

HI . . . . ALEX. HARTMANN
(Disk Directory manager)
BANKSTAT . . . . BARRY HATTAM
(Statement annal & store)
INSURE . . . . ROY VANDERSTEEN
(Analyse home contents)
SPOOL64K . . . . WARREN WARNE
(Printer spooler req 64K)
2BC . . . . WARREN WARNE
(Hold 2 sep progs in mem)
DATABASE . . . . PAUL HUMPHREYS
(THE tape database)
RESTACC . . . . DUNG LY
(Tape restruant accounts)
PRSPDSHT . . . . . . GRAHAM MORPHETT
(Disk print out SPDSHEET)
PERSMAN . . . . PAUL HUMPHREYS
(Personal finance management)
CC5 . . . . GRAHAM MORPHETT
(Sales Invoicing-tape sys)
COCOFILE . . . . BRIAN DOUGAN
(Tape data base)
DPMS . . . . PAUL HUMPHREYS
(Disk Program Management Sys)
4OKGREY . . . . RAY GAUVREAU
(40K Basic for grey 64K CoCo)
TAXATION . . . . ?
(Calc tax payable)
SPDSHEET . . . . GRAHAM MORPHETT
(Disk 22 coloum spreadsheet)
ACS3 . . . . GREG WILSON
(Multi disk data base)

Next:

Best of CoCoOz #5. Adventure Games Due: April '86
Best of CoCoOz #6. Preschool Education Due: June '86

SUBSCRIPTION FORM

GOLDSOFT. P.O. BOX 1742, SOUTHPORT, 4215.

AUSTRALIAN RAINBOW $24.75
☐ 6 months $39.95
☐ 12 months $39.95

AUSTRALIAN CoCo/Softgold
☐ 6 months $19.00
☐ 12 months $31.00

RAINBOW ON	☐ TAPE	☐ DISK
☐ 6 months	$ 81.00	$ 81.00
☐ 12 months	$144.00	$172.00
☐ deb monthly	$ 15.00	$ 15.00

CoCoOz ON	☐ TAPE	☐ DISK
☐ 6 months	$ 42.00	$ 58.00
☐ 12 months	$ 75.00	$102.50
☐ deb monthly	$ 9.50	$ 10.95

MicoOz	☐ TAPE
☐ 6 months	$ 42.00
☐ 12 months	$ 75.00

Additional Requirements:

. . . .
. . . .
. . . .

Sub No:
Name:
Address:
P.C.
Phone No.:
Credit Card No.:

☐ Bankcard ☐ Visa ☐ Mastercard

Authorised AMT $ . . . .

OFFICE USE ONLY:
Received:-

User Group Contacts

(Stop between numbers = b.h. else a.h.; but, hyphen between = both.)

ADELAIDE	JOHN HAINES	08 278 3560
NORTH	STEVEN EISENBERG	08 250 6214
BRIGHTON	GLENN DAVIES	08 296 7477
GREENACRES	BETTY LITTLE	08 261 4083
MORPHETTVALE	KEN RICHARDS	08 384 4503
PORT NOARLUNGA	ROB DALZELL	08 386 1647
SEACOMBE HTS	GLENN DAVIS	08 296 7477
STURT	MARY DAVIS	08 296 7477
ALBURY	RON DUNCAN	060 43 1031
ARMIDALE	DANIEL CLARKSON	067 72 8031
BAIRNSDALE	COLIN LEHMANN	051 57 1545
BALLARAT	MARK BEVELANDER	053 32 6733
BIGGENDEN	ALAN MENHAM	071 27 1271
BLACKWATER	ANNIE MEIJER	079.82.6931
BLAXLAND	BRUCE SULLIVAN	047 39 3903
BOWEN	TERRY COTTON C/O	077 86 2220
BRISBANE:		
BIRKDALE	COLIN NORTH	07 824 2128
BRASSALL	BOB UNSWORTH	07 201 8659
EAST	ROB THOMPSON	07 848 5512
IPSWICH	MILTON ROWE	07 281 4059
PINE RIVERS	BARRY CLARKE	07 204 2806
SOUTH WEST	GRAHAM BUTCHER	07 376 3400
SANDGATE	MARK MIGHELL	07 269 5090
SCARBOROUGH	PETER MAY	07 203 6723
BROKEN HILL	TERRY NOONAN	080 88 2382
BUNDABERG	RON SIMPKIN	C/O TANDY
CAIRNS	GLEN HODGES	070 54 6583
CAMDEN	KEVIN WINTERS	046.66.8068
CANBERRA NTH	JOHN BURGER	062 58 3924
CANBERRA STH	LES THURBON	062 88 9226
CHURCHILL	GEOFF SPOWART	051 22 1389
COFFS HARBOUR	BOB KENNY	066 51 2205
COOMA	ROSS PRATT	0648 23 065
COORANBONG	GEORGE SAVAGE	049 77 1054
COOTAMUNDRA	CHERYL WILLIS	069 42 2264
DALBY	ANDREW B. SIMPSON	074.62.3228
DARWIN	BRENTON PRIOR	089.81.7766
DENILIQUIN	WAYNE PATTERSON	058 81 3014
DUBBO	GRAEME CLARKE	068 89 2095
EMERALD	LEIGH EAMES	059 68 3392
FORBES	JOHANNA VAGG	068 52 2943
FORSTER	GARY BAILEY	065 54 5029
GLADSTONE	CAROL CATHCART	079 78 3594
GOLD COAST	GRAHAM MORPHETT	075 51 0015
GOSFORD	PETER SEIFERT	043 32 7874
GOULBURN VALLEY	TONY HILLIS	058 59 2251
GRAFTON	PETER LINDSAY	066 42 2503
GUYRA	MICHAEL J. HARTMANN	067 79 7547
HASTINGS	MICHEAL MONCK	059 79 2879
HERVEY BAY	LESLEY HORWOOD	071 22 4989
HOBART	BOB DELBOURGO	002 25 3896
JUNEE	PAUL MALONEY	069 24 1860
KALGOORLIE	TERRY BURNETT	090.21.5212
KINGSTON	WIM DE PUIT	002 29 4950
KEMPSEY	RICK FULLER	065 62 7222
LISMORE	ROB HILLARD	066 24 3089
LITHGOW	DAVID BERGER	063 52 2282
MACKAY	LEN MALONEY	079511333x782
MAFFRA	MAX HUCKERBY	051 45 4315
MAITLAND	LYN DAWSON	049 49 8144
MARYBOROUGH	NORM WINN	071 21 6638
MELBOURNE:		
DANDENONG	DAVID HORROCKS	03 793 5157
DONCASTER	JUSTIN LIPTON	03 857 5149
FRANKSTON	BOB HAYTER	03.783.9748
NARRE WARREN	LEIGH EAMES	03 704 6680
NTH EASTERN	KEVIN KAZAZES	03 437 1472
MELTON	MARIO GERADA	03 743 1323
RINGWOOD	IVOR DAVIES	03 758 4496
SUNBURY	JACK SMIT	03.744.1355
MOREE	ALF BATE	067 52 2465
MORWELL	GEORGE FRANCIS	051 34 5175
MT ISA	PAUL BOUCKLEY-SIMONS	077 43 6280
MUDGEE	BRIAN STONE	063-72-1958
MURGON	PETER ANGEL	071 68 1628
NAMBUCCA HDS	WENDY PETERSON	065 68 6723
NARROMINE	GRAEME CLARKE	068 89 2095
NEWCASTLE	LYN DAWSON	049 49 8144
NOWRA	ROY LOPEZ	044 48 7031
ORANGE	STEVE LOVETT	063.62.4025
or	JIM JAMES	063 62 8625
PARKES	DAVID SMALL	068 62 2682
PENRITH	ALEX SCHOFIELD	047 31 5303
PERTH	IAN MACLEOD	09 448 2136
PORT LINCOLN	BILL BOARDMAN	086 82 2385
PORT MacQUARIE	RON LALOR	065 83 8223
PORT PIRIE	KEVIN GOWAN	086 32 1368
ROCKHAMPTON	KEIRAN SIMPSON	079 28 6162
SALE	BRYAN McHUGH	051 44 4792
SHEPPARTON	ROSS FARRAR	058 25 1007
SMYTHESDALE	TONY PATTERSON	053 42 8815
SPRINGWOOD	DAVID SEAMONS	047 51 2107
SWAN HILL	BARRIE GERRAND	050.32.2838
SYDNEY:		
BANKSTOWN	CARL STERN	02 646 3619
BNKSTWN WEST	ARTH PITTARD	02 72 2881
BLACKTOWN	KEITH GALLAGHER	02-627-4627
CAMPBELLTOWN	LEO GINLEY	02 605 4572
CARLINGFORD	ROSKO MCKAY	02 624 3353
CHATSWOOD	BILL O'DONNELL	02 419 6081
or	MARK ROTHWELL	02 817 4627
CLOYTON	HERMAN FREDRICKSON	02 6236379
HILLS DIST	DENIS CONROY	02 671 4065
HORNSBY	ATHALIE SMART	02 848 8830
KENTHURST	TOM STUART	02 654 1610
LEICHHARDT	STEVEN CHICOS	02 560 6207
or	GORGE ECHEGARAY	02 560 9664
LIVERPOOL	LEONIE DUGGAN	02-607-3791
MacQUARIEFLDS	KEITH ROACH	02 618 2858
ROSEVILLE	KEN UZZELL	02 467 1619
SUTHERLAND	IAN ANNABEL	02 528 3391
SYDNEY EAST	JACKY COCKINOS	02 344 9111
TAMWORTH	ROBERT WEBB	067 65 7256
TAHMOOR	GARY SYLVESTER	046 81 9318
TARA	STEVEN YOUNGBERRY	
TONGALLA	TONY HILLIS	058 59 2251
TOOWOOMBA	GRAHAM BURGESS	076 30 4254
TOWNSVILLE	JOHN O'CALLAGHAN	077 73 2064
TRARALGON	MORRIS GRADY	051 66 1331
UPPER HUNTER	TERRY GRAVOLIN	065 45 1698
URALLA	FRANK MUDFORD	067 78 4391
WAGGA WAGGA	CES JENKINSON	069 25 2263
WHITEROCK	GLEN HODGES	070 54 6583
WHYALLA	MALCOLM PATRICK	086 45 7637
WONTHAGGI	LOIS O'MEARA	056 72 1593
WYNYARD	ANDREW WYLLIE	004 35 1839
WYONG	JOHN WALLACE	043 90 0312
YARRAWONGA	KEN SPONG	057 44 1488

SPECIAL INTEREST GROUPS

BUSINESS

BRIZBIZ	BRIAN BERE-STREETER	07 349 4696

OS9 GROUPS

BANKSTOWN	CARL STERN	02 646 3619
BRISBANE	JACK FRICKER	07 262 8869
CARLINGFORD	ROSKO MCKAY	02 624 3353
COOMA	FRED BISSELING	0648 23263
KALGOORLIE	TERRY BURNETT	090.21.5212
SYDNEY EAST	JACKY COCKINOS	02.344.9111
SYDNEY NTH	MARK ROTHWELL	02 817 4627

MC-10 GROUPS

LITHGOW	DAVID BERGER	063 52 2282
ORANGE	DAVID KEMP	063 62 2270
PORT LINCOLN	BILL BOARDMAN	086 82 2385
ROCKHAMPTON	TIM SHANK	079 28 1846
SYDNEY	RAJA VIJAY	02 519 4106
WARRNAMBOOL	GARY FURR	055 62 7440

TANDY 1000 / MS DOS

BRISBANE		
NORTH	BRIAN DOUGAN	07 30 2072
SOUTH	BARRY CAWLEY	07 390 7946
GOLD COAST	GRAHAM MORPHETT	075 51 0015
MELBOURNE	TONY LLOYD	03 500 0878
SYDNEY	ROGER RUTHEN	047.39.3903
WYONG	JOHN WALLACE	043 90 0312

FORTH

BRISBANE	JOHN POXON	07 208 7820
PORT LINCOLN	JOHN BOARDMAN	086 82 2385
SYDNEY	JOHN REDMOND	02 85 3751

ROBOTICS

BOWEN	TONY EVANS	077 86 2220
GOLD COAST	GRAHAM MORPHETT	075 51 0015
TAMWORTH	ROBERT WEBB	067 65 7256
WAGGA WAGGA	CES JENKINSON	069 25 2263

AUSTRALIAN RAINBOW MAGAZINE
REGISTERED BY AUSTRALIAN POST —
PUBLICATION NO. QBG 4009
AUSTRALIAN COCO/softgold
REGISTERED BY AUSTRALIAN POST —
PUBLICATION NO. QBG 4007.
PO BOX 1742,
SOUTHPORT. QLD. 4215.

In
Australian CoCo
This Month:

* Music!
* A Driving Test!
* Expert Systems.
* Four Education Progs!
* Sale of the Century!
* M-Masters (a game).

* plus all your favourite regular columns!

POSTAGE PAID AUSTRALIA