Star Attractions:

FILEMASTER 2 programs: FORMAT & RETRIEVAL comprise a powerful data file manager. Great for everything from phone lists to legal abstracts. Needs 32K. Design your own data structure. Up to 500 characters per record. Up to 15 searchable fields in any combination. On Disk. ...................................................... $34.95

SPACE Multi-faceted simulation of life in interstellar society. You and opponents must make life & death decisions. Keeps track of your progress from one game to next. Needs 48K and Aplesoft ROM. Disk .............................................................. $29.95

ADVENTURE Fight off pirates and vicious dwarves. 700 travel options, 140 locations, 64 objects. Needs ROM & 48K. Disk. .............................................................. $29.95

16K CASETTE INVENTORY Use item number, description, stock amount, reorder amount, restock date, cost & sell price. Holds up to 140 items. Tape .............................................................. $35

32K DISK INVENTORY: Use stock numbers description, vendor, record of purchase and sales date, amount on hand, cost & sell price, total value. Holds up to 300 items. Disk .............................................................. $40

32K DATA BASE Cross file for phone lists, bibliographies, recipes. Run up to 9 lines of 40 columns each. Search by item anywhere. Disk .............................................................. $50

24K HI-RES LIFE SIMULATION Conway’s equations on 296x180 screen. A mathematical simulation to demo population growth with birth, death and survival as factors. Tape .............................................................. $10

16K CIRCUIT LOGIC DEVELOPMENT AID Evaluate circuits of up to 255 gates, including AND, OR, NOR, NAND, XOR, XNOR and INVERTER. Tape .............................................................. $10

16K MORSE CODE TRAINER Learn Morse Code, and transmit or receive over radio. Tape .............................................................. $10

18K DEVIL’S DUNGEON: Adventure through dark passages where monsters, demons, poisonous gas, dropoffs threaten . . . all to discover fantastic treasures. Comes with instruction book. Tape .............................................................. $10

16K PACIFICA: Discover the floating island and rescue the beautiful princess. To win you must recover the enchanted crown, but you face the threat of magic spells and demons. Tape .............................................................. $9.95

Don’t see what you’ve been looking for, here? Then write for our FREE SOFTWARE CATALOG. We’re saving one just for you!

To order software, add $2 shipping. To transfer tape versions to disk add $5. California residents add 6% sales tax.

We’re saving one just for you!

Rainbow’s Casino: 9 gambling games: Roulette, Blackjack, Craps, Horserace, and a few originals that Vegas hasn’t heard about. Needs 16K. Tape .............................................................. $29.95

16K SPACE WAR: You in your space capsule battle against the computer’s saucer . . . in hi-res graphics. Tape .............................................................. $12

16K MEMORY VERIFY Diagnostic routine to check range of memory. Indicates faulty addresses, data in memory cell, and faulty data. Tape .............................................................. $5

16K APPLEODION Music synthesis composes original Irish jigs. Enter your own music and save on tape or disk. Includes 3 Bach fugues. Tape .............................................................. $10

16K APPLEVISION Demo for Hi-Res graphics and music. Tape .............................................................. $10

32K COMPU-READ 5 programs to teach you speed reading, in stages. Includes synonym and antonym identification. You control your rate of speed, or keep up with the computer’s pace. Tape .............................................................. $24.95

48K PERCEPTION I, II, III random shapes and sizes must be matched. In III, you control format and display time and get weighted scores. Needs ROM. Each Disk .............................................................. $24.95

32K STORY TELLER Use your bizarre imagination and input key words for fantastic and funny tales. Never the same story twice. Tape .............................................................. $12.95

32K WAR/RESCUE Engage in 10 battles with your infantry against the Apple robots. Calculate Apple’s strategy and win more battles than the computer. Tape .............................................................. $12.95

24K POLAR PLOT Plot polar equations in Hi-Res Graphics. Tape .............................................................. $10

32K SHAPE SCALER Utility to generate and animate Hi-Res graphic shapes. Simple routine provided to inspect position of shapes, and specify precise X/Y coordinates and scale. Needs ROM. Disk .............................................................. $13.95

32K ZINTAR/PROPHET Great party game. Under control of the mighty Zintar’s edict you take a very special trip to the world of Krintar. Heightened visual graphics. Needs ROM. Disk .............................................................. $16.95

APPLE MONITOR PEELED Everything you wanted to know about the Apple Monitor but couldn’t figure out. User-written manual in plain English clears your confusion. Only .............................................................. $9.95

Rainbow Computing Inc.

Garden Plaza Shopping Center, Dept. 11A
9719 Reseda Blvd., Northridge, CA 91324
Telephone: (213) 349-5560
RUN THIS PROGRAM

10 Enter data in form below
20 Goto mailbox
30 Mail form
40 Receive the Personal Computer Catalog
5D End

Well Done!

Follow this simple program and you will receive The Personal Computer Catalog. The one reference book to fine quality personal computers, software, supplies and accessories. This valuable catalog is FREE so mail your order today.

Name ____________________________________________________________
Address __________________________________________________________
City ___________________________ State _______ Zip __________
Do you own a computer? _______What type? __________________________
Do you use your computer for: Business? ______________
Personal? ____________Education? ____________ Other? ____________

Mail this form to:

Weldon Electronics
4150 Hillcrest Road
Wayzata, MN 55391

Or phone: (612) 884-1475
Attention “Old” Pet™ Owners:

Not sure about the ROM Retrofit Kit from Commodore? Now you can use both sets of Commodore ROMs and others as well.

The Basic Switch allows switch selection of either ROM set (your original set or your retrofit set) from Commodore. Plus, Model 15-A includes an additional zero insertion force socket allowing easy use of ROM like the BASIC Programmer's Toolkit... concurrently.

Model 14-E The economy model of The Basic Switch. Stand alone board and harness without case and case hardware. The free standing unit is ready to accept your ROMs.

Model 14-D Same as Model 14-E but includes attractive protective case and mounted Basic Switch board.

Note that Model 14 Series does not allow for expansion ROMs like the BASIC Programmer's Toolkit.

Model 15-A The Basic Switch plus... includes expanded cable assembly and zero insertion force socket. Your 15th ROM simply plugs in... enabled while either ROM set is selected. Socket 15 may be readdressed by the user for additional flexibility.

The Basic Switch is sold in assembled form only. All models are designed for easy attachment to your Pet with a convenient cable assembly. No soldering or drilling is required. The Basic Switch mates with a cable assembly at your primary board, and does not use the physical connectors of any Pet ports.

Model 15-A allows you to use the BASIC Programmer's Toolkit without the need for the additional $25.00 board or tying up your ports. And since we've designed the 15th socket to be readdressable, watch for more ROM packs later in the Fall.

The Basic Switch:

<table>
<thead>
<tr>
<th>Model</th>
<th>Base Price</th>
<th>Retrofit Kit</th>
<th>Programmer's Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 14-E</td>
<td>$64.95</td>
<td>$149.95</td>
<td>$149.95</td>
</tr>
<tr>
<td>Model 14-D</td>
<td>$77.95</td>
<td>$162.95</td>
<td>$149.95</td>
</tr>
<tr>
<td>Model 15-A</td>
<td>$99.95</td>
<td>$184.95</td>
<td></td>
</tr>
</tbody>
</table>

Model 15-A with installed ROM Retrofit and BASIC Programmer's Toolkit: $229.95

Model 15-A with installed ROM Retrofit and both Toolkits: $274.95

“Old” Pets were shipped with 24 or 28 pin ROMs. You must check which you have, and specify at time of order.

Pet™ is a trademark of Commodore Business Machines, Inc. of Santa Clara, Calif. The BASIC Programmer's Toolkit is a product of Palo Alto IC's, A Division of Nestar Systems, Inc. North Carolina residents add 4% sales tax. All orders add $2.50 shipping.
Table of Contents

Nicer Writer
by Rick Conolly

Disassembling the DOS 3.2
by William Reynolds

Hooking PET to Ma Bell
C. H. Scanlon

Spelunker
by Thomas R. Mimlitch

6522 Timing and Counting Techniques
by Marvin L. De Jong

Card Shuffling Program for KIM-1
by Hark Chan

How Do You Connect Peripherals to Your Superboard II
by Bruce Hoyt

The MICRO Software Catalog: XIII
by Mike Rowe

Hypocycloids
by E. D. Morris

SYM-1 6532 Programmable Timer
by Robert A. Peck

Letters

A Real-Time Clock for OSI Disk Systems
by Robert T. Kintz

6502 Bibliography: Part XIII
by Dr. William R. Dijf

Advertiser's Index

AB Computers
52

Beta Computer Devices
42

Computer Components
32,33

The Computerist, Inc.
25

Computer Shop
47

Connecticut microComputers
54

Electronic Specialists, Inc.
53

H. Geller Computer Systems
43

Hudson Digital Electronics
40

Micro Technology Unlimited
26

MUSE Software
4

Powersoft, Inc.
20

Programma International
14

Progressive Software
42

Pyrotec Programming
46

Rainbow Computing, Inc.
1FC

Saunders Marketing
63

SKYLINES: Electronic Works
52BC

Small Systems Services, Inc.
2

Softtech, Inc.
56

Softsys Software
IBC

SpectraSoft
56

SYNOptic Software
56

Texttech
57

Weldon Electronics
1

West Side Electronic
58

October 1979
Issue Number 17

Staff

Editor/Publisher
Robert M. Tripp

Assistant Editor
Mary Ann Curtis

Business Manager
Maggie E. Fisher

Circulation Manager
Carol Ann Stark

Production Assistant
L. Catherine Bland

Controller
Donna M. Tripp
SUPER-TEXT is a professional word processing system for the Apple II and Apple II Plus computers.

SUPER-TEXT is the most innovative word processor available on any personal microcomputer and includes features previously found only on word processing systems costing thousands more! An advanced multiple paging system allows you to view two text screens simultaneously. You may keep notes or instructions on one text screen while you edit on the other.

SUPER-TEXT is a character-oriented editor with complete cursor controls to easily move the cursor to any position in the text with a minimum of keystrokes. Built-in floating point math and automatic tabbing facilitate the preparation of all manual reports including financial reports, insurance forms, real estate settlements and more.

SUPER-TEXT is easier to operate than a typewriter yet challenges the flexibility of pencil and paper.

SELECTED FEATURES:
EDITING - Full floating cursor. Cursor control - left, right, up, down or to center of screen. Add or insert a character, word or line. Automatic carriage return eliminates a word breaking at the end of the screen line. Delete a character, word, line or screen. Automatic on-screen tabbing and right or left justification of columns. Unlimited text movement. Scroll either a page or a line forward or back. Move to the beginning or end of the text or screen. Move to the last change made in the text. Move to a block marker. Global search and replace (selective or all). Block operations - copy, delete or save to disk. Select multiple or single screen mode.

ADVANCED FILE HANDLING - Requires only two keystrokes to load or save a file to disk. The file name does not have to be entered which eliminates misspelling and "FILE NOT FOUND" errors. Save entire text or portion of to disk. Complete file merging capabilities. MATH - Automatic column totals. Formula computations. User selectable number of decimal positions. Automatically switches to scientific notation when necessary. 14 significant digits.


MICRO INFORMATION SYSTEMS® (48k) $99.95 is a breakthrough in effective information systems of any size. This system handles accounts payable/receivable, inventories, appointment calendars, cost estimating, real estate listings, sales solicitation, manpower accounting, selective mailings and label printing, dietary information, phone directories and more! On diskette.

U-DRAW II™ (32k) $95.95, a complete graphics package for the Apple II with disk. You can create a figure and store, expand, contract or move it anywhere on your video screen with a few simple keystrokes. Save individual figures or complete drawings on disk and recall them later. U-DRAW II automatically builds and edits multiple figure shape tables that are directly transferable to your BASIC programs. You won't find better graphics capabilities at 100 times the price!

APPLET EDU-DISK® (32k) $49.95 A complete multi-program C.A.I. system for the APPLE II includes program editor and APPLET interpretation on diskette with extensive on-line HELP lessons plus documentation manual.

THREE MILE ISLAND™ (48k) $99.95 Is the technology of a nuclear reactor too complex to handle? Now you have the opportunity to decide for yourself, with THREE MILE ISLAND, a realistic simulation of a pressurized nuclear reactor. Four spectacular displays monitor the containment building, turbines, filters, condenser, reactor core and the pump house. Values, pumps, turbines, filters and control rods are individually activated by keyboard command. The comprehensive documentation describes in detail the operating mechanisms and component interactions.

SUPER-LOAD CASSETTES
U-DRAW (16k) $17.95
ELECTRIC CRAYON (8k) $17.95
MAZE GAME (8k) $12.95
ESCAPE (16k) $12.95
SIDE SHOWS (8k) $12.95
TANK WARS (8k) $12.95
MUSIC BOX (8k) $12.95
BASEBALL (16k) $14.95
UNCLE SAM'S JIGSAW (32k) $12.95
GLOBE WAR (32k) $17.95

*Plus APPELSOF Board

MUSE CO., 7112 Darlington Drive, Baltimore, MD 21234
Order by phone (301) 681-8531 MASTERCHARGE and VISA welcome
Is screen wraparound a necessary fact of life? Or can the computer adapt to conventional line ending rules? This little BASIC output routine goes a long way toward eliminating wraparound once and for all.

Has this ever happened to you: A group of friends are admiring your expensive investment. With the flair of a true computer expert, you press the appropriate buttons, push RETURN, and tell the expectant guests to watch the screen. "Hi!", the computer prints. "Thanks for answering the questions as well as you did, Rick. I can state that you should live 55.215677 more years and have 2.15662 children."

You wait for the applause. Instead, you hear, "How come the words run off the end of the line?" Dead silence. You are embarrassed—for your guests, of course. Instead of seeing the brilliant output of your sophisticated program, your guests saw:

Hi! Thanks for answering the questions as well as you did. Rick. I can state that you should live 55.215677 more years and have 2.15662 children.

Now, you and I know that screen wraparound is a fact of life. Perhaps the program concatenated a bit. Or, possibly, it was adapted from an article written by some thoughtless author with a 64 column screen or an 80 column printer. In either case, you probably will soon tire of explaining that nothing is wrong with your magic machine; it just prints funny, sometimes.

This is the wrong approach! We don’t adapt to the computer’s idiosyncrasies; it adapts to ours. Right? The little subroutine at lines 35000 thru 35010 does a lot to help the wraparound problem. It is a human-oriented subroutine that prints on the screen using much the same rules we would use with a typewriter. Specifically,

It will break a line at a space, comma, period, colon, hyphen, or other character you specify.

If a word is longer than the allowable line, it will be hyphenated (rather arbitrarily, but this is a small subroutine).

At your pleasure, it will indent the first line of the output. This helps increase legibility.

Four variables control the output format. They may be entered once, at the beginning of a program, or they may be changed within the program if required. The variables used are:

CW Column Width. This specifies the maximum columnar width of your output device, and is used for error catching.

M1 Margin indent on the first line.

M2 Margin indent on subsequent lines. (Note: Left justification

M3 Margin indent on subsequent lines.

October, 1979

MICRO -- The 6502 Journal
35000 IF M3 - M1 > CW OR M3 - M2 > CW THEN PRINT "LINE TO LONG FOR PRINTER. ":PRINT :PRINT :END :REM CHECK FOR LINE LENGTH
35001 LN = LEN (NS); FOR I = M3 - M1 TO 1 STEP - 1:BPS = MID? (NS,I,1): IF BPS = " " OR BPS = OR BPS = " " OR BPS = " " OR LN < M3 - M1 THEN 35003:REM FIND BREAK POINT
35003 HTAB M1: PRINT LEFTS (NS, I); IF LN < = M3 - M1 THEN RETURN
35004 N1$ = RIGHTS (N1S, LN - I):
35005 IF LEFTS (N1S,I) = " " THEN LN = LEN (N1$) - 1:N1S = RIGHTS (N1S, LN):GOTO 35003:REM DELETE EXCESS SPACES
35006 LN = LEN (NS): FOR I = M3 - M2 TO 1 STEP - 1:BPS = MID? (N1S,I,1): IF BPS = " " OR BPS = " " OR BPS = " " OR LN < M3 - M2 THEN 35008:REM FIND BREAK POINT
35007 NEXT I. HTAB M2: PRINT LEFTS (N1S,M3 - M2 - I);:PRINT "- ":I = M3 - M2 - I:LN = LEN (N1S):GOTO 35009:REM HYPHENATE LONG WORD
35008 HTAB M2: PRINT LEFTS (N1S ,I):
35009 IF LN < = M3 - M2 THEN RETURN
35010 N1S = RIGHTS (N1S, LN - I): GOTO 35005
PRINTOUT OF STRING NS AS IT WOULD NORMALLY BE PRINTED FROM A PROGRAM:
THIS IS AN EXAMPLE OF A LONG SENTENCE THAT COULD COME EITHER FROM A PROGRAM WRITTEN FOR A 64 OR 80 COLUMN SCREEN OR PRINTER, OR FROM ONE THAT CONCATINATES. SUPERCALIFRAGALISTICEXPIALIDOCIOUS, NOT SUPERCALIFRAGALISTICEXPIALIDOCIOUS, NOT
Nicer printout of string NS:
"Why did it print 'COMPUTER'?"

The word "Supercalifragilisticexpialidocious" (Does anyone really know how to spell it) is entered in the string NS of the sample run to point out two characteristics of the nice print subroutine. In the first nice print example, the length of the word has forced it down one line, leaving the preceding line rather short. In the second example, where the word is longer than the allowable line length, super... is arbitrarily hyphenated. A short line should not appear too often with a 40 column line length, since most words are 10 letters or less in length.

Nicer writer is easy to incorporate into a program, and fast in execution. It will make your programs appear more professional and, best of all, it will keep your friends from asking questions like "Why did it print 'COMPUTER'?"
Disassembling the DOS 3.2

You "Can't tell the players without a score card" and you can not effectively use the Apple II DOS 3.2 without this important information on its organization.

On the surface, DOS 3.2 is identical to DOS 3.1. Upon booting, the DOS is loaded (slave or master), the greetings program is run, MAXFILES defaults to 3, and HIMEM is set at $9600. DOS 3.2 still communicates with the rest of the APPLE via input and output hooks at $36, 37, 38, and 39. (All addresses refer to a 48K machine.)

The differences are many: In Apple­soft, DOS does the call 3314 or call 54514 automatically, volume checking is ignored unless explicitly defined in the command, and the system defaults to NOMON C, I, O status. The hooks at $36 and 37 (the print routine) now contain $9EB1. The routine to restore DOS is now at $9DBF. This can be called if page 3 is overwritten. The command and error message tables are in different locations. The command table is the same as in the DOS 3.1. The error messages, however, are quite different. After a BLOAD, AS is now found at $AA72, 3; LS is now found at $AA80, 1.

When the keyboard input routine ($9E81), is called, DOS checks the mode. If it is in direct mode, the DOS reads the keyboard, then goes to the print routine. The print routine has seven routines of its own, 0-6. It calls the correct one, depending on whether the mode is direct, deferred, execute, read or write, etc. These routines are all inter-related.

In direct mode, when a return is detected, DOS attempts to match the string in the keyboard input buffer ($200-2FF) to a command in the table. In the print mode, direct or deferred, it stores all characters in the keyboard input buffer until a return is detected. If then checks for a CTRL-D as the first character. If not found, DOS drops out and returns control to wherever it came from. However, if Control D is detected, DOS attempts to match the string to the command table. If a match is not made, it prints "Syntax Error".

When DOS matches a command, it then checks for names, if needed, or numbers, if needed. After getting all data required, a check for optional data is made. After any optional data is read, numbers are changed to hex if need be, the maximum and minimum ranges are compared, then if all data is OK, the number is stored and DOS returns to check for any other optional data.

A routine gets the correct address from the stack, then executes the command. I have highlighted a few of the commands:

- RUN and IN# do the same function as in BASIC, except that DOS will set the hooks properly before releasing control.
- MON and NOMON set a mask at $AA74 as follows: 0 = monitor nothing, $10 = monitor 0, $20 = monitor I, $40 = C, and combinations thereof.
- MAXFILES resets HIMEM and PP (INT BASIC) and allocates a file buffer via a subroutine at $A7D4.
- BRUN does a BLOAD then a JMP ($AA72).

RUN does a load, then jumps to a routine which executes the program. Which routine is used is dependent upon which language is being used, BASIC, FP RAM, or FP ROM.

LOAD reads the file type and does either INT or FP as needed, then loads the program. When in FP mode, and after the program is loaded, DOS does the call 3314 or call 54514 as needed to set the program pointers for Applesoft.

FP attempts to find a ROM card and turn it on. If possible, it sets the return addresses via a routine at $9D84. If no card is found, the DOS runs Applesoft, then goes to a routine at $9DEA to set return addresses correctly.

INT makes certain the ROM card is off, then goes to $9D84 to set return addresses correctly.

If a person wishes to use DOS from a language or operating system not standard to the APPLE, there is no problem, unless an error is detected. If you do not wish an error message to cause a return to BASIC or Applesoft, the address at $9D5E and F can be changed for your particular system.

Whenever a change in language is done, DOS updates it's return address stack from the stack for that particular language. All commands except PR#, IN#, MON, NOMON, INT, FP (if in ROM), and MAXFILES go through routines that use file buffers.
All commands may be called from monitor or machine language, provided
(1) a language change is not needed, (2) the file names have been placed into the
name buffer(s), and (3) that any other
parameters have been properly placed
into their locations as needed.

The disk controller card contains two
(2) PROM's, 256 bytes each. One PROM
contains the program to start the
booting of the DOS. The other is used for
a program that, together with some
other IC's, actually controls the head
position, reading a bit, writing a bit, send­ing
the byte to the APPLE bus, and get­
ting a byte from the APPLE bus. The
following locations control the hardware
functions. Add 0050 to each address, S
= the slot number of the controller card.

C080-87 These addresses sequen­tially step the motor that
moves the head back and
forth. Odd addresses step
one way, and even ad­
dresses step the other way.

C088 Turns off the drive motor.
C089 Turns on the drive motor.
C08A Enables drive two.
C08B Enables drive one.
C08C,D Control connecting the AP­
PLE bus to the hardware for strobing the byte in or out of
the 74LS323 IC shift regis­
ter, depending upon the pre­
viously set status of
C08E,F.
C08E,F Read/Write control.

I have documented all routines, sub­
routines, buffers, and other locations to
the best of my ability in the memory
maps that follow. Notes tell the function
and usage of each. On most items I have
given only the starting address. The end
address is implied to be the next
documented location minus one. On
stacks of addresses, the parenthized
number is the number of addresses con­tained in that stack. Remember that any
two-byte items are always stored low
byte first. Documentation of addresses
in the B000-BFFF area may be in error
because that area got too complex for
me to retain my sanity.

My thanks to my family for their time
and patience, to other persons for their
articles on DOS functions, APPLE for
their excellent documentation, without
which I would have had no idea what
was going on, and to Terry and Kent at
Computerland of Portland, for use of
their printer to obtain 60 feet of hard
copy, and their moral support.

APPLE II DOS 3.2 Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>95FF</td>
<td>End of user RAM: HIMEM = 49151</td>
</tr>
<tr>
<td>9600</td>
<td>Start of data buffer</td>
</tr>
<tr>
<td>9700</td>
<td>Start of track and sector buffer</td>
</tr>
<tr>
<td>9800</td>
<td>Start of miscellaneous info buffer</td>
</tr>
<tr>
<td>982D</td>
<td>Start of name of file</td>
</tr>
<tr>
<td>984B,C</td>
<td>Address of start of miscellaneous info buffer ($9600)</td>
</tr>
<tr>
<td>984D,E</td>
<td>Address of start of track and sector buffer ($9700)</td>
</tr>
<tr>
<td>984F,0</td>
<td>Address of start of data buffer ($9600)</td>
</tr>
<tr>
<td>9851,2</td>
<td>Address of start of name buffer, next file ($0000 = no more files)</td>
</tr>
<tr>
<td>9853</td>
<td>Data</td>
</tr>
<tr>
<td>9953</td>
<td>Track and sector</td>
</tr>
<tr>
<td>9A53</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>9A80</td>
<td>Name</td>
</tr>
<tr>
<td>9A9E,F</td>
<td>Address of start of miscellaneous info buffer ($9A53)</td>
</tr>
<tr>
<td>9AA0,1</td>
<td>Address of start of track and sector buffer ($9953)</td>
</tr>
<tr>
<td>9AA2,3</td>
<td>Address of start of data buffer ($9653)</td>
</tr>
<tr>
<td>9AA4,5</td>
<td>Address of start of name buffer of next file down ($92D2)</td>
</tr>
<tr>
<td>9AA6</td>
<td>Data</td>
</tr>
<tr>
<td>9BA6</td>
<td>Track and sector</td>
</tr>
<tr>
<td>9CA6</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>9CD3</td>
<td>Name</td>
</tr>
<tr>
<td>9CF1,2</td>
<td>Address of start of miscellaneous info buffer ($9CA6)</td>
</tr>
<tr>
<td>9CF3,4</td>
<td>Address of start of track and sector buffer ($9BA6)</td>
</tr>
<tr>
<td>9CF5,6</td>
<td>Address of start of data buffer ($9AA6)</td>
</tr>
<tr>
<td>9CF7,8</td>
<td>Address of start of name buffer of next file down ($9A80)</td>
</tr>
<tr>
<td>9CF9 –</td>
<td>Unused</td>
</tr>
<tr>
<td>9CFF</td>
<td>Unused</td>
</tr>
</tbody>
</table>

Address of name of first file
DOS keyin routine address
DOS print routine address
Name number 1 buffer address
Name number 2 buffer address
Bottom of DOS
Address stack for the internal print routines (7)
Address stack for the DOS command routines (28)
Address stack for return to the current language (6)
Address stack for return to Integer BASIC
Address stack for return to Applesoft ROM (6)
Address stack for return to Applesoft Disk (6)
(3D3G) Control B, re-enters INT or FP (ROM only)
(3D0G) Restores DOS and re-enters current language
Restores $3D0 – $3FF from $9E51 – $9E80
Stack for the above routine
Keyboard input routine
Calls correct internal print routine, depending
upon mode
Restores keyboard and print hooks
Internal routine for information from the disk
Internal routine for printing
Prints and exits DOS
Keyboard input internal routine
Internal routine for sending information to disk
Routine to correct internal routine
Used by the EXEC command
Mask MON status, print and exit
A229  PR# routine
A22E  IN# routine
A233  MON routine
A23D  NOMON routine
A251  MAXFILES routine
A263  Start of DELETE routine
A271  Start of LOCK routine
A275  Start of UNLOCK routine
A27D  Start of VERIFY routine
A281  Start of RENAME routine
A298  Start of OPEN routine
A2EA  Start of CLOSE routine
A31I  BSANE routine
A35D  BLOAD routine
A38E  BRUN routine
A397  SAVE routine
A413  LOAD routine
A4D1  Run routine
A4E5  Runs Integer BASIC program
A4F0  CHAIN routine
A4FC  Runs FP ROM program
A506  Runs FP RAM program
A510  WRITE routine (set up)
A51B  Read routine (set up)
A54F  INIT routine
A56E  Catalog routine
A57A  FP routine
A59E  INT routine
A5C6  EXEC routine
A5DD  Position routine
A60E  Starts the read process
A626  Starts the write process
A644  Stores data coming from text file into keyboard buffer. Used by the EXEC command.
A65E  Error checking?
A679  Closes files, exits DOS
A682  Goes to hardware routines
A69D  Sets up address of name section of next file
A6AB  Close the buffer last used
A6C4  Prints, "SYNTAX ERROR"
A6C8  Prints, "NO BUFFERS AVAILABLE"
A6CC  Prints, "PROGRAM TOO LARGE"
A6D0  Prints, "FILE TYPE MISMATCH"
A6D5  Prints other error messages by message number contained in $AA5C
A71A  Moves parameters given to locations for use by hardware routines
A743  Moves name from the name buffer to the name section of the file buffer
A74E  Moves addresses of sections of file buffers to locations for use by hardware routines
A764  Attempts to find a file buffer already in use by the name given
A74F  Checks file type
A7C4  Sets up file buffers and addresses (used by MAXFILES)
A851  Restores DOS hooks
A884  Start of command table
A909  This is a table of two-byte masks. One byte is used to determine what type of extra data is needed by a command. The other byte is used by the hardware routines for what file type to create or look for.
A941  Table containing the letters V, D, S, L, R, B, A, C, I, O. This is used when checking for optional data.
A94A  Table of bytes for determining what type of optional data to look for.
A995  Table of minimum and maximum ranges for V, D, S, L, R, B, A.
A971  Start of error message table
AA3F  Relative address of start of error message, i.e. ($AA55,1)
AA4F,50  Address of name section of next available file buffer
AA51  Internal print routine number
AA52  PR# hooks out of DOS
AA53,4  IN# hooks out of DOS
AA55,6  Number of total file buffers
AA57  Number of file buffers not in use
AA59  Temporary storage used by various routines
AA5E  Mask for MON and NOMON
AA5F  Command number
AA60 – 61  Found $ from a BLOAD
AA62 – 65  Temporary storage used by various routines
AA66,7  Defined volume number
AA68,9  Defined drive number
AA6A,B  Defined slot number
AA6C,D  Defined length
AA6E,F  Defined record number
AA70,1  Defined byte number
AA72,3  Defined address
AA74  Start of file name buffer number 1
AA75  Start of file name buffer number 2
AA81  Control D
AA82  Mode (direct, deferred, etc.)
AA86  Value used for language, e.g. INT = 0, FP RAM = C0, FP ROM = 80
AA87  The name, "Applesoft"
AA88  Address of start of IOB (used by RWTS)
AAC1,2  Address of start of buffer for track/sector list (used by RWTS)
AAC3,4  Address of start of buffer for data (used by RWTS)
AAC5,6  Top of total RAM in the APPLE II
AAC7,8  Address stack for hardware routines (14)
AAC9  Address stack for hardware routines (6)
October, 1979

AAF1  Address stack for hardware routines (6)
AAFD  Goes to the correct hardware routine
AB28  Reads VTOC and reads directory attempting to find an entry with the same name as the one given. If not found, checks the table of masks to see if it is allowed to create a file. If it may, it does so, and if not, it exits with "FILE NOT FOUND" or "LANGUAGE NOT AVAILABLE"
ABDC  Clears miscellaneous info hardware buffer, sets volume number, drive number and slot number.
AC06  Close routine. Updates VTOC, track bit map, and sector count of directory entry as needed.
AC3A  Rename routine. Finds directory entry, stores new name in entry, then writes that directory sector back to disk.
AC5B  Goes to correct hardware routine
AC70  Goes to correct hardware routine
AC87  Sets parameters for following routine
AC9A  Actually reads text file
AC93  Sets parameters for following routine
AC96  Reads program or binary file
ACA8  Puts byte being read into buffer
ACBB  Sets parameters for following routine
ACBE  Writes into text file
ACC7  Sets parameters for following routine
ACCA  Writes program or binary file
ACDA  Gets byte being written from buffer
ACEF  Lock hardware routine
ACF6  Unlock hardware routine
AD12  Sets parameters for following routine
AD18  Verify hardware routine
AD2B  Delete hardware routine
AD54  Part of delete routine, frees sectors used by deleted file.
AD98  Catalog hardware routine
AE42  Part of catalog, prints the number in $44 as three digit ASCII.
AE6A  Moves miscellaneous info from the file buffer to the hardware buffer.
AE7E  Moves miscellaneous info from the file buffer to the hardware buffer.
AE8E  Initialize hardware routine
AF08  Sets 42 and 43 as pointers to sections of the file buffer
AF1D  Writes data section of file buffer to disk
AF34  Writes track/sector list section of file buffer to disk
AF4B  Sets hardware pointer to the track and sector list section of the file buffer being used
AF5E  Checks position in file. If out of current sector, reads/writes next sector, updates VTOC buffer, updates track/sector list section of file buffer if in write mode.
AFDC  Reads from disk into data section of file buffer
AFE4  Sets hardware pointers to data section of file buffer being used
AFF7  Reads VTOC to its buffer ($B3BB - B4BA)
AFFB  Writes VTOC from its buffer
B011  Reads a directory sector into its buffer ($B4BB - B5BA). Initially reads sector A, successive entries into this subroutine read successive sectors from the disk. When all sectors have been read and the subroutine is called again, it will merely exit with the carry set.
B037  Writes current directory sector from buffer to disk.
B052  Sets up IOB for directory sectors, goes to RWTS
B0A0  End of above if no error
B0A1  Start of error handling routine for above
B0B6  Checks position in file, reads/writes next sector as needed
B134  Initializes data section of file buffer to all zeroes
B15B  Sets next position in file
B194  Increments position in file
B1A2  Sets next RAM address
B1B5  Calculates how much RAM is left
B1C9  Reads VTOC and successive entries, attempting to find the specified file name.
B21E  Puts name of file into directory
B224  Sets next sector, updates VTOC buffer
B2C3  Updates VTOC
B2DD  Calculates track bit map for VTOC
B300  Sets/checks parameters for file?
B35F  Routine with different entry points to exit the hardware routines with error
B397 - A6  Temporary storage for hardware routines
B3A7 - AA  T, I, A, B Used by catalog for file types
B3AB,C In reverse order, the string, "DISK VOLUME"
B3AD - BA  VTOC buffer
B3BB  VT0C buffer
B4BB  Directory buffer
B5BB - D0  Temporary storage for hardware routines
B5D1 - FF  Miscellaneous info section of currently used file
B600  Buffer. Purpose?
B700  Reads drive 1, current slot, $B1 sectors, track 0, sector A into RAM starting at $1B00. Boot routine?
B74A  Writes $0A sectors, starting from $B600, then $1B sectors, starting at $1B00, beginning at track 0 sector 0.
B793  Increments track/sector as needed and data address for above two routines
B7B5  Calls RWTS, checks status upon return
B7C2  Sets address of data buffer, and sets expected volume number
B7DB  Stores zeroes in one page, starting at the address in $42, 43
B7E7  Start of IOB and device characteristics table
B800  Part of RWTS?
BA90 - FF  Temporary storage for RWTS?
BB00  One-page buffer (RWTS?)
BC00  One-page buffer (RWTS?)
BD00  Start of RWTS
BFD4  End of RWTS
BFD5  Various endings sections for the hardware routines
BFFF  End of RAM
Hooking PET to Ma Bell

The dream of many microcomputerists to use their system as a terminal connected to a large computer system can become a practical reality. The $50.00 hardware for any 6502 based system, and the software for a PET, are fully described.

Having worked with my 8K PET for almost a year, I have become hooked on microcomputers and am enjoying learning and experimenting with a great machine. Like most microcomputer enthusiasts, I dream of more memory, disks, printers, etc. However, attempting to raise a family on a teacher's pay means that I have limited funds. So I wired up a PET to RS-232 modern interface, plugged into a modem, and bingo — by dialing up the computer system on the campus of Arkansas State University, I have all of these plus much more hooked to my PET. If you have telephone access to a computer system or a friend with an answer modem on his computer, here is the hardware and software to get you started communicating on the telephone.

The interface shown in Figure 1 can be built for under $50 including connectors, wiring, etc., and can be plugged into any RS-232 modem (I have a U.S. Robotics Model 310 which lists for $149). A TTL compatible modem can be wired directly to pins 2 and 6 of the MC6850. All the parts, except the crystal, are fairly common and can be ordered from most mail order electronics parts firms. The 1.229 megaHertz crystal can be ordered from any crystal manufacturer for around $10. This interface can be connected to any 6502 or 6800 based microcomputer that allows direct access to the microprocessor bus, for example, the APPLE, KIM, SYM, SWT, OSI, etc.

The software is written in BASIC and makes the PET act like a TTY type "dumb" terminal. The control characters are obtained by using the shift key. For example, control S is simply shift S. Although this program appears to limit the PET, it really doesn't since you can hit the stop key, write and run a program in the extra RAM and get back to the terminal program with a RUN 190 or a GOTO 190. For example, you could write a BASIC program starting at line number 500, compute a bunch of data, POKE the data to the modem, and then return to the terminal program with a GOTO 190.

Software

The MC6850 Asynchronous Communications Interface Adapter (or, in the buzz words of computerland, simply speak the letters A-C-I-A) is located in page B and has multiple addresses. I use hex BFF6 = 49142 as the address to POKE to the control register and to PEEK at the status register. Address BFF7 = 49143 is used to POKE a byte to the modem and to PEEK at a byte from the modem.

The BASIC program provides directions for the operator, data transfer from the modem to PET, data transfer from PET to the modem, and miscellaneous programming needs.

Lines 101 - 105 POKE a machine language routine into the second cassette buffer, and line 110 POKEs the

```plaintext
10 REM TERMINAL PROGRAM
20 REM BY C.H. SCANLON
30 REM STATE UNIVERSITY, ARKANSAS
50 REM 72467
105 FOR I = 826 TO 861: READ X: POKE I, X: NEXT
110 POKE 537, 58: POKE 538, 3
115 POKE 49142, 3
120 POKE 59468, 14
130 PRINT "(cs) * ** TERMINAL * * * *"
140 PRINT "(cd)(cd) Type RUN 190 but don't hit the return yet".
150 PRINT "(cd) Dial 935-9372 and wait for the tone".
160 PRINT "(cd) Place receiver in holder and hit return".
180 STOP
190 POKE 49142, 129
195 FOR I = 1 TO 30: NEXT: POKE 49143, 7
200 GET AS: IF AS = "": GOTO 200
210 IF ASC(AS) = "shift S" THEN PRINT "(cs)"
215 IF ASC(AS) < 192 GOTO 300
220 IF ASC(AS) > 244 GOTO 300
225 POKE 49143, ASC(AS) - 192: GOTO 200
300 POKE 49143, ASC(AS): GOTO 200
```

NOTE: (cs) means clear screen and (cd) means cursor down.

Figure 2

October, 1979 MICRO -- The 6502 Journal

Charles H. Scanlon
P.O. Box 22
Arkansas State University
State University, AR 72467
address of this routine into the interrupt address location of RAM so that, when an interrupt occurs, PET will check the ACIA before it checks the other possible interrupt locations.

When the ACIA receives a serial byte from the modem, it strips off the start and stop bits, converts it to parallel, and then interrupts the CPU. PET is then routed to the routine beginning at 033A.

Lines 033A - 033C transfer the contents of the ACIA status register to register X and lines 033D - 033E cause a branch if bit 7 is set, indicating that the ACIA did interrupt the CPU and has a byte to transfer. Otherwise, lines 033F - 0341 transfer PET to the ROM interrupt sequence. Lines 0342 - 0344 transfer a data byte from the ACIA to register A and line 0345 clears the CPU interrupt to allow for other interrupts. Lines 0346 - 0347 strip the parity bit from the data byte and line 0348 transfers it to register X for temporary storage.

Next, to erase the cursor, lines 0349 - 034A load A with $20 (note that CHR$(32) is a blank). Lines 034B - 034D get the current position of the cursor on the video line and lines 034E - 034F then clear the cursor.

To type the character, line 0350 retrieves the data byte from register X and stores it in register A and lines 0351 - 0353 then types the character in the next print position.

To set the cursor, lines 0354 - 0355 load register A with $A0 (note that CHR$(160) = reverse blank), lines 0356 - 0358 get the current position of the cursor on the video line, and lines 0359 - 035A then set the cursor.

Lines 035B - 035D then transfer control back to the PET interrupt routine. Back in the BASIC program, line 115 POKEs 3 into the ACIA control register which then resets the ACIA. Line 120 sets the lower case letter mode and then lines 120 - 180 print instructions and stop.

Since the answer modem at Arkansas State University uses seven bits plus parity plus two stop bits, line 190 programs the ACIA to transfer data in this mode. Reference 1 explains how to program other modes. Also, since the Arkansas State University computer initially waits for a control G, line 195 has a delay and then POKEs a 7 = ASCII CTRL-G to the modem. Lines 200 - 300 then wait to get a character from the keyboard, convert the character to ASCII, and POKE it to the ACIA.

Hardware

The MC6850 is wired directly to the CPU bus through the memory expansion port. I use page B by wiring CS2 to SELB. Details of programming the ACIA can be found in reference 1.

The 1.229 megaHertz crystal and the C4060 counter put out a 4800 Hertz square wave to the ACIA. The ACIA further divides it by 16 to obtain a baud rate of 300. Reference 2 indicates how to get...
You will need a ±12 and +5 volt power supply. If you use a TTL compatible modem, you won’t need the ±12 volt supply and you can get +5 volts from the second cassette port.

Questions

There are lots of software questions I have not answered. For example, how can a program be copied directly from the cassette to the modem? How can a program or data file be “saved” by sending it to the storage facilities at the other end of the line and how can it be retrieved later? With the exception of displaying more characters, what can an expensive “smart” terminal do that PET can’t do? As I stated earlier, this article is merely a start.

References

1. An Introduction to Microcomputers
FORMAT

PROGRAMMA's FORMAT (Version 1.0) is a command oriented text processor designed to be fully compatible with PIE (PROGRAMMA Improved Editor).

FORMAT's system of imbedded commands (within the text) give it an ease of operation similar to text formatters found on some mini-computers.

FORMAT features right margin justification, centering, page numbering, and auto-paragraph indent.

The following commands are available with FORMAT:

.ad Begin adjusting right margins
.bp n Begin page numbered n
.br Cause a line break
.ce n Center next n lines without fill
.fi Start filling output lines
.fo t Foot title becomes t
.he t Head title becomes t
.in n Indent n spaces from left margin
.li n Literal, next n lines are text
.ll n Line length including indent is n
.ls n Set line spacing to n
.ml n Top spacing including head title
.m2 n Spacing after heading title
.m3 n Spacing before foot title
.m4 n Bottom spacing including foot title
.na Stop adjusting right margins
.nf Stop filling output lines
.pl n Page length is n lines
.pp n Begin paragraph .sp, .fi, .ti n
.sp n Space down n lines, except at top
.ti n Temporary indent of n
.ul n Underline next n input lines

All orders include 3% postage and handling.

Apple II is a registered trademark of Apple Computer, Inc.
California residents add 6% Sales Tax
VISA & MASTERCARD accepted.
Spelunker

Spelunker is not for the faint of heart! It presents many interesting and useful programming techniques in the pleasant format of a game. As you play, keep telling yourself "It's only a game, it's only a game..."

This is an adventure fantasy series in which you become directly involved in exploration of a mysterious cavern in southwest Kentucky called Devil's Delve. If you have never played before, you should take a guide along. The guide will read the chamber descriptions as you enter each room for the first time. He can also supply some hints and clues to help you when you are stuck. Only the guide should use the room descriptions, word lists, and the map of the caverns. However, younger players may need some of these aids to help them.

Spelunker is an interactive game. You must converse with the program in order to explore the caverns and locate their treasures. You can talk in sentences, if you wish; but the program will use only one verb and one noun to establish meaning. For this reason, it is best to converse in verb/noun phrases. In the case of moving from chamber to chamber, for example, enter "GO W" or simply "W" and the verb "GO" will be implied. The Spelunker program will move you into the next room to the west upon receiving this command. Other examples might include "TAKE LIGHT" or "JUMP DOWN".

With this brief introduction you should be ready to explore the caverns of Spelunker. While you are about it, try drawing a map of the cave. You may also wish to discover exactly what vocabulary is understood by the program. The material that follows is for the guide only — so don't ruin your first adventure by peeking at it.

For the Guide Only

In the 16K APPLE II version of Spelunker, the chamber descriptions are not part of the program because of limited memory size. These room descriptions have been prepared for the adventurer's guide. The guide may read each room description as the adventurer enters the chamber for the first time.

1. **Mouth:** You are at the mouth of a large cavern. The sides of the entrance slope steeply upward, and a mysterious passage leads west into the cave.

2. **Tree room:** A towering, withered tree stands in what appears to be a dried up river bed. From it you seem to hear echoing sounds saying, "Water...water...water..."

3. **Writing room:** Do not read this description if the room is dark. The writing room is a large, oval chamber with tall ceilings and massive stalagmites. The smooth eastern wall has some writing on it — cryptic characters that spell out, "THE SPIRITS OF THE FRUIT."

4. **Pit room:** A small chamber with an immense stalagmite hanging from the center of the ceiling, directly over the mouth of a bottomless pit.

5. **South lake shore:** You stand at the edge of a misty lake that stretches endlessly out before you to the north.

6. **West lake shore:** You are standing on a damp, sandy shoreline with a very low passage leading off to the west. A clammy draft issues from the low-ceilinged passage.

7. **North lake shore:** A small, sandy beach on the northern edge of Misty Lake.

8. **Maze room:** Also known as the Swiss cheese room. You lose your sense of direction because twisting passages are coming and going at all points of the compass.

9. **Frozen river room:** What appears to be a petrified river bed slopes gently upward leading toward he west. It has a low, four-foot ceiling.

10. **Swift river room:** You hear swiftly running water, as you enter this room, and you see a narrow, churning, underground river flowing to the south.

11. **Hub room:** A magnificently decorated chamber with crystalline designs and intricate rock formations. A narrow, fast moving river flows through the hub room.

12. **Ice room:** Mysteriously, ice forms very quickly in this chamber, encapsulating anything left there for too long. There is so much ice that you can't even get into the room; however, you see an exit on the other side of the chamber.

13. **Chimney room:** A small, smoke filled chamber with a fire burning in a natural fireplace in the north wall. Apparently, a chimney leads far up through the rock and out of the cavern.

14. **Gold room:** As you enter this room, the first thing that you notice is a pile of golden treasures nestled into a nook on the far side. Before you take another step, a foul-smelling ogre jumps out from a hole in the side wall and rushes forward to protect his gold.

15. **Bones room:** Lining the walls of this chamber are the skeletons of pirates long since dead. An ominous curse is uttered by all of the skeletons in unison, as you enter the room, and the curse shadows your travels throughout the cavern.

16. **Bat room:** The ceiling is all but invisible for the tens of thousands of bats sleeping there. In one corner of this room lies an old, rusted chest. As you open the chest, the bats begin to stir. Inside the chest is a king's ransom in jewels: diamonds, rubies and emeralds.

17. **Ghost room:** An eerie feeling of demonic power lurks in this chamber.

18. **Misty Lake:** You are in the middle of Misty Lake. A strange glow emanates from the bottom of the lake. You turn off your light and notice an enormous, bright pearl nestled inside a gigantic clam. The clam is at the bottom of the lake, in only ten feet of water.

19. **Swift River:** This narrow, fast flowing river is outside the cavern. It runs south for a few yards and then disappears underground.
Table 1: Sample word tables for the guide.

The following lists of verbs and nouns are for use if you are having difficulty in communicating with Spelunker. Not all of these words have meaning or utility in this adventure. I didn’t want to make it too easy!

**VERBS**

BITE CARRY CLIMB DIG DRINK DRIVE
DROP EAT FIGHT GO HELP HIT
JUMP KILL PUT RUB RUN START
STOP TAKE THROW USE WALK WISH

**NOUNS**

APPLE AX BATS BOMB BONES CAVE
CHEST CLAM CURSE DOWN E FIRE
GHOST GOLD ICE KNIFE LAKE LAMP
LIGHT N NW NE SW OGRE PEARL
RAFT RIVER ROPE S SE SW TENT TREE TRUCK UP W WATER

Having been exposed to a fantasy program called *Adventure* which seems to reside on many large timesharing networks, I was challenged to see if this type of game could be handled on a micro. Thus, the dream stage began. I thought up monsters, treasures, a cave structure, tools, tricks and battles. The major goals emerged:

Pseudo-English input commands (verb-noun phrases)

Interconnected rooms one could travel through

Objects one could take, put, carry and use

Monsters / treasures; do battle, take rewards

Secrets to be discovered

The obvious method was to tabularize as much data as possible so that similar functions could be implemented as subroutines. This left only special handling routines to be added.

The program was organized into five major sections. Lines numbered 30xxx initialize the tables and variables. Lines numbered 4xxx to 10xxx print out the current location and status for the player. Lines numbered 1xxx read and decode the input string. Lines in the 2xxx range perform the command action, if possible. In lines with 3xxx numbers the monsters have an opportunity to react to their environment. Each of these sections was developed, tested and integrated separately from the others.

**Input commands**

A list of verbs and nouns was developed and categorized as to nature or function. After entering these tables into the program, I worked on the routine to read and decode input commands. Each word was picked out of the input string, then searched for in the noun and verb lists. The first recognized verb and noun numbers were the output of this routine, and this output controlled the action routines. I later added an edit to compare the noun type and verb type to see if they were compatible.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Type</th>
<th>Direction</th>
<th>Sensative</th>
<th>Noun Types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Location</td>
<td>Weapon</td>
<td>Monster</td>
</tr>
<tr>
<td>1</td>
<td>GO</td>
<td>1 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>JUMP</td>
<td>11 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>RUN</td>
<td>1 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>WALK</td>
<td>1 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>DRIVE</td>
<td>1 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>CLIMB</td>
<td>3 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>DIG</td>
<td>2 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>CARRY</td>
<td>116 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>DROP</td>
<td>116 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>PUT</td>
<td>116 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>TAKE</td>
<td>116 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>12</td>
<td>USE</td>
<td>36 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>13</td>
<td>WISH</td>
<td>36 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>14</td>
<td>THROW</td>
<td>4 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>15</td>
<td>HELP</td>
<td>8 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>16</td>
<td>KILL</td>
<td>8 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>17</td>
<td>STOP</td>
<td>40 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>18</td>
<td>HIT</td>
<td>8 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>19</td>
<td>FIGHT</td>
<td>8 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>20</td>
<td>RUB</td>
<td>16 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>21</td>
<td>START</td>
<td>32 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>22</td>
<td>DRINK</td>
<td>64 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>23</td>
<td>EAT</td>
<td>64 x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>24</td>
<td>BITE</td>
<td>64 x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Cave room structure

The map was finalized, giving each room a number. The interconnections were entered into the N, E, S and W arrays, with a positive number indicating an exit in that direction to the room number specified. A series of statements were inserted in order to print out the current room descriptions, but at the time only the room name was printed. Later I discovered that there was not enough memory to put in the complete descriptions in any event.

The first of the action routines — the MOVE routine — was coded next. If there was a possible move in the requested direction, the LOC variable was set to the new room and its description was printed. This portion was a lot of fun to test and debug.

Monsters, treasures and battles

The monsters and treasures were merely noun objects in the caves, like all of the other things. A relationship was defined between the monster, his treasure, the player, and the player’s use of weapons. Thus grew up the monster table and the weapons table. The monster table identifies the monster, determines his strength, defines his treasure, identifies his home chamber, and determines how quickly he moves about the caves. The monsters move through the caverns to find their treasures if they are stolen. In the table are certain base probability factors for the monster to kill the player, steal all the player’s treasures, or steal only the treasure that originally belonged to the monster.

The weapons table details the power of each of the player’s weapons and determines which monsters they are effective against. The next action routine was ready to implement: the ATTACK routine. This is invoked whenever a weapon is used, put, thrown, and so on. Any monsters in the room are attacked, and their life forces are decreased by a random amount limited by the force of the weapon used. When a monster’s life force is reduced to zero, it is eliminated.

Of course, it is not fair to let the player cut the demons to shreds without allowing them to fight back. Thus came the REACTION routines. Happy monsters are those that have their own treasures in their room and have not been attacked. Any monsters that are not happy will seek someone to vent their anger upon, and that person is the player. A very intricate set of probabilities decides the outcome of this anger. The more the monster has been hurt by the player’s attacks, the weaker his counterattack will become. But also, the more times he has countered in vain, the madder he gets! Nothing is more deadly than a mad monster.

<table>
<thead>
<tr>
<th>Noun Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>NE</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>SE</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>SW</td>
</tr>
<tr>
<td>W</td>
</tr>
<tr>
<td>NW</td>
</tr>
<tr>
<td>DOWN</td>
</tr>
<tr>
<td>CAVE</td>
</tr>
<tr>
<td>LAKE</td>
</tr>
<tr>
<td>RIVER</td>
</tr>
<tr>
<td>TREE</td>
</tr>
<tr>
<td>AX</td>
</tr>
<tr>
<td>BOMB</td>
</tr>
<tr>
<td>CURSE</td>
</tr>
<tr>
<td>FIRE</td>
</tr>
<tr>
<td>KNIFE</td>
</tr>
<tr>
<td>CLAM</td>
</tr>
<tr>
<td>BATS</td>
</tr>
<tr>
<td>BONES</td>
</tr>
<tr>
<td>GHOST</td>
</tr>
<tr>
<td>OGRE</td>
</tr>
<tr>
<td>CHEST</td>
</tr>
<tr>
<td>GOLD</td>
</tr>
<tr>
<td>PEARL</td>
</tr>
<tr>
<td>LAMP</td>
</tr>
<tr>
<td>RAFT</td>
</tr>
<tr>
<td>ROPE</td>
</tr>
<tr>
<td>TENT</td>
</tr>
<tr>
<td>TRUCK</td>
</tr>
<tr>
<td>LIGHT</td>
</tr>
<tr>
<td>WATER</td>
</tr>
<tr>
<td>APPLE</td>
</tr>
<tr>
<td>ICE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Room Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Room</td>
</tr>
<tr>
<td>1 Mouth</td>
</tr>
<tr>
<td>2 Tree</td>
</tr>
<tr>
<td>3 Writing</td>
</tr>
<tr>
<td>4 Pit</td>
</tr>
<tr>
<td>5 South Lake Shore</td>
</tr>
<tr>
<td>6 West Lake Shore</td>
</tr>
<tr>
<td>7 North Lake Shore</td>
</tr>
<tr>
<td>8 Maze</td>
</tr>
<tr>
<td>9 Frozen River</td>
</tr>
<tr>
<td>10 Swift River Room</td>
</tr>
<tr>
<td>11 Hub</td>
</tr>
<tr>
<td>12 Ice</td>
</tr>
<tr>
<td>13 Chimney</td>
</tr>
<tr>
<td>14 Gold</td>
</tr>
<tr>
<td>15 Bones</td>
</tr>
<tr>
<td>16 Bats</td>
</tr>
<tr>
<td>17 Ghost</td>
</tr>
<tr>
<td>18 Moosly Lake</td>
</tr>
<tr>
<td>19 Swift River</td>
</tr>
<tr>
<td>20 Intersect 1</td>
</tr>
<tr>
<td>21 Intersect 2</td>
</tr>
<tr>
<td>22 Intersect 3</td>
</tr>
<tr>
<td>49 Falls (Over)</td>
</tr>
<tr>
<td>50 Home</td>
</tr>
</tbody>
</table>
Lots of testing and refinements later, SPELUNKER took its maiden voyage. Surely a program like this is never finished. The framework has been laid for all sorts of adventures, whatever one can imagine. And, now that I have more memory, I can expand the scope and capabilities of the program.

Program Flow

Initialize - 30000's
Dimension and set up data for nouns, verbs, noun types, verb types, status or location of nouns, north, south, east and west tunnel connections, monster life force tables, and weapons table.

Output - 4000's
Print room descriptions, possible exit directions, and room contents as well as your possessions.

Input/Decode - 1000's
Read your typed-in command, select each word and scan it against the noun and verb lists. The first valid noun and verb are edited and used to control the rest of the program.

Actions - 2000's
This routine handles takes and puts, special verbs and nouns, your attacks on life forces, and movement through the caverns.

Reactions - 3000's
The demonic life forces who have been attacked or who do not have their own treasures fight back. Based on complex probabilities, they either kill, steal your treasures, or wander the caverns in search of you.

Microbes

Move It: Relocating PET Source Programs and Object Code
Herman, 16:17

The following table should have appeared with Harvey H. Herman's article in MICRO 16:17 MOVE IT...

AIM-65 in the Ham Shack
De Jong, 16:29

The following table should have appeared with Marvin L. De Jong's article. It is a table of ASCII to Morse code lookups which is used by the "Ham Shack" program.

Micro

Figure 1: Monitor Relocation Procedure
LIST
0 REM SPELUNKER I
1 GOTO 30000: REM TO INITIALIZE
1000 PRINT "?: INPUT IN$: IN$(LEN(IN$)+1)="" GO N * " ;I=1
1005 NOUN=0: VERB=0
1010 GOSUB 1500: GOSUB 1600: GOSUB 1700
1020 IF W3$"*" THEN 1010
1050 NTYP=NTYP(NOUN): VTYP=VTYP(VERB)
1060 IF (VTYP MOD (NTYP+2))=NTYP THEN 2000
1070 PRINT "ICH VERSTEHE NICHT"
1080 GOTO 3000
1200 GOTO 2000
1500 W3$=":S=0: FOR I=1 TO LEN(IN$): IF S=0 THEN 1520: IF IN$<I>=" ": S=S+1: W3$(S)=IN$(I)
1520 NEXT I
1530 RETURN
1600 IF NOUN>0 THEN RETURN : FOR J=1 TO NUMN: IF W3$#NOUN$(J+5-4, J+5) THEN NEXT J: IF J=NUMN THEN RETURN : NOUN=J: W2$=W3$
1610 RETURN
1700 IF VERB>0 THEN RETURN : FOR J=1 TO NUMV: IF W3$#VERB$(J+5-4, J+5) THEN NEXT J: IF J=NUMV THEN RETURN : VERB=J: W1$=W3$
1710 RETURN
2000 REM MOVE
2010 NLOC=0
2020 IF NOUN=8 THEN 2200
2030 IF NOUN MOD 2)=1 THEN 2100
2040 IF LOC#11 AND LOC#8 THEN 1070
2100 GOTO 2100+NOUN+10
2110 NLOC=N(LOC): GOTO 2190
2120 NLOC=0: GOTO 2190
2130 NLOC=E(LOC): GOTO 2190
2140 NLOC=15: IF LOC=8 THEN NLOC=8: GOTO 2190
2150 NLOC=S(LOC): GOTO 2190
2160 NLOC=22: IF LOC=8 THEN NLOC=8: GOTO 2190
2170 NLOC=W(LOC): GOTO 2190
2180 NLOC=12: IF LOC=8 THEN NLOC=8: GOTO 2190
2190 IF RAFT=1 THEN NLOC=ABS(NLOC)
2191 RAFT=0: PLOC=LOC
2192 IF NLOC>0 THEN LOC=NLOC
2193 IF NLOC#12 THEN 2900
2194 IF M<55)<5 THEN 2900
2195 IF PLOC=6 THEN S<12)=0
2196 IF PLOC=11 THEN W<12)=0
2197 GOTO 2900
2200 IF NOUN#9 OR NOUN=10) AND ROPE=0 THEN GOTO 1070
2205 IF NOUN=9 THEN 2250
2210 IF LOC#5 AND LOC#13 THEN 1070
2220 IF LOC=5 THEN LOC=4
2230 IF LOC=13 THEN LOC=50
2240 GOTO 3000
2250 IF NOUN=10 THEN 2300
2260 IF LOC#4 THEN 1070
2270 LOC=5: GOTO 3000
2280 IF VERB=8 OR VERB=11 THEN 2320: GOTO 2350
2290 IF NUMP=8 THEN 1070
2305 IF NOUN=34 AND (LOC=19 OR LOC=10 OR LOC=5 OR LOC=18 OR LOC=7 OR LOC=6 OR LOC=11) THEN 2345
2330 IF STA(NOUN)#LOC THEN 1070
2335 IF NOUN=28 AND M(50)>0 THEN 1070
2340 STA(NOUN)=-1: GOTO 3000
2350 IF VERB=9 OR VERB=10 OR VERB=14 THEN 2370: GOTO 2400
2370 IF STA(NOUN)#-1 THEN 1070
2380 STA(NOUN)=LOC
2383 IF NOUN#33 THEN 2420
2385 IF VERB#10 THEN STA(33)=0
2387 LIGHT=0
2390 GOTO 2420
2400 IF VERB#12 THEN 2900
2410 IF STA(NOUN)#-1 THEN 1070
2420 FOR WT=1 TO NUMW+5-4 STEP 5
2425 IF NOUN#WT<WT( ) THEN 2480
2430 FOR D=2 TO 4
2435 IF STA(WT+D)#LOC THEN 2470
2440 FOR M=1 TO NUMM*10-9 STEP 10
2445 IF WT(WT+D)#M(M) THEN 2460
2446 HT=RND(WT(WT+1))/CURSE+1
2448 M(M+9)=M(M+9)-HT
2449 IF M(M+4)=0 THEN M(M+4)=1
2450 PRINT "ASSAULT ON NOUN$:M<M)>.. M<M)*5-4); "; HT; " UNITS"
2452 PRINT "ITS LIFE FORCE IS NOW ":M(M+9); ";"
2455 IF M(M+9)#0 THEN 2460
2456 PRINT NOUN$:M(M)*5-4,M(M)+5); " HAS BEEN ELIMINATED"
2457 STA(M(M))=0
2460 NEXT M
2470 NEXT D
2480 NEXT WT
2490 IF NOUN#16 OR VERB=10 THEN 2500
2492 STA(16)=0: GOTO 2493+ RND (4)
2493 N(LOC)=0: GOTO 2500
2494 E(LOC)=0: GOTO 2500
2495 S(LOC)=0: GOTO 2500
2496 W(LOC)=0
2500 IF NTVF'#32 THEN 2900
2510 IF NOUN#33 THEN 2520: IF VERB=12 THEN LIGHT=1: GOTO 2900
2520 IF NOUN#29 THEN 2530:RAFT=1: GOTO 2900
2530 IF NOUN#30 THEN 2540:ROPE=1: GOTO 2900
2540 REM
2560 IF NOUN#11 THEN ROPE=0
2570 IF STA(30)=LOC THEN ROPE=1
2580 IF LOC=12 THEN 3000
2590 W(12)=6:S(12)=11
3000 REM RE-ACTION
3010 FOR M=1 TO NUMM*10-9 STEP 10
3020 IF STA(M(M))#0 THEN 0020 SUB 3800
3030 NEXT M
3040 IF STA(35)=0 AND STA(34)=2 THEN STA(35)=2
3050 GOTO 4000
3060 REM MONS SUB
3062 MRM=STA(M(M)) MOD 100
3070 IF (STA(M(M+1)) MOD 100)=MRM AND M(M+4)=0 THEN 3900
3080 IF MRM=LOC THEN 3860
3090 M(M+2)=M(M+2)+M(M+3) MOD 6
3100 IF M(M+2)#0 THEN RETURN
3145 GOTO 3850+ RND (4)
3150 NLOC=N(MRM): GOTO 3855
3151 NLOC=E(MRM): GOTO 3855
3152 NLOC=S(MRM): GOTO 3855
3853 NLOC=M(MRM): GOTO 3855
3855 IF NLOC<1 THEN RETURN
3858 STR=M(M): NLOC=STR(M(M))=M(MRM): RETURN
3860 M(M+4)=M(M+4)+1
3865 KP=M(M-5)-(STR(M(M)+1)=-1)*40+9*(M(M+4)-2)*M(M+9)/100+CURSE
3866 IF KP>60 THEN KP=60
3870 SAP=M(M+6)+9*(M(M+4)-2)*M(M+9)/100+CURSE
3871 IF SAP>70 THEN SAP=70
3875 SRF=M(M+7)+9*(M(M+4)-2)*M(M+9)/100+CURSE
3876 IF SRF>80 THEN SRF=80
3877 PRINT "ATTACK BY ": NOUNS$=((M(M)-1)+5+1,M(M)+5)
3879 R1=RND(100): R2=RND(100): R3=RND(100)
3880 IF KP>R1 THEN 3920
3885 IF SAP>R2 THEN 3940
3887 IF STA(M(M+1))#-1 THEN RETURN
3890 IF SRF>R3 THEN 3960
3895 RETURN
3900 STA=M(M)
3905 STA=M(M+8)
3910 RETURN
3920 VTAB 23: TAB 1: PRINT "THE ": NOUNS$=((M(M)-1)+5+1,M(M)+5)" KILLED YOU!"
3924 PRINT KP, R1
3925 END
3940 FOR I=1 TO NUMN
3945 IF NTYP(I)=16 AND STR(I)=-1 THEN STR=STR(M+8)
3950 NEXT I
3957 PRINT "ALL YOUR REWARDS STOLEN"
3958 PRINT SAP; R2
3959 GOTO 3900
3960 PRINT "HE TOOK BACK HIS VALUABLE"
3962 PRINT SAP; R3
3965 GOTO 3900
4000 REM OUTPUT
4020 FOR I=3 TO 9: VTAB 1: TAB 2: PRINT ".": NEXT I
4050 IF LOC<1 OR LOC>50 THEN GOTO 4051
4060 GOTO 4000+100+LOC
4090 VTAB 23: TAB 1
4095 IF LIGHT=1 OR LOC<3 OR LOC>19 THEN 9100
4097 PRINT "IT IS VERY DARK"
4099 GOTO 9100
4100 LOC$="MOUTH ": GOSUB 4070
4109 GOTO 4090
4200 LOC$="TREE ROOM": GOSUB 4070
4209 GOTO 4090
4200 LOC$="WRITING ROOM": GOSUB 4070
4209 GOTO 4090
4300 LOC$="PIT": GOSUB 4070
4400 LOC$="SOUTH LAKE": GOSUB 4070
4500 LOC$="WEST LAKE": GOSUB 4070
4600 LOC$="MAZE ROOM": GOSUB 4070
4700 LOC$="NORTH LAKE": GOSUB 4070
4800 LOC$="SOUTH LAKE": GOSUB 4070
LOC$="FROZEN RIVER": GOSUB 4070
GOTO 4030
LOC$="RIVER ROOM": GOSUB 4070
GOTO 4090
LOC$="HUB ROOM": GOSUB 4070
GOTO 4090
LOC$="ICE ROOM": GOSUB 4070
GOTO 4090
LOC$="CHIMNEY": GOSUB 4070
GOTO 4090
LOC$="GOLD ROOM": GOSUB 4070
GOTO 4030
LOC$="BONES": GOSUB 4070
IF STH(35)=-1 THEN CURSE=CURSE+15
GOTO 4030
LOC$="BATS": GOSUB 4070
GOTO 4090
LOC$="GHOST ROOM": GOSUB 4070
GOTO 4030
LOC$="MISTY LAKE": GOSUB 4070
GOTO 4090
LOC$="SWIFT RIVER": GOSUB 4070
GOTO 4030
LOC$="INTERSECTION": GOSUB 4070
GOTO 4030
GOTO 4030
LOC$="OVER FALLS": GOSUB 4070
VTAB 23: TAB 1: GOTO 3030
LOC$="YOUR HOME": GOSUB 4070
AMT=0
IF STRT(25)=1 THEN AMT=AMT+13
IF STRT(26)=1 THEN AMT=AMT+22
IF STRT(27)=1 THEN AMT=AMT+8
IF STRT(28)=1 THEN AMT=AMT+5
VTRB 23: TRB 1
IF AMT=0 THEN 9090
PRINT "YOU HAVE FOUND $":AMT":": RND (900)+100:" IN TREASURES"
IF AMT>13 THEN PRINT "NICE SPELUNKING!"
PRINT "GOOD-BYE"
END
FOR I=2 TO 10: VTAB I: TAB 30: PRINT "": NEXT I
IF LIGHT=0 AND LOC>2 AND LOC#19 THEN 9290
VTAB 5: TAB 33: PRINT "": TAB 33: PRINT "+": POKE 50,63
IF N(LOC)=0 OR E(LOC)<0 AND RND RAFT=0 THEN 9150: VTAB 3: TAB 33: PRINT 
N": TAB 33: PRINT "+"
IF S(LOC)=0 OR W(LOC)<0 AND RND RAFT=0 THEN 9160: VTAB 8: TAB 33: PRINT 
S": TAB 33: PRINT "W"
IF E(LOC)=0 OR W(LOC)<0 AND RND RAFT=0 THEN 9170: VTAB 6: TAB 35: PRINT 
E": TAB 35: PRINT "W"
IF W(LOC)=0 OR W(LOC)<0 AND RND RAFT=0 THEN 9180: VTAB 6: TAB 30: PRINT 
"W"
IF (LOC=5 OR LOC=13) AND ROPE=1 THEN 9185: GOTO 9190
VTAB 2: TAB 33: PRINT "UP"
IF LOC#4 OR ROPE=0 THEN 9200
VTAB 8: TAB 33: PRINT "DOWN"
GOTO 92000
GOTO 9210: GOTO 9290
VTAB 3: TAB 30: PRINT "N": TAB 30: PRINT "W"
GOTO 9215: GOTO 9220: VTAB 3: TAB 35: PRINT "E": TAB 35: PRINT "N"
9220 VTRB 8: TAB 30: PRINT "W": TAB 35: PRINT "S": TAB 30: PRINT "W": TAB 35: PRINT "E"
9230 POKE 50, 255
9235 IF LIGHT=0 AND LOC>2 AND LOC<19 THEN 9400
9240 TAB 5: TAB 2: J=0
9250 FOR 1=1 TO NUMN-1
9255 PRINT NOUNS*<<1-1>*5+1, 1*5); 111 " ;
9260 J<5+1: IF J=4 THEN TAB 14
9270 NEXT 1
9280 VTTRB 13: TAB 2: FOR 1=1 TO 12: PRINT " " ;
9290 NEXT I
9300 VTRB 23: TAB 1: GOTO 1000
30000 REM INITIALIZE ROUTINE
30010 DIM IN$(40), NOUNS$(255), VERBS$(255), W1$(5), W2$(5), W3$(5), NTYP$(50), VTVP(50), STAV(50)
30020 DIM N(50), E(50), S(50), W(50)
30030 TEXT : CALL -936
30040 DIM LOC$(26), SPC$(5), POSS$(10), M(6+10)
30050 SPC$=" "
30060 NUMW=6
30070 DIM WT(5+NUMW)
30080 LOC=1
30090 REM INITIALIZE VARIABLES
30100 REM SHOULD BE READ AND DATA STMS
30110 NOUNS$(LEN(NOUNS$)+1)="N NE E SE S SW W NW UP DOWN"
30120 NOUNS$(LEN(NOUNS$)+1)="CAVE LAKE RIVERTREE"
30130 NOUNS$(LEN(NOUNS$)+1)="AX BOMB CURSEFIRE KNIFE"
30140 NOUNS$(LEN(NOUNS$)+1)="CLAM BATS BONESHOSTOGRE"
30150 NOUNS$(LEN(NOUNS$)+1)="CHESTGOLD PEARLAMP"
30160 NOUNS$(LEN(NOUNS$)+1)="RAFT ROPE TENT TRUCKLIGHT"
30170 NOUNS$(LEN(NOUNS$)+1)="WATERAPPLEICE"
30180 NOUNS$(LEN(NOUNS$)+1)="*****"
30190 NUMN=37
30200 VERBS$(LEN(VERBS$)+1)="GO JUMP RUN WALK DRIVECLIME"
30210 VERBS$(LEN(VERBS$)+1)="DIG"
30220 VERBS$(LEN(VERBS$)+1)="CARRY DROP PUT TAKE USE WISH THROW"
30230 VERBS$(LEN(VERBS$)+1)="HELP KILL STOP HIT FIGHT"
30240 VERBS$(LEN(VERBS$)+1)="PUB"
30250 VERBS$(LEN(VERBS$)+1)="START DRIVE"
30260 VERBS$(LEN(VERBS$)+1)="DRINK DRIVE"
30270 VERBS$(LEN(VERBS$)+1)="BITE"
30280 VERBS$(LEN(VERBS$)+1)="*****"
30290 NUMV=26
30300 FOR 1=1 TO 10: NTYP(I)=1: NEXT 1
30310 FOR 1=11 TO 14: NTYP(I)=2: NEXT 1
30320 FOR 1=15 TO 19: NTYP(I)=4: NEXT 1
30330 FOR 1=20 TO 24: NTYP(I)=8: NEXT 1
30340 FOR 1=25 TO 28: NTYP(I)=16: NEXT 1
30350 FOR 1=29 TO 32: NTYP(I)=32: NEXT 1
30360 FOR 1=33 TO 35: NTYP(I)=64: NEXT 1
30370 NTYP(36)=32
30380 NTYP(37)=6
30390 FOR 1=1 TO 6: VTYP(I)=1: NEXT 1
30400 VTYP(2)=11: VTYP(6)=3
30410 OCTOBER, 1979 MICRO - THE 6502 JOURNAL 17:23
30420 VTYP(7)=2
30430 FOR I=8 TO 11: VTYP(I)=16: NEXT I
30432 VTYP(12)=36: VTYP(13)=36: VTYP(14)=4
30440 FOR I=15 TO 19: VTYP(I)=8: NEXT I
30442 VTYP(17)=40
30450 VTYP(20)=16
30460 FOR I=21 TO 22: VTYP(I)=32: NEXT I
30470 FOR I=23 TO 25: VTYP(I)=64: NEXT I
30500 FOR I=1 TO 14: STR(I)=0: NEXT I
30510 STR(I)+4=16: STR(17)=15
30520 STR(I)=13: STR(I)=1: STR(16)=18
30530 STR(I)+16=15: STR(23)=17
30540 STR(I)=14: STR(I)=16: STR(26)=14
30550 STR(I)+18=12: STR(29)=5
30560 STR(I)=9: STR(I)+1=1: STR(I)=1
30570 STR(I)+1=1: STR(I)=0: STR(I)=0
30580 STR(I)=12
30600 FOR I=1 TO 50: N(I)=0: S(I)=0: E(I)=0: W(I)=0: NEXT I
30610 N(I)+50=0: N(I)=0: N(I)+50=8: N(I)+10=9
30620 N(I)+10=13: N(I)+22=22: N(I)+18=7
30630 N(I)+19=1: N(I)=8: N(I)+11=11
30650 E(I)+21=11
30670 S(I)+21=22: S(I)+22=16
30680 W(I)=1: W(I)=20: W(I)+12=8: W(I)+11=21
30690 W(I)=6: W(I)+11=6: W(I)+18=6: W(I)+21=17: W(I)+22=21
30700 POKE 50, 63
30720 VTAB 2: TAB 1
30730 FOR I=2 TO 23: PRINT "": TAB 29: IF I=11 THEN PRINT "": TAB 39: PRINT "": NEXT I
30800 FOR I=1 TO 60: M(I)=0: NEXT I
30810 M(I)+1=24: M(I)+4=26: M(I)+6=68: M(I)+7=9: M(I)+8=8: M(I)+9=7
30820 M(I)+10=13: M(I)+11=16: M(I)+12=22: M(I)+13=21: M(I)+14=22: M(I)+15=16
30830 M(I)=19: M(I)+24=6: M(I)+29=17: M(I)+30=50
30840 M(I)+31=26: M(I)+32=27: M(I)+33=1: M(I)+34=60: M(I)+35=65: M(I)+36=18: M(I)+40=68
30850 M(I)+41=36: M(I)+42=28: M(I)+43=1: M(I)+47=60: M(I)+49=12: M(I)+50=25
30860 M(I)+51=22: M(I)+53=1: M(I)+59=15: M(I)+60=75
30890 NUMM=6
30900 WT(I)+15=15: WT(I)+2=100: WT(I)+3=24: WT(I)+4=0: WT(I)+5=0
30910 WT(I)+6=16: WT(I)+7=150: WT(I)+8=24: WT(I)+9=22: WT(I)+10=36
30920 WT(I)+11=18: WT(I)+12=30: WT(I)+13=21: WT(I)+14=22: WT(I)+15=36
30930 WT(I)+16=19: WT(I)+17=50: WT(I)+18=24: WT(I)+19=20: WT(I)+20=0
30940 WT(I)+21=33: WT(I)+22=30: WT(I)+23=23: WT(I)+24=0: WT(I)+25=0
30950 WT(I)+26=36: WT(I)+27=40: WT(I)+28=21: WT(I)+29=0: WT(I)+30=0
30999 GOTO 4000
31999 TAB 1: PRINT "": RETURN
32000 PRINT PEEK (202)+ PEEK (203)*256)-( PEEK (204)+ PEEK (205)*256): END
The COMPUTERIST Has It All!!  
The leaders in Expansion Accessories for AIM / SYM / KIM  
Featuring a Power Supply/Enclosure for the AIM 65

AIM PLUS™
ENCLOSURE
WITH BUILT IN
POWER SUPPLY

SPECIFICATIONS:
INPUT: 110/220 VAC 50/60 Hz
OUTPUT: +5V @ 5A
+24V @ 1A
GROUNDED THREE-WIRE LINE CORD
ON/OFF SWITCH WITH PILOT LIGHT
Enclosure has room for the AIM and one additional board: MEMORY PLUS or VIDEO PLUS

AIM ENCLOSURE: $450
AIM PLUS: $110
AIM and AIM PLUS: $485

Plus some very elegant expansion boards

MEMORY PLUS™ FOR AIM/SYM/KIM
8K STATIC RAM
Sockets for 8K Eprom
6522 I/O Port
EPROM PROGRAMMER
MEMORY PLUS: $200
FULLY ASSEMBLED AND TESTED

VIDEO PLUS™ FOR AIM/SYM/KIM
UPPER/lower case ASCII
128 Additional User Programmable Characters: GRAPHICS, SYMBOLS, FOREIGN CHARACTERS
Programmable Screen Format up to 80 CHARACTERS, 24 LINES
KEYBOARD and LIGHT PEN Interfaces
Up to 4K DISPLAY RAM
Provision for 2K EPROM
Provision to add 6502 for STAND-ALONE SYSTEM
ASSEMBLED AND TESTED WITH 2K DISPLAY RAM
VIDEO PLUS: $245

Many other products available. Write for a complete catalog [or call if you are in a hurry!]

The prices listed above do not include shipping and handling.

PO Box 3
S. Chelmsford, MA 01864
817/358-3649

© © O M P y T F H I s M S T F
PO Box J
S. Chelmsford, MA 01864
817/358-3649
CHALLENGER IP OWNERS: Tired of 24x24 graphics? $40 worth of components will upgrade your video display to give 30 lines x 50 characters on screen and double your processor speed. Complete software and step by step hardware modifications for only $8.00. Contact: S. Chalfin
905 Clinton St
Philadelphia, PA 19107

PET LANGUAGE MACHINE GUIDE: comprehensive manual to aid the machine language programmer. More than 30 routines are fully detailed so that the reader can put them to use immediately. Specify old or new ROMs. $6.95 plus $.75 for postage. Visa or Mastercharge accepted. Money back guarantee (10 days). Contact: ABACUS SOFTWARE
P.O. Box 7211
Grand Rapids, MI 49510

WANTED Disk operating system, hardware and software for use with 16K SYM-1. Contact: Kenneth A. Edelstein
341 West 71st Street
New York, NY 10023

SYM-1 OWNERS-SYM/KIM Appendix to First Book of KIM details changes to KIM games run on BASIC SYM-1. Changes shown line by line; load, modify, and run. Appendix only $4. First Book of KIM $9., combo both $12.50 postpaid. Order from: Robert A. Peck
P.O. Box 2231
Sunnyvale, CA 94087

NEW BOWLING PROGRAM FOR APPLET: One to four players compete in a standard bowling game. Challenging game with excellent quality. APPLESOFT program. Cassette $9.95. Order from: C.E. Hoverson
125 Marcella Road
Hampton, VA 23666

"STAR TREK" for your OSI Challenger IP or Superboard II, needs 5K RAM. Runs in machine language. Very fast and tough to beat. Send check or money order for $12.95 to: David Van't Hof
RR1
Hospers, IA 51238

Advertising Software for APPLE: $25 buys SCROLLING WONDER, GIANT LETTER, and HI-RES ALPHANUMERICS: on cassette, 16kb. $25 buys MULTI-MESSAGE with INTERLEAVED KALEIDOSCOPE, and MULTI-MESSAGE with INTERLEAVED ABSTRACT ART: on cassette, 32kb. Send check or money order to: CONNECTICUT INFORMATION SYSTEMS
218 Huntington Rd.
Bridgeport, CT 06606

QUALITY SOFTWARE for the PET: Cheque-Check (best balancer) $7.95; MetricCalc (RPN, with conversions) $7.95; Men-Explorer (learn PET details) $7.95; Billboard ("Times Square" message of choice) $49.95. Send check or money order to: MICRO SOFTWARE SYSTEMS
Dept. M879, P.O. Box 1442
Woodbridge, VA 22193
VA res. add 4%
tax.

THE FLEXIBILITY YOU HAVE DREAMED ABOUT IS NOW AVAILABLE!
- 320 WIDE X 200 HIGH RESOLUTION
- EACH DOT INDIVIDUALLY ADDRESSABLE
- SOFTWARE SUPPORT — LEVEL 1 GIVES GRAPHICS & TEXT CONTROL AT MACHINE LANGUAGE SPEED BUT ACCESSIBLE FROM BASIC BY GOSUB AND VARIABLE STATEMENTS.
- DUAL PORT 8K BYTE MEMORY ON BOARD ALLOWS FULL USE OF MEMORY FOR OTHER TASKS (SEE YOUR PROGRAMS IN THEIR DIGITAL FORM IF YOU LIKE!)
- DOUBLES THE MEMORY SIZE OF AN 8K PET
- COMPLETELY TRANSPARENT SCREEN REFRESH - NO SNOW OR BLINKING EVER - THE PROPER WAY TO DJ IT!

REAL GRAPHICS FOR PET

MICRO TECHNOLOGY UNLIMITED
841 GALAXY WAY
PO BOX 4596
MANCHESTER, NH 03108
(603) 627-1464
While many 6502 computerists are becoming familiar with the 6522 Versatile Interface Adapter, do you really know how all of its features work or how to use them? This tutorial will clear up the mysteries of the 6522.

Applications that require interval timers include everything from the production of simple sound effects for games to the implementation of sophisticated data logging or control processes. Because single-chip microcomputers, such as the Rockwell 6500/1 and the Intel 8048, are intended for high volume, low cost applications, the fact that they include counter/timer logic is a testimony to the importance of counter/timer functions for a large variety of applications. Several simple applications will be explained.

The techniques will focus on the counter/timers found on the 6522 Versatile Interface Adapter. The 6522 is currently popular in a number of microcomputer systems that utilize the 6502, including the SYM-1, the AIM 65, and the MICRO PLUS. Expansion boards such as the MEMORY PLUS also include the 6522, and the 6522 can be easily interfaced to the popular KIM-1 (see 6502 User Notes, No. 13, pg. 16). However, the techniques that are described will frequently be applicable to any counter/timer with only minor modifications in the hardware or the programs.

The basic features included in many counter/timers (also called interval timers) are shown in Figure 1. This block diagram shows that a counter/timer consists of three registers: the counter register which is either an 8-bit register or a 16-bit register, a flag register, and a control register. A number, N, is loaded into the counter register by a WRITE (typically an STA) instruction. If the counter is a 16-bit register, then two write instructions are required. In 6502
systems these registers are simply some of the 65536 memory locations. After $N$ is loaded into the counter, it is decremented at a rate determined by the clock signal connected to the counter.

When $N$ decrements through zero, one of the bits in the flag register is set to logic one. Thus, the contents of the counter register change as follows: $N$, $N-1$, $N-2$, ..., 2, 1, 0, and on the next clock cycle the flag is set. Consequently it actually takes $N + 1$ clock cycles to “time out.” This summarizes the fundamentals of the counting/timing process.

The control register is used to select one of several modes available to the programmer. For example, in one mode the contents of the counter register are decremented at the rate as the system clock, while in another mode pulses on an external pin cause the counter to decrement, and in a third mode the counter is automatically reloaded after each time-out. The modes available with a 6522 will be discussed in more detail below.

The 6522 Versatile Interface Adapter

The 6522 Versatile Interface Adapter is a complex integrated circuit that includes two eight-bit I/O ports, four pins associated with handshaking signals for these two I/O ports, and two interval timers. The I/O ports and handshaking pins will only be of incidental interest, and we will describe the use of a few of these features as the need arises. Our principal interest is in the two counter/timers that are available on the 6522, called T1 and T2 respectively. Of course, the various registers needed to control timer operations and to select the various timing modes will also be of interest.

In most 6502 microcomputer systems, the 6522 will be interfaced to occupy 16 contiguous memory locations. The AIM 65 and SYM-1, for example, use locations with addresses $A000$ to $A00F$ for the 6522. Table I summarizes the names of each of these 16 locations, while Table II lists the functions of the registers. Of particular interest are the timer locations $A004$ through $A009$, the interrupt flag register (IFR), and the control register (ACR). These correspond precisely with the registers mentioned above in connection with Figure 1. That is, the IFR is the flag register and the ACR is the control register.

Both counter/timers, T1 and T2, on the 6522 are 16-bit devices; that is, a 16-bit number is loaded into the counter register and then decremented until time-out. Because the counter registers are 16-bit registers, two WRITE operations are needed to load the counter since only eight bits of data can be written at one time.

To prevent one eight-bit number (the low-order byte) from being decremented

while the other (the high-order byte) is still not loaded, temporary storage latches are provided. Using the T2 timer as an example, the low-order eight bits of the number $N$, to be loaded into the counter are loaded into the low-order byte of the T2 latch (T2LL). Nothing happens. Next, the high-order eight bits of $N$ are loaded into the high-order byte of the T2 counter. Referring to Table II, this last operation has three important and simultaneous consequences:

- The byte stored in the T2 latch (T2LL) is transferred to the low-order byte of the T2 counter (T2CL). T2 now contains a 16-bit number.
- The interrupt flag that signals the time-out, bit five of the IFR, is cleared (set to zero). It will be set (to one) when the number $N$ decrements through zero.
- The countdown begins.

The T1 timer has two latches, one to store the low-order byte to be transferred to the counter, and one to store the high-order byte to be transferred to the counter. One reason for this difference is that the T1 timer has a “free-running” mode. At the end of one time-out, the two bytes of data stored in the latches are automatically transferred to the 16-bit T1 counter to start a new timing interval.

Furthermore, the values in the two latches may be changed during one timing interval to give a new value for the next interval. The examples that follow should make these points clear. Additional discussion of the READ operations outlined in Table II will also be postponed until required by a specific example.

A Simple Delay Loop Using the T2 Timer

The most common application of counter/timers is the implementation of delay loops. The counter/timer replaces a series of instructions that are designed to waste time. The counter/timer simplifies greatly the instructions that are necessary to program a time delay, and furthermore, the computer may execute other tasks during the delay produced by the timer, a feat that is much more difficult to perform with a software implemented delay loop.

An assembly language version of a simple delay loop using the T2 timer on the 6522 is listed in Table III. The mnemonics are perfectly general for 6502 systems, but the addresses of the registers of the 6522 are the ones given in Table II for the AIM 65 and the SYM-1. Programmers using other systems need only change the addresses to correspond to the locations of the 6522 registers in the address space of their systems.
Table I: Memory Assignment Names for the 6522 VIA.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>SYMBOL</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A000</td>
<td>ORB</td>
<td>Port B Input/Output Registers</td>
</tr>
<tr>
<td>$A001</td>
<td>ORA</td>
<td>Port A Input/Output Registers (with handshaking)</td>
</tr>
<tr>
<td>$A002</td>
<td>DDRB</td>
<td>Port B Data Direction Register</td>
</tr>
<tr>
<td>$A003</td>
<td>DDRA</td>
<td>Port A Data Direction Register</td>
</tr>
<tr>
<td>$A004</td>
<td>T1LL</td>
<td>Timer 1 Latch Low-order Byte (WRITE)</td>
</tr>
<tr>
<td>$A004</td>
<td>T1CL</td>
<td>Timer 1 Counter Low-order Byte (READ)</td>
</tr>
<tr>
<td>$A005</td>
<td>T1LH</td>
<td>Timer 1 Latch High-order Byte (WRITE)</td>
</tr>
<tr>
<td>$A005</td>
<td>T1CH</td>
<td>Timer 1 Counter High-order Byte (READ)</td>
</tr>
<tr>
<td>$A006</td>
<td>T1LH</td>
<td>Timer 1 Latch Low-order Byte (READ or WRITE)</td>
</tr>
<tr>
<td>$A007</td>
<td>T2LL</td>
<td>Timer 2 Latch Low-order Byte (READ or WRITE)</td>
</tr>
<tr>
<td>$A008</td>
<td>T2CL</td>
<td>Timer 2 Counter Low-order Byte (WRITE)</td>
</tr>
<tr>
<td>$A009</td>
<td>T2CH</td>
<td>Timer 2 Counter High-order Byte (READ or WRITE)</td>
</tr>
<tr>
<td>$A00A</td>
<td>SR</td>
<td>Shift Register</td>
</tr>
<tr>
<td>$A00B</td>
<td>ACR</td>
<td>Auxiliary Control Register (Control Register for Timers)</td>
</tr>
<tr>
<td>$A00C</td>
<td>PCR</td>
<td>Peripheral Control Register</td>
</tr>
<tr>
<td>$A00D</td>
<td>IFR</td>
<td>Interrupt Flag Register (Status Register)</td>
</tr>
<tr>
<td>$A00E</td>
<td>IER</td>
<td>Interrupt Enable Register</td>
</tr>
<tr>
<td>$A00F</td>
<td>ORA</td>
<td>Port A I/O Register (without handshaking)</td>
</tr>
</tbody>
</table>

 systems. Pay careful attention to the comments in Table III, because they relate each step to points in our previous discussion. Figure 2 is a flowchart of the delay loop, and it has a box for each of the instructions in Table III.

In the program listing given in Table III, timing begins at the completion of the STA T2CH instruction. The program waits in the loop consisting of the series of instructions LDA IFR, AND $20, BEQ WAIT until the time-out of the T2 timer sets bit five of the interrupt flag register. The formula for the time T required for the interval timer to time-out is:

\[ T = (N + 1)T_C \]

where N is the 16-bit number loaded into the counter and T_C is the clock period (typically one microsecond).

If the branch instructions (LDA IFR, AND $20, BEQ WAIT) are taken into account, then the total loop time, \( T_L \), is given by the expression:

\[ (N + 6)T_C < T_L < (N + 14)T_C \]

The uncertainty of eight cycles in the loop time arises from the uncertainty of where the T2 counter/timer actually times out in the series of test and branch instructions within the loop. For the numbers that were used in Table III,

Table II: Memory Assignments and Functions of Some of the Registers of the 6522 VIA.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>SYMBOL</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A004</td>
<td>T1LL</td>
<td>WRITE (STA T1LL): Load an eight-bit number into the low-order byte of the T1 latch.</td>
</tr>
<tr>
<td>$A004</td>
<td>T1CL</td>
<td>READ (LDA T1CL): Read the contents of the low-order byte of the T1 counter, and clear the interrupt flag, bit six of the IFR.</td>
</tr>
<tr>
<td>$A005</td>
<td>T1LH</td>
<td>WRITE (STA T1LH): Load an eight-bit number into the high-order byte of the T1 latch, transfer the contents of both T1 latches to the T1 counters, clear the T1 interrupt flag, and start the counting process.</td>
</tr>
<tr>
<td>$A006</td>
<td>T1CH</td>
<td>READ (LDA T1CH): Read the contents of the high-order byte of the T1 counter.</td>
</tr>
<tr>
<td>$A006</td>
<td>T1LL</td>
<td>WRITE (STA T1LL): Load an eight-bit number into the low-order byte of the T1 latch.</td>
</tr>
<tr>
<td>$A007</td>
<td>T1LH</td>
<td>READ (LDA T1LH): Read the contents of the low-order byte of the T1 latch.</td>
</tr>
<tr>
<td>$A007</td>
<td>T1LH</td>
<td>WRITE (STA T1LH): Load an eight-bit number into the high-order byte of the T1 latch and clear the T1 interrupt flag.</td>
</tr>
<tr>
<td>$A007</td>
<td>T1LH</td>
<td>READ (LDA T1LH): Read the contents of the high-order byte of the T1 latch.</td>
</tr>
<tr>
<td>$A008</td>
<td>T2LL</td>
<td>WRITE (STA T2LL): Load an eight-bit number into the low-order byte of the T2 latch.</td>
</tr>
<tr>
<td>$A008</td>
<td>T2CL</td>
<td>READ (LDA T2CL): Read the contents of the low-order byte of the T2 counter, and clear the interrupt flag, bit five of the IFR.</td>
</tr>
<tr>
<td>$A009</td>
<td>T2CH</td>
<td>WRITE (STA T2CH): Load and eight-bit number into the high-order byte of the T2 counter, transfer the contents of the low-order byte in the T2 latch to the low-order byte of the T2 counter, clear the T2 interrupt flag, and start the counting process.</td>
</tr>
<tr>
<td>$A009</td>
<td>T2CH</td>
<td>READ (LDA T2CH): Read the contents of the high-order byte of the T2 counter.</td>
</tr>
<tr>
<td>$A00B</td>
<td>ACR</td>
<td>Bits five, six, and seven control the modes of T1 and T2.</td>
</tr>
<tr>
<td>$A00D</td>
<td>IFR</td>
<td>Bit six equal to one signals a time-out of the T1 counter/timer. Bit five equal to one signals a time-out of the T2 counter/timer.</td>
</tr>
</tbody>
</table>
Figure 3. 60 Hz Signal Conditioner for the Low Overhead Clock. A circuit based on the 555 timer and using only the +5V supply can be found in Berlin's 555 Timer Applications Sourcebook, pgs.2-13.

\[ T = (634E + 1)T_o = 0.05 \text{ seconds for a one microsecond clock.} \]

The loop time is between 5 and 13 microseconds longer. For many applications, this uncertainty will be of no consequence.

As pointed out earlier, the microprocessor need not be idle while the timer is timing out. For the particular delay of 0.05 seconds programmed in Table III, a total of 50,000 clock cycles elapse while the timer is running. During that time, between 25,000 and 10,000 instructions could be executed by the 6502. These instructions would be placed between the STA T2CH and the LDA IFR instructions. This is the principal advantage of the counter/timer implemented delay loop; that is, the microprocessor can be performing meaningful tasks during the timing-out process.

### Counting Pulses — A 24-Hour Clock

The T2 timer can also be used to count pulses from an external source. This is useful for frequency counting (MICRO, June 1979, pg. 41) or any other event counting application such as radioactive half-life measurements. The T2 timer is placed in its pulse counting mode by setting bit five in the auxiliary control register (ACR) to logic one, and applying the TTL level pulses to bit six of port B, PB6. To illustrate this mode, and to illustrate how the timers can be used to generate interrupt requests (IRQs), we have chosen to describe a simple 24-hour clock that requires very little computer time overhead.

The 60 Hz power line frequency is sufficiently stable over long periods for many clocks. Somewhere in your microcomputer system you will probably be able to locate a low-voltage 60 Hz source. This is conditioned by the circuit shown in Figure 3 to produce a 60 Hz square wave, and the output is applied to PB6 to be counted. Clearly there are 3600 (0E10) such pulses in a minute.

The T2 counter/timer will be programmed to count 3600 pulses followed by an interrupt request. The interrupt routine increments one location in memory to keep track of minutes, and when this location reaches 60, another location is incremented to keep track of the hours. At the beginning of the interrupt routine the T2 counter/timer is reloaded with 3600 for the next period.

The program is listed in Table IV. The first two instructions set bit five of the ACR to logic one. Next the timer is loaded with $0E0F. Note that $0E0F + 1 = 3600. The LDA $A0 and STA IER instructions enable interrupts from bit five of the interrupt flag register (IFR) of the 6522 to the 6502 microprocessor's IRQ pin, a connection that is usually internal to the microcomputer system.

To enable interrupt request signals from T2, bit five of the IER (interrupt enable register) must be set to logic one, with bit seven of the IER also set to logic one. At the end of the timing interval, not only will bit five of the IFR be set to one, but also the IRQ pin on the 6502 microprocessor will be pulled to logic zero, producing an interrupt request.

The next instruction after enabling the interrupt from the T2 timer is the CLI instruction that allows the 6502 to recognize these interrupts. The last instruction in the main program should not be taken literally. It is simply an infinite loop that represents the user's main program, a FORTRAN interpreter for example. The interrupt routine is also given in Table IV. Timekeeping routines have been described in several other articles (MICRO, March 1979, pg. 5), so the details will not be repeated here. Note that in order for the program to execute, the IRQ vector must point to the starting address of the interrupt request routine, in our case $0300. Note also, that the program could be easily modified to keep track of seconds by counting only 60 pulses, something that can be done with an eight-bit counter like the one on the R650/1. The hours-minutes clock requires only about 50 microseconds per
Table IV. Low Overhead 24-hour Clock.

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0200</td>
<td>A9 20</td>
<td>LDA $20 Put T2 in its pulse-counting mode</td>
</tr>
<tr>
<td>$0202</td>
<td>8D 0B A0</td>
<td>STA ACR by setting bit five to logic one.</td>
</tr>
<tr>
<td>$0205</td>
<td>A9 0F</td>
<td>LDA $0F Set T2 to count 3600 pulses.</td>
</tr>
<tr>
<td>$0207</td>
<td>8D 0A A0</td>
<td>LDA $20</td>
</tr>
<tr>
<td>$020A</td>
<td>A9 0E</td>
<td>LDA $0E Set up T2CH.</td>
</tr>
<tr>
<td>$020C</td>
<td>8D 09 A0</td>
<td>LDA $40 Get minutes.</td>
</tr>
<tr>
<td>$020F</td>
<td>A9 A0</td>
<td>LDA $A0 Get minutes.</td>
</tr>
<tr>
<td>$0211</td>
<td>8D 0B A0</td>
<td>STA IER</td>
</tr>
<tr>
<td>$0214</td>
<td>58</td>
<td>CLI Set up interrupt enable register to permit IRQ from T2.</td>
</tr>
<tr>
<td>$0215 AC15 02</td>
<td>HERE</td>
<td>JMP HERE</td>
</tr>
</tbody>
</table>

INTERRUPT ROUTINE

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0200 A9 0B</td>
<td>LDA $0B</td>
<td>Start counting pulses again by loading T2CH.</td>
</tr>
<tr>
<td>$0202 8D 0B A0</td>
<td>STA T2CH</td>
<td>Clear carry for addition.</td>
</tr>
<tr>
<td>$0205 A9 0F</td>
<td>CLC</td>
<td>Set decimal mode for addition.</td>
</tr>
<tr>
<td>$0207 8D 0A A0</td>
<td>LDA MIN</td>
<td>Get minutes.</td>
</tr>
<tr>
<td>$020F 8D 00</td>
<td>ADC $01</td>
<td>Add one.</td>
</tr>
<tr>
<td>$0200 C9 60</td>
<td>CMP $60</td>
<td>Is one hour complete?</td>
</tr>
<tr>
<td>$0211 A9 00</td>
<td>BNE DONE</td>
<td>No, get out of interrupt routine.</td>
</tr>
<tr>
<td>$0213 85 00</td>
<td>LDA $00</td>
<td>Yes, set minutes to zero.</td>
</tr>
<tr>
<td>$0215 18</td>
<td>STA MIN</td>
<td></td>
</tr>
<tr>
<td>$0217 65 18</td>
<td>LDA HRS</td>
<td>Get hours.</td>
</tr>
<tr>
<td>$0217 65 01</td>
<td>ADC $01</td>
<td>Add one.</td>
</tr>
<tr>
<td>$0224 0B 01</td>
<td>STA HRS</td>
<td></td>
</tr>
<tr>
<td>$0227 8D A9</td>
<td>CMP $2L</td>
<td>Is one day complete?</td>
</tr>
<tr>
<td>$0229 00</td>
<td>BNE DONE</td>
<td></td>
</tr>
<tr>
<td>$022B 85 01</td>
<td>LDA $00</td>
<td>Clear hours.</td>
</tr>
<tr>
<td>$022D 08</td>
<td>STA HRS</td>
<td></td>
</tr>
<tr>
<td>$0230 68</td>
<td>DONE</td>
<td>Clear decimal mode.</td>
</tr>
<tr>
<td>$0232 04 D0</td>
<td>DRE</td>
<td>Return to the main program.</td>
</tr>
</tbody>
</table>

Producing Long Time Delays

The maximum time delay that can be produced with the T2 counter/timer when it is decrementing at the system clock rate is approximately (2FFFF + 1)TC or 0.065536 seconds if TC = 1 microsecond. In certain applications longer time delays are necessary. To obtain these delays, the T1 timer is used in conjunction with the T2 counter/timer. We digress for a moment to introduce the T1 timer.

The T1 timer can be used to implement a simple delay loop in exactly the same way as the T2 timer. Refer to Table III. If the addresses $A004 and $A005 replace addresses $A008 and $A009, respectively, and if bit six of the interrupt flag register (IFR) is tested rather than bit five, then the program in Table III will work in exactly the same way except that the T1 timer is being used.

The same equation gives the loop time and, as in the case of the T2 timer, the maximum delay is about 0.065 seconds. The T1 timer cannot, however, count pulses. Consequently it cannot replace the T2 timer in the program listed in Table IV. In place of the pulse counting modes, the T1 timer has a free-running mode, and it is capable of toggling the logic level on pin seven of Port B, PB7.

The initialization of the free-running mode with PB7 toggling is illustrated in a simple program shown in Table V. This program will produce a square wave output on PB7. The period of the square wave is given by the equation:

$$T_p = 2(N + 2)T_c$$

where $T_p$ is the period of the square wave, N is the 16-bit number loaded into the T1 timer, and $T_c$ is the period of the system clock (Typically one microsecond). The frequency of the square wave is $f = 1/T_p$.

To initialize this mode, bits seven and six of the auxiliary control register (ACR) must be set. Thus, the program in Table V begins by loading $30H$ into the ACR. Timing is initiated by loading the high-order byte of N into location $A005$ which corresponds to T1LH. Once started, the square wave will run forever, no matter what else is happening in the program, provided the registers that control the behavior of the T1 timer are not changed. That is, after the timer “times out”, it will automatically reload the two counter registers from the numbers stored in its latches, T1LL and T1LH.

The last instruction in Table V is an infinite loop that simulates the user's program intended to run concurrently with generation of the square wave. Table VI lists some values for N that are frequently used in timing applications. If you have an oscilloscope, run the program with various values of N and connect the
ATARI® PERSONAL
CCI COMPUTER SYSTEMS DELIVERS . . . NOW!!!

ATARI® 400™
PERSONAL COMPUTER SYSTEM
PRICE INCLUDES:
Computer Console
Basic Language Cartridge
Basic Language Programming Manual (Wiley)
400 Operator's Manual with Note Book
Power Supply
Tv Switch Box

$549.99

ATARI® 800™
PERSONAL COMPUTER SYSTEM
PRICE INCLUDES:
Computer Console
Basic Language Cartridge
Education System Master Cartridge
Basic Language Programming Manual (Wiley)
800 Operator's Manual with Note Book
ATARI 410 Program Recorder
Guide to Basic Programming Cassette
8K RAM Module+Power Supply+
Tv Switch Box

$999.99

WE PROMISE TO DELIVER!!!
• We GUARANTEE ship dates on prepaid Computer System orders. Send your orders in now!!!
• If for reasons beyond our control we miss a ship date, we will REFUND the shipping and handling charges and give you a 10% DISCOUNT on any future software purchases for your ATARI® System.*
• For prepaid system orders, get a FREE Accessory Controller of your choice.
*Order must be prepaid by Cashier's Check made out to Compu Components of Orange County.

ATARI® 810™
DISC DRIVE
ATARI® DISKETTES
CX8100 BLANK DISKETTES
CX8101 DISK FILE MANAGER
$5.00/ea.

ATARI® 820™
PRINTERS
$5.99/ea.

ATARI® 410™
PROGRAM RECORDER $89.99
ADD-ON MEMORY (800 ONLY)
CX852 8K RAM MEMORY MODULE $124.99
CX853 16K RAM MEMORY MODULE $249.99

PERIPHERALS AND ACCESSORIES

SOFTWARE

APPLICATIONS
BASIC GAME AND
PROGRAM CASSETTES

ATARI USER'S GROUP INFORMATION . CALL (714) 891-2584

WE PROMISE TO DELIVER!!!

COMPUTER COMPONENTS OF ORANGE COUNTY
6791 Westminster Ave., Westminster, CA 92683 714-891-2584 Telex 182274

PERIPHERALS AND ACCESSORIES

ROM CARTRIDGES
CX6001 EDUCATION SYSTEM MASTER CARTRIDGE
$34.99

KEY: (J) = uses joystick controller
(J) = uses paddle controller
(D) = uses driving controller

EDUCATION SYSTEM CASSETTE PROGRAMS
CX6001 U.S. HISTORY
CX6003 U.S. GOVERNMENT
CX6004 WORLD HISTORY (WESTERN)
CX6005 BASIC SOCIOLOGY
CX6006 COUNSELING PROCEDURES
CX6007 Principles of Accounting
CX6008 PHYSICS


CX6009 GREAT CLASSICS (ENGLISH)
CX6010 BUSINESS COMMUNICATIONS
CX6011 BASIC PSYCHOLOGY
CX6012 EFFECTIVE WRITING
CX6013 AUTO MECHANICS
CX6014 PRINCIPLES OF ECONOMICS
CX6015 SPELLING
CX6016 BASIC ELECTRICITY
CX6017 BASIC ALGEBRA


APPLICATIONS
CX4101 GUIDE TO BASIC PROGRAMMING
CX4102 ATARI BASIC
CX4103 ASSEMBLER DEBUG
CX4104 MUSIC COMPOSER
CX4105 COMPUTER CHESS


CX4106 GREAT CLASSICS (ENGLISH)
CX4107 BUSINESS COMMUNICATIONS
CX4108 BASIC PSYCHOLOGY
CX4109 EFFECTIVE WRITING
CX4110 AUTO MECHANICS
CX4111 PRINCIPLES OF ECONOMICS
CX4112 SPELLING
CX4113 BASIC ELECTRICITY
CX4114 BASIC ALGEBRA


BASIC GAME AND
PROGRAM CASSETTES

CX4101 GUIDE TO BASIC PROGRAMMING
CX4102 ATARI BASIC
CX4103 ASSEMBLER DEBUG
CX4104 MUSIC COMPOSER
CX4105 COMPUTER CHESS


(C) October Delivery  (N) November Delivery

—Prices subject to change.—
In Southern California We’re Number One!!

COMPUTER COMPONENTS (CCI) OF ORANGE COUNTY, your full support computer store presents the finest in personal computers

Send for our 36 page fully illustrated catalog $3.00. Save 50¢ by sending the top one inch of this ad with your order.

AN INDUSTRY STANDARD

Apple II or Apple II Plus with 16K $1195

SPECIAL
13" Color TV compatible with Apple II only 99¢ with system purchase

APPLE II will change the way you think about computers. That's because it is specifically designed to handle the day to day tasks of education, financial planning, building security, scientific calculation, and entertainment. APPLE II is appealing and comfortable (like other appliances that make your life easier); and it brings to personal computing a new level of simplicity.

A PROFESSIONAL BUSINESS SYSTEM

CBM™

ACCESSORIES FOR PET

SOFTWARE FOR PET

COMPUTER COMPONENTS OF ORANGE COUNTY

6791 Westminster Ave., Westminster, CA 92683 714-891-2584 Telex 182274

Hours: Tues-Fri 11:00 AM to 8:00 PM—Sat 10:00 AM to 6:00 PM—Sun 12:00 to 4:00 PM (Closed Mon)

Master Charge, Visa, B of A are accepted. Allow 2 weeks for personal check to clear. Add $2.00 for handling and postage. For computer systems please add $10.00 for shipping, handling and insurance. California residents add 6% Sales Tax.
input of the oscilloscope to PB7 to monitor the square wave. You can use the program to calibrate your oscilloscope sweep time. If you have a frequency counter, measure the frequency of the square wave at PB7 to verify the equation, using the values for N given in Table VI. N is the number to be loaded into T1.

Note that the frequency of the square wave produced at PB7 by the program listed in Table V is as precise as the crystal oscillator frequency used for the system clock. This is because the square wave frequency is independent of any instruction length. The principal advantage of the free-running mode of the T1 timer is that the time between interrupt flag settings (or the frequency of the square wave on PB7) is independent of any instruction length. Thus, one can construct very precise time-keeping routines (MICRO, March 1979, pg. 5) or time measuring routines.

To produce simple delay loops for long time intervals, the pulses from PB7 are fed to PB6. Timer T1 operates in its free-running mode, and timer T2 operates in its pulse counting mode. Consequently, T2 counts the pulses produced by T1 on PB7. A program to produce a delay of one hour is given in Table VII. This program may be easily modified to produce delays of 1, 10, 60, 100, 1000, 10000, 36000, or 65536 seconds.

Timer T1 produces a square wave whose period is 0.1 second. These pulses are counted by the T2 counter/timer. If nine is loaded into T2, then 10 pulses, each of 0.1 second duration, will be counted, giving a delay of one second. Other time intervals are programmed accordingly. Of course, there is an uncertainty of several microseconds in the actual loop time, but this uncertainty will be unimportant for most applications.

If the program in Table VII is modified to allow T2 to produce interrupt requests (IRQs) by loading $A0 into the interrupt enable register (IER) at location $A00E (refer to Table IV), then it could be used in connection with the interrupt routine given in Table IV to produce a 24-hour clock program. To generate an interrupt every minute, as required by the low-overhead clock, load T2 with $0030 instead of $C39F as shown in Table VII and your clock should run. These modifications are shown in the AIM 65 disassembly format.

### Sound Effects

The T1 timer can be used in its free-running mode to toggle PB7, and PB7 can be used to drive an amplifier. If the frequency is in the audible range, then a tone will be heard. A series of tones may make up a song. Table VIII lists the frequencies necessary to produce three oc-

---

**Table V. Program to Produce a Square Wave Output on PB7.**

<table>
<thead>
<tr>
<th>FREQUENCY</th>
<th>PERIOD</th>
<th>( N + 2 )</th>
<th>( N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Hz</td>
<td>0.10 sec</td>
<td>$\frac{50000}{160} = 312.5$</td>
<td>$312.5$</td>
</tr>
<tr>
<td>100 Hz</td>
<td>0.01 sec</td>
<td>$\frac{5000}{1$60} = 31.25$</td>
<td>$31.25$</td>
</tr>
<tr>
<td>1000 Hz</td>
<td>1.00 ms</td>
<td>$1000$</td>
<td>$1000$</td>
</tr>
<tr>
<td>10 kHz</td>
<td>0.10 ms</td>
<td>$100$</td>
<td>$100$</td>
</tr>
<tr>
<td>100 kHz</td>
<td>0.01 ms</td>
<td>$10$</td>
<td>$10$</td>
</tr>
<tr>
<td>250 kHz</td>
<td>4.00 us</td>
<td>$250$</td>
<td>$250$</td>
</tr>
</tbody>
</table>

---

**Table VI. Table for Producing Various Square Wave Frequencies.**

| \$O200 A9 CO | START | LDA $80 | Set bits seven and six of the ACR. |
| $O202 8D 0B A0 | STA $ACR$ | STA T1 | putting the T1 timer in its free-running mode with a square wave output on PB7. |
| $O205 A9 4D | LDA $4D | LDA T2 | Initialize T1 timer to run with a period of \(2(5$C34E + 2) = 100000\) microseconds. |
| $O207 8D 06 A0 | STA T1LL | LDA $8C9F + 1 = 36000$ | Start timer toggling PB7. |
| $O20A A9 C0 | LDA $9F | STA T2 | Set up T2 to count $8C9F + 1 = 36000$ counts. (36000)(0.1sec) = 1 hour. |
| $O20B 8D 08 A0 | STA T2LL | LDA $8C$ | Start counting. Clear IFR. |
| $O211 A9 8C | LDA $9F$ | STA T2CH | Check interrupt flag register to see if bit five has been set, indicating that T2 has counted 36000 pulses. |
| $O216 8D 09 A0 | STA T2CH | LDA $20 | Break to the monitor at the end of an hour. |
| $O219 A9 20 | LDA $8D$ | TEST IFR | Check to see if bit five has been set, indicating that T2 has counted 36000 pulses. |
| $O21B 2C 0D A0 | TEST IFR | LDA $20$ | Break to the monitor at the end of an hour. |
| $O21E F0 FB | BK TEST | LDA $8D$ | Break to the monitor at the end of an hour. |
| $O220 00 | LDA $8D$ | BK | Break to the monitor at the end of an hour. |

---

**Table VII. Program to Produce a One-Hour Delay.**

---

**Figure 6. Circuit to measure the time duration, \( T \), of a positive pulse.**

The CB1 pin must be programmed to produce an interrupt on the negative transition of the pulse by loading PCR4 with a zero. Change the byte at $0217 from $10 to $00 in the listing in Table X to accomplish this.
taves of notes on the equally tempered scale (note middle A corresponds to 440 Hz and successive note frequencies are related by a factor equal to the 12th root of two). Also listed in Table VIII are the half periods in microseconds; that is, the numbers that must be loaded into the T1 timer to produce the notes. Since the period of the square wave is \((N + 2)T_c\), each of the numbers in the last column of Table VIII should be decremented by two.

Table VIII. Note Table for Producing Tones on the Equally Tempered Scale.

<table>
<thead>
<tr>
<th>I.D. NUMBER</th>
<th>NOTE</th>
<th>FREQUENCY</th>
<th>PERIOD/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td></td>
<td>Hertz</td>
<td>Microseconds</td>
</tr>
<tr>
<td>$00</td>
<td>C_0</td>
<td>130.813</td>
<td>$0BEB</td>
</tr>
<tr>
<td>$01</td>
<td>C_0#</td>
<td>138.591</td>
<td>$0E18</td>
</tr>
<tr>
<td>$02</td>
<td>D_0</td>
<td>146.832</td>
<td>$0DLD</td>
</tr>
<tr>
<td>$03</td>
<td>D_0#</td>
<td>155.563</td>
<td>$0CBE</td>
</tr>
<tr>
<td>$04</td>
<td>E_0</td>
<td>164.814</td>
<td>$0BDA</td>
</tr>
<tr>
<td>$05</td>
<td>F_0</td>
<td>174.614</td>
<td>$0B2F</td>
</tr>
<tr>
<td>$06</td>
<td>F_0#</td>
<td>184.997</td>
<td>$0A8F</td>
</tr>
<tr>
<td>$07</td>
<td>G_0</td>
<td>195.998</td>
<td>$0F7</td>
</tr>
<tr>
<td>$08</td>
<td>G_0#</td>
<td>207.652</td>
<td>$0F6</td>
</tr>
<tr>
<td>$09</td>
<td>A_0</td>
<td>220.000</td>
<td>$0B8</td>
</tr>
<tr>
<td>$0A</td>
<td>A_0#</td>
<td>233.082</td>
<td>$061</td>
</tr>
<tr>
<td>$0B</td>
<td>B_0</td>
<td>246.945</td>
<td>$079</td>
</tr>
<tr>
<td>$0C</td>
<td>(middle) C_1</td>
<td>261.626</td>
<td>$077</td>
</tr>
<tr>
<td>$0D</td>
<td>C_1#</td>
<td>277.183</td>
<td>$07C</td>
</tr>
<tr>
<td>$0E</td>
<td>D_1</td>
<td>293.665</td>
<td>$06A</td>
</tr>
<tr>
<td>$0F</td>
<td>D_1#</td>
<td>311.127</td>
<td>$047</td>
</tr>
<tr>
<td>$10</td>
<td>E_1</td>
<td>329.628</td>
<td>$05E</td>
</tr>
<tr>
<td>$11</td>
<td>F_1</td>
<td>349.228</td>
<td>$039</td>
</tr>
<tr>
<td>$12</td>
<td>F_1#</td>
<td>369.995</td>
<td>$054</td>
</tr>
<tr>
<td>$13</td>
<td>G_1</td>
<td>391.995</td>
<td>$05C</td>
</tr>
<tr>
<td>$14</td>
<td>G_1#</td>
<td>415.304</td>
<td>$0B4</td>
</tr>
<tr>
<td>$15</td>
<td>A_1</td>
<td>440.000</td>
<td>$0L7</td>
</tr>
<tr>
<td>$16</td>
<td>A_1#</td>
<td>466.164</td>
<td>$0L3</td>
</tr>
<tr>
<td>$17</td>
<td>B_1</td>
<td>493.883</td>
<td>$03F</td>
</tr>
<tr>
<td>$18</td>
<td>C_2</td>
<td>523.251</td>
<td>$03B</td>
</tr>
<tr>
<td>$19</td>
<td>C_2#</td>
<td>554.365</td>
<td>$038</td>
</tr>
<tr>
<td>$1A</td>
<td>D_2</td>
<td>587.330</td>
<td>$035</td>
</tr>
<tr>
<td>$1B</td>
<td>D_2#</td>
<td>622.254</td>
<td>$032</td>
</tr>
<tr>
<td>$1C</td>
<td>E_2</td>
<td>659.255</td>
<td>$02F</td>
</tr>
<tr>
<td>$1D</td>
<td>F_2</td>
<td>698.456</td>
<td>$0CC</td>
</tr>
<tr>
<td>$1E</td>
<td>F_2#</td>
<td>739.989</td>
<td>$0A4</td>
</tr>
<tr>
<td>$1F</td>
<td>G_2</td>
<td>783.991</td>
<td>$02E</td>
</tr>
<tr>
<td>$20</td>
<td>G_2#</td>
<td>830.609</td>
<td>$026</td>
</tr>
<tr>
<td>$21</td>
<td>A_2</td>
<td>880.000</td>
<td>$023</td>
</tr>
<tr>
<td>$22</td>
<td>A_2#</td>
<td>932.328</td>
<td>$01B</td>
</tr>
<tr>
<td>$23</td>
<td>B_2</td>
<td>997.767</td>
<td>$01F</td>
</tr>
</tbody>
</table>

A program to play songs using the notes in Table VII is listed in Table IX. The identification numbers (I.D. numbers) of the notes in the song to be played are stored in a song table starting at $0400. Actually, the song could be stored anywhere in memory that is convenient, simply by changing the base address of the song table. The base address of the song table is stored in $0050 and $0051, called SONG and SONG + 1, respectively.

The song table given in Table IX simply plays the three octave scale from Table VIII with a variety of durations as indicated by the duration table. You are invited to make your own song or translate someone else's song into I.D. numbers. Better yet, write a song interpreter that does the translation for you.

Measuring the Time Between Events

A number of applications require that the time between two successive events be measured. The events might be the start and finish of a race, the arrival of cosmic rays, two heartbeats of an animal, and many others. If the events are periodic, then the time between events can be obtained by first measuring the frequency of the events with a...
Table IX. Program to Play a Song.

$0050 = SONG, [SONG] = $00
$0051 = SONG + 1, [SONG + 1] = $01
$0052 = DUR, [DUR] = $00
$0053 = DUR + 1, [DUR + 1] = $02
$0000 = NOTE (See Note Table)

NOTE TABLE
$0000 EC 16 8C D8 2D 8D F5
$0008 66 DF 5F E7 75 OA A5 45
$0010 EE 96 6E FA B2 6E 2F F2
$0018 EE 84 51 21 F4 CA A2 7C
$0020 98 16 F8 8E BE CD OF
$0028 OB 84 5A 9A 09 08 08 07
$0030 07 07 06 05 05 05 04
$0038 O4 O4 O4 O3 O3 O3 O3
$0040 O2 02 O2 O2 O2 O2 O1

DURATION TABLE
$0200 C1 C2 C3 C4 08 10 20 10 08
$0208 C4 C3 C2 C1 C0 08 10 20 08
$0210 10 08 06 02 01 00 00 00
$0218 20 40 00 00 00 00 00 00
$0220 C2 C1 C0 00

SONG TABLE (Plays scale)
$0400 00 01 02 03 04 05 06 07
$0408 08 09 0A 0B 0C 0D 0E 0F
$0410 10 11 12 13 14 15 16 17
$0418 18 19 1A 1B 1C 1D 1E 1F
$0420 20 21 22 23

$0000 A9 CO START LDA SOC
$0002 ED 0B AO STA ACR
$0005 AO 00 LDI SOC
$0007 E1 50 MORE LDA [SONG], T
$0009 AA TAX
$000A D5 00 LDA NOTE, X
$000B ED 06 AO STA TILL
$000D BA CX TAX
$0010 18 CLC
$0011 69 24 ADC $24
$0013 AA TAX
$0014 B5 00 LDA NOTE, X
$0016 ED 05 AO STA TILH
$0019 E1 52 LDA (DUR), Y
$001B F0 24 BEQ CUT
$001D AA TAX
$001E A9 FF AGN LDA $FF
$0020 ED 06 AO STA T2LL
$0022 A9 FF LDA $FF
$0025 ED 09 AO STA T2CH
$0028 A9 20 LDA $20
$002A 2C 0D AO BACK BIT IFR
$002C F0 FB BEQ BACK
$002F CA DEX
$0030 D0 EC BNE A3N
$0032 E6 50 INC SONG
$0034 D0 02 BNE PAST
$0036 E6 51 INC SONG + 1
$0038 E6 52 BNE PAST INC DFR
$003A DO 02 BNE T-BNE
$003C E6 53 INC DFR + 1
$003E 0C 07 02 THERE JMP MORE
$0041 A9 00 OUT LDA $20
$0043 ED 0B AO STA ACR
$0046 00 BRK

Figure 4. Circuit to measure the time interval, T, between two successive pulses.
Table X. Program to Measure the Time Between Two Pulses.

<table>
<thead>
<tr>
<th>Location</th>
<th>Opcode</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0200 A9 00</td>
<td>START</td>
<td>LDA $00 Clear display registers.</td>
</tr>
<tr>
<td>$0202 85 01</td>
<td>STA LEAST</td>
<td>Least-significant byte of time.</td>
</tr>
<tr>
<td>$0206 85 02</td>
<td>STA MIDST</td>
<td>Middle byte.</td>
</tr>
<tr>
<td>$0206 85 03</td>
<td>STA MOST</td>
<td>Most-significant byte of time.</td>
</tr>
<tr>
<td>$0208 A9 01</td>
<td>STA DDRB</td>
<td>Initialise PB0 to be an output pin.</td>
</tr>
<tr>
<td>$0210 8E 00 AO</td>
<td>STA PB6</td>
<td>Initialise PB6 to logic one, then toggle it to preset the 7474 flip-flop.</td>
</tr>
<tr>
<td>$0214 A9 01</td>
<td>LDA $01 Initialize PB0 to be an output pin.</td>
<td></td>
</tr>
<tr>
<td>$0216 85 11</td>
<td>STA .CNTHI</td>
<td>High-order byte stored in CNTHI.</td>
</tr>
<tr>
<td>$0218 A9 13</td>
<td>LDA $13</td>
<td>$1386 + 2 = 5000, so f = 100 Hz, Tp = 0.01s.</td>
</tr>
<tr>
<td>$0220 A9 8D</td>
<td>STA DDRB</td>
<td>Set period of square wave on PB7 so that Tp = 0.01 second.</td>
</tr>
<tr>
<td>$0222 8D 06 AO</td>
<td>STA TILL</td>
<td>$1386 + 2 = 5000, so f = 100 Hz, Tp = 0.01s.</td>
</tr>
<tr>
<td>$0224 CE 00 AO</td>
<td>STA PCR</td>
<td>LDA $FF Now get the low-order byte of the count.</td>
</tr>
<tr>
<td>$0226 A9 01</td>
<td>LDA $01 Initialise PB0 to be an output pin.</td>
<td></td>
</tr>
<tr>
<td>$0228 8D 08 AO</td>
<td>STA T2LL</td>
<td>Set period of square wave on PB7 so that Tp = 0.01 second.</td>
</tr>
<tr>
<td>$022A A9 FF</td>
<td>NEXT</td>
<td>Start square wave running.</td>
</tr>
<tr>
<td>$022C CE 00 AO</td>
<td>STA T2CH</td>
<td>Set up pulse counter T2 to start at $FFFFF.</td>
</tr>
<tr>
<td>$022E 8D 09 AO</td>
<td>STA T2CH</td>
<td>Start counting pulses when the event pulse is detected.</td>
</tr>
<tr>
<td>$0230 AE 00 AO</td>
<td>LDA PB6</td>
<td>Start counting pulses when the event pulse is detected.</td>
</tr>
<tr>
<td>$0232 AD 06 AO</td>
<td>STA T2HC</td>
<td>Clear IFR flag.</td>
</tr>
<tr>
<td>$0234 8D 00 AO</td>
<td>STA T2LL</td>
<td>Read the interrupt flag register. Mask all except IFR. Wait until flag is set, then the result is displayed.</td>
</tr>
<tr>
<td>$0236 FA 00</td>
<td>BNE T2CH</td>
<td>Then timing is finished, so convert the answer to decimal and display it.</td>
</tr>
<tr>
<td>$0238 85 01</td>
<td>STA LEAST</td>
<td>Preset the flip-flop by toggling PB6.</td>
</tr>
<tr>
<td>$023A 88 00</td>
<td>STA MDST</td>
<td>Most-significant byte of time.</td>
</tr>
<tr>
<td>$023C 85 02</td>
<td>STA MIDST</td>
<td>Middle byte.</td>
</tr>
<tr>
<td>$023E 85 03</td>
<td>STA MOST</td>
<td>Most-significant byte of time.</td>
</tr>
<tr>
<td>$0240 A9 FF</td>
<td>NEXT</td>
<td>Measure another interval.</td>
</tr>
</tbody>
</table>

SUBROUTINE CNVD

<table>
<thead>
<tr>
<th>Location</th>
<th>Opcode</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0240 38</td>
<td>CNVD</td>
<td>SEC Set for carry for subtractions that follow.</td>
</tr>
<tr>
<td>$0242 A9 FF</td>
<td>LDA $FF</td>
<td>Find ($FFFF - N2) = number of pulses counted.</td>
</tr>
<tr>
<td>$0244 ED 09 AO</td>
<td>SBC T2CH</td>
<td>High-order byte stored in CNTHI.</td>
</tr>
<tr>
<td>$0246 85 00</td>
<td>STA CNTHI</td>
<td>Now get the low-order byte of the count.</td>
</tr>
<tr>
<td>$0248 A9 FF</td>
<td>LDA $FF</td>
<td>Low-order byte stored in CNTHL.</td>
</tr>
<tr>
<td>$024A ED 08 AO</td>
<td>SBC T2CL</td>
<td>Conversion of hex to decimal starts here.</td>
</tr>
<tr>
<td>$024C 85 10</td>
<td>STA CNTHL</td>
<td>X contains number of bits to convert.</td>
</tr>
<tr>
<td>$0250 85 00</td>
<td>STA CNTDO</td>
<td>Shift one bit at a time into the carry flag.</td>
</tr>
<tr>
<td>$0252 8F</td>
<td>ROL CNTHL</td>
<td>X will serve as a counter for a triple-precision addition, with LEAST, MIDST, and MOST holding the answer.</td>
</tr>
<tr>
<td>$0254 85 11</td>
<td>STA CNTHI</td>
<td>Increment X to zero, then three bytes have been added.</td>
</tr>
<tr>
<td>$0256 46</td>
<td>DEY</td>
<td>Decrement Y until all the bits have been used.</td>
</tr>
<tr>
<td>$0258 80</td>
<td>ROL CNTHL</td>
<td>When X = 0, conversion is complete.</td>
</tr>
<tr>
<td>$025A 8A</td>
<td>ROL CNTHL</td>
<td>Jump to AIM 65 Display Routine.</td>
</tr>
<tr>
<td>$025C A5 00</td>
<td>LDA $00</td>
<td>Now clear the counter locations to get the time for the next two pulses.</td>
</tr>
<tr>
<td>$025E 85 01</td>
<td>STA LEAST</td>
<td></td>
</tr>
<tr>
<td>$0260 85 02</td>
<td>STA MIDST</td>
<td></td>
</tr>
<tr>
<td>$0262 85 03</td>
<td>STA MOST</td>
<td></td>
</tr>
<tr>
<td>$0264 60</td>
<td>RTS</td>
<td>Return to the timing program.</td>
</tr>
</tbody>
</table>

We will assume that the events produce positive pulses, and we will not try to describe how the positive pulses can be produced. Rather, our problem will be restricted to measuring the time between two successive positive pulses. A circuit and a program to accomplish this are shown in Figure 4 and Table X, respectively.

The circuit was inspired by Carlin's and Howard's article on the Intel 8255 in Computer Design, May 1979, pg. 213. The positive pulses clock a 7474 flip-flop, producing a logic-one voltage at the Q output of the 7474 at the time interval between the leading edges of the two pulses. With the T1 timer producing square waves on PB7, the logic-one voltage on the Q output opens the gates to the P66 (by means of the 7400 NAND gate), where they are counted by the T2 counter/timer. For example, if a square wave whose frequency is 10 Hz (Tp = 0.1 second) is applied to the 7400 NAND gate, and 250 such pulses are counted on P66, then the corresponding time interval is (250)(0.1) = 25.0 seconds, with a resolution of 0.1 second.

Clearly, no software is required to detect the pulses, and consequently very narrow pulses can be detected. Also, the programmer has control over the frequency of the square wave applied to the NAND gate. The resolution can be changed from 4.0 microseconds to 0.10 microseconds by varying the number loaded into T1.

Refer again to Table VI for a choice of frequencies for the free-running mode of the T1 timer that might be appropriate for a given application. Since the T2 timer is capable of counting to 65536, the maximum time interval that can be measured with a square wave whose period is $T_p$ is:

$$T_{\text{max}} = 65536(T_p)$$

where $T_{\text{max}}$ is the maximum time interval that can be measured, $T_p$ is the period of the square wave that is applied to PB7, $N$ is the number loaded into T1, and $T_c$ is the system clock period.

Refer again to Figure 4. When the second pulse occurs, the Q output of the 7474 flip-flop makes a transition to logic one. This also signals the conclusion of the timing interval. If Q is connected to CB1, the 6522 can be programmed to set a flag in the IFR when the logic-zero-to-logic-one transition on CB1 occurs. At this time the T2 counter/timer can be read, the result converted to decimal,
APPENDIX A. LOW—OVERHEAD CLOCK MODIFICATION

SUBROUTINE AIMDSP

The largest number of pulses from PB7 that can be counted on pin PB6 by the T2 counter/timer is $FFFF + 1$ or 65536. Each memory location is capable of storing two BCD digits, thus three memory locations are required to store a number as large as 65536. These three memory locations have addresses $0001$, $0002$, and $0003$; namely those locations that contain the decimal equivalent of the output of the T2 counter/timer at the conclusion of the timing interval, and $N_1$ is the number in the T1 timer. Refer to Table VI for the necessary $N_1$ to produce a suitable $T_p$. Values of $T_p$ that are multiples of ten are most useful. The origin of the number $FFFF$ in the equation lies in the fact that the T2 counter/timer is loaded with $FFFF$ before timing begins. For the listing shown in Table X, $T_p$ is 0.01 seconds, so the equation becomes:

$$T_m = 0.01 (FFFF - N_2)$$

The precision with which one can measure the true time $T$ between the events depends on the resolution, $T_p$, since clearly the true time need not be an exact integral number of $T_p$. Our analysis shows that the actual time, $T$, is given by the expression:

$$T_m - 1.5 T_p < T < T_m + 1.5 T_p$$

Thus, if greater precision is required, then $T_p$ can be reduced.

The conversion subroutine, CNVD, performs the operation ($FFFF - N_2$) shown in the equations. To get $T$, this number must be converted to decimal and then multiplied by $T_p$ which, in our case, is 0.01 seconds. The hexadecimal to decimal conversion algorithm used in CNVD is from Peatman’s book Microcomputer Based Design, while the coding used is from Butterfield’s “Multimode Adder” in 6502 User Notes, No. 13, pg. 23.

Subroutine CNVD also calls a subroutine named AIMDSP. This routine displays the contents of locations with addresses $0001$, $0002$, and $0003$; namely those locations that contain the time $T$, now in decimal. No attempt has been made to locate the decimal point in these subroutines. As long as the period, $T_p$, if the square wave on PB7 is a multiple of ten, 0.01 second for example, the user should have no trouble placing his decimal point mentally.

In any case, subroutine AIMDSP is an AIM 65 dependent subroutine that has been published previously, so only its AIM 65 mini-disassembly format is given here. Owners of other microcomputer systems will want to substitute a suitable routine to display the contents of storage locations.
of the three locations mentioned. Such routines for the KIM-1 and SYM-1 are readily available.

The time interval chosen for the listing in Table X is suitable for "stopwatch" functions, and a suitable stopwatch interface to the circuit of Figure 4 is given in Figure 5. This circuit simply debounces the switch when it is momentarily closed at the beginning and the end of the interval to be timed. Phototransistor circuits can also be used to produce positive pulses when light beams are interrupted. A photoplethysmograph can be used to measure the time interval between heartbeats, turning the circuit of Figure 4 into a cardiotachometer.

One way to test the circuit of Figure 4 and the program in Table X is to apply a square wave of known frequency to the clock input on the 7474. For example, if the pulses from the signal conditioner shown in Figure 3 are applied to the 7474, then the time interval should be 1/60 of a second. Since 1/60 = 0.01666, and if $T_p = 0.0001$ second ($N_1 = 0030$ from Table VI), then the number 1666 should be displayed for the time between successive positive pulses. Be sure to change the bytes at $0221$ and $0226$ to $30$ and $00$, respectively, in Table X if you make this test.

Finally, if an event can be made to produce a single positive pulse for its duration, the length of the event may be measured using a slightly modified form of the program in Table X and the circuit shown in Figure 6.

In conclusion I should like to point out that the programs and circuits given are the simplest ones I could construct. You will want to add more elegant features. The purpose of this article was to introduce a few basic techniques, not to present elaborate designs. If you come up with a neat design as a result of something you learned here, I would be very interested in getting a letter from you. Better yet, write up your circuit and program and publish both in MICRO. Although the circuits and programs described here were intended to be building blocks for more elaborate microprocessor based designs, the stopwatch interface and timing program could be used for "time and motion" studies around the house. Just make sure your spouse's motions do not make you lose track of the time!

Editor: Portions of this article are from Dr. De Jong's forthcoming book tentatively entitled 6502 Microcomputing, to be published by Howard W. Sams and Company, and scheduled for release later this autumn.
The HDE DM816-MD1 Mini Disk System is the peripheral you have been waiting for. No longer bounded by long and unreliable cassette saves and loads, your computer becomes a sophisticated system for program development or general purpose use. With the HDE Mini-Disk you load and save programs in seconds, not minutes or hours. And, since all transfers to and from the Mini-Disk are verified for accuracy, the data will be there when you need it.

The HDE DM816-MD1 Mini-Disk has been “systems” engineered to provide a complete and integrated capability. Software and hardware have been built as a team using the most reliable components available. The systems software includes the acclaimed and proven HDE File Oriented Disk System and Text EDitor, requiring only 8K for the operating software and overlay area. Systems expanding programs available include the two-pass HDE assembler, the Text Output Processing System and Dynamic Debugging Tool. Hardware includes a Western Digital 1771 based controller in a state-of-the-art 4½ x 6½” card size, Shugart SA 400 drive and the Alpha power supply.

The storage media for the DM816-MD1 is the standard, 5⅛” mini diskette readily available at most computer stores, and HDE has designed the system so that the diskettes rotate only during disk transactions, favorably extending media life. A disk format routine included with the system, formats the diskettes, verifies media integrity by a comprehensive R/W test and checks drive RPM. Additional utilities provide ascending or descending alpha numeric sort, disk picking, text output formatting, file renaming, file addressing and other capabilities.

The HDE mini-disk system is available direct or from these Fine Dealers:

<table>
<thead>
<tr>
<th>JOHNSON COMPUTER</th>
<th>PLAINSMA N MICROSYSTEMS</th>
<th>ARESCO</th>
<th>LONG ISLAND COMPUTER GENERAL STORE</th>
<th>LONE STAR ELECTRONICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box 523</td>
<td>Box 1712</td>
<td>P.O. Box 43</td>
<td>103 Atlantic Avenue</td>
<td>Box 488</td>
</tr>
<tr>
<td>Medina, Ohio 44256</td>
<td>Auburn, Ala. 36830</td>
<td>Audubon, Pa. 19407</td>
<td>Lynbrook, N.Y. 11563</td>
<td>Manchaca, Texas 78652</td>
</tr>
</tbody>
</table>

Complete with all hardware, interconnecting cables, FODS, text editor and user and installation manuals.
Card Shuffling Program for KIM - 1

Your 6502 might play poker like Amarillo KIM, but does it always have to pass the deal? Not if you teach it to shuffle cards!

Entertaining friends with computer games certainly makes all the effort of assembling a personal computer worthwhile. However, if you happen to have a small microcomputer with limited memory and very few software tools, there are not many games available. As an example, most card games need a random number generator to shuffle cards.

The standard method to generate random numbers (as used in most BASIC interpreters) is not suitable for this purpose. Since some of the bare-bone computers do not even have the software to perform multiplication, it is asking too much for them to generate floating-point random numbers. To make these small computers more entertaining, a simple method to shuffle cards is described here. This method is implemented in a KIM. The machine instructions use about 80 bytes. There is lots of memory left for playing card games. The only drawback is that it requires the operator to press the interrupt key in order to stop the program.

The card shuffling program consists of two portions. The second portion is the main program that shuffles cards. It just keeps on shuffling until the interrupt key is pressed. The first portion is an interrupt service routine used to ensure an orderly ending of the program. The program is relocatable, and the two portions can be in separate locations.

This feature makes it easy to incorporate the shuffling program into a complete card-playing program. However, it is important that the user initialize the interrupt vectors to jump to the interrupt service routine.

To keep the computer code relocatable, the initialization of the 2 byte address is left to the user. The storage area for the cards, together with 4 bytes of working space, are in page 0. In this program, the storage area starts at address 0001. However, the program can be changed easily to move the storage area to other locations in page 0.

The deck of cards is stored in an array at locations (hex) 0001 to 0034. The value of

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0120</td>
<td>0200</td>
</tr>
<tr>
<td>0130</td>
<td>0200 A2 36</td>
</tr>
<tr>
<td>0140</td>
<td>0202 8A</td>
</tr>
<tr>
<td>0150</td>
<td>0203 95 00</td>
</tr>
<tr>
<td>0160</td>
<td>0205 CA</td>
</tr>
<tr>
<td>0170</td>
<td>0206 D0 FA</td>
</tr>
<tr>
<td>0180</td>
<td>0208 86 38</td>
</tr>
<tr>
<td>0190</td>
<td>020A A5 35</td>
</tr>
<tr>
<td>0200</td>
<td>020C 38</td>
</tr>
<tr>
<td>0210</td>
<td>020D B9 34</td>
</tr>
<tr>
<td>0220</td>
<td>020F B0 FB</td>
</tr>
<tr>
<td>0230</td>
<td>0211 18</td>
</tr>
<tr>
<td>0240</td>
<td>0212 69 35</td>
</tr>
<tr>
<td>0250</td>
<td>0214 AA</td>
</tr>
<tr>
<td>0260</td>
<td>0215 B5 35</td>
</tr>
<tr>
<td>0270</td>
<td>0217 B5 00</td>
</tr>
<tr>
<td>0280</td>
<td>0219 B5 37</td>
</tr>
<tr>
<td>0290</td>
<td>021B A5 36</td>
</tr>
<tr>
<td>0300</td>
<td>021D 0A</td>
</tr>
<tr>
<td>0310</td>
<td>021E 0A</td>
</tr>
<tr>
<td>0320</td>
<td>021F 18</td>
</tr>
<tr>
<td>0330</td>
<td>0220 65 36</td>
</tr>
<tr>
<td>0340</td>
<td>0222 18</td>
</tr>
<tr>
<td>0350</td>
<td>0223 69 01</td>
</tr>
<tr>
<td>0360</td>
<td>0225 85 36</td>
</tr>
<tr>
<td>0370</td>
<td>0227 18</td>
</tr>
<tr>
<td>0380</td>
<td>0228 65 35</td>
</tr>
<tr>
<td>0390</td>
<td>022A 38</td>
</tr>
<tr>
<td>0400</td>
<td>022B E9 33</td>
</tr>
<tr>
<td>0410</td>
<td>022D B0 FB</td>
</tr>
<tr>
<td>0420</td>
<td>022F 18</td>
</tr>
<tr>
<td>0430</td>
<td>0230 69 34</td>
</tr>
<tr>
<td>0440</td>
<td>0232 AA</td>
</tr>
<tr>
<td>0450</td>
<td>0233 B4 00</td>
</tr>
<tr>
<td>0460</td>
<td>0235 A5 37</td>
</tr>
<tr>
<td>0470</td>
<td>0237 95 00</td>
</tr>
<tr>
<td>0480</td>
<td>0239 A6 35</td>
</tr>
<tr>
<td>0490</td>
<td>023B 94 00</td>
</tr>
<tr>
<td>0500</td>
<td>023D A5 38</td>
</tr>
<tr>
<td>0510</td>
<td>023F C9 00</td>
</tr>
<tr>
<td>0520</td>
<td>0241 00 C7</td>
</tr>
</tbody>
</table>

ORG $0200
LDXIM $36
L1 TXA
STA2X $00
DEX
BNE L1
STXZ $38
L2 LOOP LDAZ $35
SEC
SBCIM $34
BCS L2
CLC
LDAZ $35
ADCI $35
TAX
LDAZ $35
STAZ $35
LDAZ $00
STAZ $37
LDAZ $36
ASLA
ADCL $36
CLC
ADCIM $01
STAZ $36
CLC
ADCI $35
L3 SEC
SBCIM $33
BCS L3
CLC
ADCI $34
TAX
LDY2X $00
LDAZ $37
STAZX $00
LDXZ $35
STY2X $00
LDAZ $38
CMPIM $00
BEQ LOOP

Hark Chan
P.O. Box 714
Cambridge, MA 02139
After a sufficient number of operations, each address is distinct and is between hex 1 to 34 (decimal 1 to 52). After the interrupt key is pressed, the interrupt service routine sets a memory location, hex 0038, that serves as a flag to signal the end of the shuffling. This routine also restores the accumulator and the X and Y registers. It is important that the user initialize the interrupt vector to address the service routine instead of the operating system.

The sequence of cards being shuffled is actually predetermined because it is calculated from a prescribed series of operations. However, if the stop command is activated by a human operator the cards can be very random. I takes about 10^4 seconds to do one shuffle. The time to activate the stop command can easily vary by more than 0.1 second.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.

The program uses a simple random number generator to generate random pointers with values between 1 and 52. The position of all the cards is repeated.
How Do You Connect Peripherals to Your Superboard II

Since I wrote “A Close Look at the Superboard II”, MICRO 11:15, I have received several calls and letters asking for more information concerning interfacing the Superboard II to various peripherals — printers, memory boards and so on. Because of the continuing lack of information available from OSI, the manufacturer of the Superboard and the Challenger 1P, I have decided that it would be good to give some basic and rather general pointers on the use of the Superboard ports.

Since there are many different peripherals (understatement of the century) and since each one has its own requirements, I cannot be very specific about your particular device. Instead, I hope to describe the signals available on the Superboard in some detail, so that you will at least know something about its interfacing possibilities.

The J2 Port

There are four ports on the Superboard. Three of them are 12-pin Molex connectors and one of them is a 40-pin DIP socket. They are numbered J1 through J4. I shall begin with J2, since you are already using that one to interface your video monitor and your cassette. You will find a listing of the pin outs for J2 in Figure 1. Pins 7 through 10 are used for the cassette. Pins 11 and 12 are used for the video output.

I assume that you understand the basic use of these pins; and so, I will only mention that the signals generated for the cassette come from an on-board interface consisting of a Motorola 6850 ACIA and a couple of flip flops (U64). The audio input goes through an RCA 3130 which triggers a monostable one-shot and sets or resets a flip flop. This signal is then fed to the 6850.

The signals at the 6850 are designated as RxData and TxData. The 6850 also has two control signals which are not used by the cassette interface but might be useful to your peripheral. They are designated as RTS and CTS on the schematics.

Finally, there are two separate clocks which drive the 6850: TxCLK and RxCLK. These clocks set the baud rate at which the 6850 operates. For precise formation on the 6850, I suggest that you get a copy of the manufacturer’s spec sheet on this ACIA. Your dealer should have it.

Since I wrote “A Close Look at the Superboard II”, MICRO 11:15, I have received several calls and letters asking for more information concerning interfacing the Superboard II to various peripherals — printers, memory boards and so on. Because of the continuing lack of information available from OSI, the manufacturer of the Superboard and the Challenger 1P, I have decided that it would be good to give some basic and rather general pointers on the use of the Superboard ports.

Since there are many different peripherals (understatement of the century) and since each one has its own requirements, I cannot be very specific about your particular device. Instead, I hope to describe the signals available on the Superboard in some detail, so that you will at least know something about its interfacing possibilities.

The J2 Port

There are four ports on the Superboard. Three of them are 12-pin Molex connectors and one of them is a 40-pin DIP socket. They are numbered J1 through J4. I shall begin with J2, since you are already using that one to interface your video monitor and your cassette. You will find a listing of the pin outs for J2 in Figure 1. Pins 7 through 10 are used for the cassette. Pins 11 and 12 are used for the video output.

I assume that you understand the basic use of these pins; and so, I will only mention that the signals generated for the cassette come from an on-board interface consisting of a Motorola 6850 ACIA and a couple of flip flops (U64). The audio input goes through an RCA 3130 which triggers a monostable one-shot and sets or resets a flip flop. This signal is then fed to the 6850.

The signals at the 6850 are designated as RxData and TxData. The 6850 also has two control signals which are not used by the cassette interface but might be useful to your peripheral. They are designated as RTS and CTS on the schematics.

Finally, there are two separate clocks which drive the 6850: TxCLK and RxCLK. These clocks set the baud rate at which the 6850 operates. For precise formation on the 6850, I suggest that you get a copy of the manufacturer’s spec sheet on this ACIA. Your dealer should have it. I mention all of this simply because these six signals are present as TTL signals on J2, pins 1 through 6. If your peripheral requires TTL level serial data, then you will connect it to these pins.

But there is more to it than just connecting your peripheral’s cable to the right pins on J2. My Superboard II came with several parts missing. You will need to install a 7417 at U68 and a 74LS14 at U67. You will also have to install the 220 and 390 ohm resistors at R38 through R49.

Next, notice that the RxData and CTS signals coming in on pins 1 and 3 respectively are called RxData3 and CTS3 after they come from U67. “They are then routed to jumper locations W10 (the upper W10 to the right of Q2 in the schematic sheet 6) and W11. The reason for this is that you don’t want input coming from two or three different sources going to the 6850.

I recommend that you install a DPDT (double pole double throw) switch so that you can switch the RxData line going to the 6850 between RxData1, which is the cassette input; RxData3, which is the TTL level input from J2; and the RS-232 input which will be described shortly. The other pole of this switch can be used to switch CTS appropriately. To install this switch you only have to cut the trace connecting the RxData lines to RxData1 at W10.

With this switch installed, you can switch lines between three sources of input: the cassette, your peripheral on TTL level lines at J2, and some other peripheral that uses RS-232 on J3.

One more change may be needed at jumper location W5, also on sheet six of the schematic. Here, the TxCLK is wired to the RxCLK. To separate them, you merely have to cut the diagonal trace connecting them and install another switch to switch the RxCLK line on the 6850 between the TxCLK line and the RxCLK input. I recommend, however, that you not make this modification unless you need separate clocks for your peripheral. If your peripheral is pretty stable and close to 300 baud, you can probably get by as is. But if you have a peripheral that has a clock rate different from 300 baud, you will need to make this modification.

You may now ask what the RTS and CTS signals are used for. If your peripheral is a printer, it may send out a busy signal whenever it is not ready to receive another character. This signal should be active high. It should be connected to the CTS on the 6850 — that is, it should be connected to J2 pin 3. You will have to switch W11 properly, since the CTS goes through this junction. You may also have a TTL line which controls the power on/off on your peripheral. Maybe you would like to control the cassette motor. You can do this with the RTS signal. It is a signal provided by the 6850 under software control; that is, your software, since OSI doesn’t support this function.

Because it is fed through a 7417 buffer which is capable of sinking 30 milliamps, you can use it to drive a small reed relay. I purchased just such a relay, which operates on 5 volts at about 20 milliamps, and have used it to turn my cassette on and off. See Figure 2 for a schematic used to connect a relay to the RTS signal.

Now all the connections are made, but how do you instruct the computer to...
transmit and receive these signals? Remember that the cassette is also connected to the 6850; and so, as far as software is concerned, the peripheral will work just like the cassette. Whatever you write to your cassette will go to the TxData line and to your peripheral. You read your peripheral just as you would read from the cassette (after you switch W10 over).

Let us suppose that you have a printer connected to the TxData line and that it sends a busy signal back over the CTS line when it is working. Whenever you give the command to “SAVE” in BASIC, this will activate the printer just as it does the cassette, so that any characters output by BASIC will be sent to both printer and cassette. If either of them is turned on, it will print or record the data sent. And how can one tell whether the printer is busy or not? You can’t without writing some of your own software.

You see, Microsoft BASIC does not actually do any I/O; it merely jumps out to the I/O routine provided by OSI in the monitor. There are four routines that BASIC jumps to for I/O: one which inputs a character, one which outputs a character, one which is executed whenever the LOAD command is given, and one which is executed whenever the SAVE command is given. BASIC jumps to the following addresses which have instructions as shown:

Input FFEB JMPI $0218
Output FFEE JMPI $021A
Load FFF4 JMPI $021E
Save FFF7 JMPI $0220

The monitor stores the addresses of the input, output, load, and save routines at the locations $0218, $021A, $021E, and $0220 respectively every time the BREAK key is pressed. This makes BASIC transfer control to these routines when it needs I/O.

Of course, it would be easy to write your own routine and POKE the address of it in one of these locations so BASIC would then jump to your routine instead of the one in the monitor. You can disassemble the routines in the monitor, of the one in the monitor. You can then jump to your routine instead of it in one of these locations so BASIC transfer control to these routines at the locations $0218, $021A, $021E and $0220 after each time you depress the BREAK key.

The short output routine shown here illustrates how one might check for a printer busy signal. The listing includes two small programs that turn the RTS signal off and on. The latter might be employed to write a SAVE routine that could be called from BASIC and would turn the cassette or printer on automatically. Remember that you will have to put the addresses of your I/O routines in locations $0218, $021A, $021E and $0220 after each time you depress the BREAK key.

The J3 Port

The main purpose for J3 is to interface peripherals which require RS-232 signals. As can be seen in Figure 1, pins 2 and 3 are the data out and in pins. Pin 7 provides a negative voltage for the RS-232 interface. To use this, however, you will have to open the ground at jumper W10, the lower one under Q1. Even more than this, you will have to install all the hardware for the RS-232 signal level generation; that is, Q1 and Q2 and their associated resistors and diode. Once again you must set up W10 and W11 with the proper switch, as described previously, so that you can switch between the cassette and your peripheral. I believe that the description for J2 was sufficient to get you going on the software you might need to use this port.

The J4 Port

In the OSI manual on the Superboard, J4 is described as a “joystick” and “noise” port. The noise is made by turning on and off four of the keyboard

The output routine, located at $FF69, jumps to the CRT simulator routine at $BF2D which outputs a character to the screen and then checks the save flag at $0205. If the save flag is 0 it returns. If the save flag is non-zero, it outputs the character to the 6850. If this character was a carriage return (that is, $0D) then it also sends out 10 nulls ($00).

The load routine, located at $FF96, sets the save flag to 1. When you give the SAVE command, BASIC jumps to the save routine which sets the save flag. Then, whenever you output any character, BASIC jumps to the output routine which sends the character not only to the CRT, but also to the 6850. This will send it to the cassette and also to your printer. If you don’t turn on your cassette, the character will only be printed by the printer.

But I still haven’t described how you know when the printer is busy. You can PEEK at the 6850 control status register to see whether the RTS bit is low. Then you will know the printer is ready. But this is not a very good way to do it, since you would have to do such PEEK-ing prior to every print command! The better way is to write a short output routine which checks this bit for itself.

The 6850 occupies two address locations: $F000 and $F001. The first of these is the control register of the 6850 and, by writing and reading this address, one can send and receive control signals. $F001 is the data register and, by writing or reading this address, one can send and receive data from the 6850.

The short output routine shown here illustrates how one might check for a printer busy signal. The listing includes two small programs that turn the RTS signal off and on. The latter might be employed to write a SAVE routine that could be called from BASIC and would turn the cassette or printer on automatically. Remember that you will have to put the addresses of your I/O routines in locations $0218, $021A, $021E and $0220 after each time you depress the BREAK key.
### Figure 1: Superboard I/O Ports

<table>
<thead>
<tr>
<th>J1 Pin</th>
<th>J1 Signal</th>
<th>J2 Pin</th>
<th>J2 Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IRQ</td>
<td>1</td>
<td>RxData</td>
</tr>
<tr>
<td>2</td>
<td>NMI</td>
<td>2</td>
<td>RxCLK</td>
</tr>
<tr>
<td>3</td>
<td>DD</td>
<td>3</td>
<td>CTS</td>
</tr>
<tr>
<td>4</td>
<td>BD0</td>
<td>4</td>
<td>TxD ata</td>
</tr>
<tr>
<td>5</td>
<td>BD1</td>
<td>5</td>
<td>TxC LK</td>
</tr>
<tr>
<td>6</td>
<td>BD2</td>
<td>6</td>
<td>RTS</td>
</tr>
<tr>
<td>7</td>
<td>BD3</td>
<td>7</td>
<td>Mic .05 volt</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>8</td>
<td>GND</td>
</tr>
<tr>
<td>9</td>
<td>GND</td>
<td>9</td>
<td>AUX 0.5 volt</td>
</tr>
<tr>
<td>10</td>
<td>unused</td>
<td>10</td>
<td>Audio in</td>
</tr>
<tr>
<td>11</td>
<td>unused</td>
<td>11</td>
<td>GND</td>
</tr>
<tr>
<td>12</td>
<td>A2</td>
<td>12</td>
<td>Video out</td>
</tr>
<tr>
<td>13</td>
<td>A1</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>A0</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>A3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>A4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>A5</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>A6</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>A7</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>A8</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>A9</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>A10</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>A11</td>
<td>23</td>
<td>-V in for RS232 interface</td>
</tr>
<tr>
<td>24</td>
<td>A12</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>25</td>
<td>A13</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>A14</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>A15</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>GND</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>GND</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>GND</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>02</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>R/W</td>
<td>32</td>
<td>R1</td>
</tr>
<tr>
<td>33</td>
<td>BD7</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>BD6</td>
<td>34</td>
<td>R7</td>
</tr>
<tr>
<td>35</td>
<td>BD5</td>
<td>35</td>
<td>C1</td>
</tr>
<tr>
<td>36</td>
<td>BD4</td>
<td>36</td>
<td>C2</td>
</tr>
<tr>
<td>37</td>
<td>GND</td>
<td>37</td>
<td>C3</td>
</tr>
<tr>
<td>38</td>
<td>GND</td>
<td>38</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>GND</td>
<td>39</td>
<td>C5</td>
</tr>
<tr>
<td>40</td>
<td>GND</td>
<td>40</td>
<td>C6</td>
</tr>
</tbody>
</table>

Latches. These are coupled through resistors and a capacitor to pin 12 of J4. The main problem is that the resistors are not installed, nor are their values given. I have not experimented enough with these to determine what values would work best to give four bit analog output.

The main reason I have not done this experimentation is that I have not thought the "noise" would be very useful because it is coupled to the keyboard. For this reason, whenever the keyboard input routine is called, a tone is generated by a loop in that routine which sets and resets the keyboard latches.

If you wanted to produce some music, you could do so by choosing proper values for these resistors and then writing a small program to turn on and off these latches by writing to address $DF00. I would advise installing a switch between the output of pin 12 and your amplifier since you will want to turn off this noise whenever you are not generating some music or gaming sound effects. The keyboard routine's continuous tone is rather annoying after a while!

If you want a beeper to signal various conditions audibly, then I recommend that you use the RTS output at J2. It comes from a heavy buffer which could be connected through a 100 ohm resistor and a small speaker to the 5 volt line. When this RTS signal is turned on and off at the proper rate, it would make a nice beeper without the need for the amplifier that the output at J4 pin 12 requires. Also, there would be no annoying continuous tone.

The other pins on J4 are quite useful because they are connected directly to the keyboard matrix. The graphics manual has a short description of how to deactivate the CTRL-C routine and how to check for a key depressed. If you were to connect lines 1 through 11 on J4 to some switches, you could use the procedure to determine whether the switches were closed. In this way, one might simulate a joystick.

By using four switches you could indicate eight directions. North, east, south and west could be indicated when exactly one switch was closed — the switch in that particular direction on your joystick. Northeast, southeast, southwest and northwest could be indicated by two adjacent switches being closed at the same time. By this means you could move a point on the screen in any of eight directions.

Another very good use for these lines would be to add a numeric keypad in parallel with the keyboard. To do so, you need only wire the switches on the keypad so they are in parallel with the corresponding keys on the keyboard as shown in the schematic, sheet 12. See
The J1 Port

This port is what OSI uses for expansion. It has all the data and address lines in addition to several of the control lines that the 6502 produces. I suggested in my previous article that this socket could be connected to a KIM type connector to make a KIM expansion port. That is more or less true but, as you will see from checking the signals available on J1 and the required signals on the KIM expansion port, there are a few missing. The most important ones are there, and it just may be that the ones you need to operate your peripheral memory board or whatever are present.

Pin 3, the DD line, needs some explanation. This line is an incoming signal that is used to control the data buffers U6 and U7. This line must be driven by the RD/W signal, so I suggest that you connect both the RD/W signal (that you get from U21 pin 6) and the line from J1 pin 3 to the RD/W pin on the KIM expansion connector.

I think a 40 wire ribbon cable with a DIP plug on the end of it would be the best thing to make the connection from J1 to the KIM connector. Of course, some of the wires won't be used, and so, you might be able to pull some of the unused wires out and solder them to the points on the Superboard where you are going to get the missing signals.

The missing signals can be found at the following places: RD/W on U21 pin 6 as mentioned above, 02 on U11 pin 4, RST on the high (non-ground) side of the BREAK key, VCC where the 5 volt supply line enters the board, VSS any place along the edge of the board where the ground plane is, SYNC on J8 pin 7, and 01 on U8 pin 3. If you need the RDY signal, you have to make a change on the Superboard. Open the short trace coming from U8 pin 2, which is the RDY line on the 6502, and put a 4.7K pull up resistor in the opening you have made. This will enable any peripheral that needs to use the RDY line to pull it low.

After installing the resistor, you can wire the RDY line to U8 pin 2.

There are also R0, K6, SHIFT OUT, RAM/R/W, and PLL TEST line: on the KIM expansion connector, but you won't be able to get these from the Superboard. I doubt that any of the peripherals you might be interested in will require them since they are rather peculiar to the KIM.

This method of directly wiring a KIM socket to the appropriate signals on the Superboard will give you a workable KIM expansion connector even though it may look a little messy since you have to run wires to several points on the Superboard. If you plan to use several boards simultaneously, you will want to make your connections to a KIM compatible motherboard.

You may want to invest just what OSI offers in the way of peripherals before you make any of these changes and additions to your Superboard. In any case, I hope that you now understand a little more about how your Superboard works and how you might go about connecting some peripherals to it.

---

PRINT CRG (Wherever you want it)
CRT EQU $BF2D
STATUS EQU $F000
DATA EQU $F001
SAVFLG EQU $0205
CSR CRT OUTPUT TO CRT
FHA SAVE CHARACTER
LDA SAVFLG CHECK SAVE FLAG
BEQ RTN IF 0 NO 6850 OUTPUT
WAIT LDA STATUS WAIT FOR
LSRA CHARACTER
LSRA TO BE TRANSMITTED
BCC WAIT
WAIT1 LDA STATUS WAIT FOR
ANDIM $08 PRINTER
BNE WAIT1 READY
READY P.A WHEN READY
STA DATA OUTPUT DATA
RTN RTS
CAS OFF LDAIM $51
STA STATUS
RTS
CASON LDAIM $11
STA STATUS
RTS
WE’VE GOT YOUR COMPUTER

C1P: $349! A dramatic breakthrough in price and performance. Features OSI’s ultra-fast BASIC-in-ROM, full graphics display capability, and large library of software on cassette and disk, including entertainment programs, personal finance, small business, and home applications. It’s a complete programmable computer system ready to go. Just plug-in a video monitor or TV through an RF converter, and be up and running. 15K total memory including 8K BASIC and 4K RAM — expandable to 8K.

C1P MF: $995! First floppy disk based computer for under $1000! Same great features as the C1P plus more memory and instant program and data retrieval. Can be expanded to 32K static RAM and a second mini-floppy. It also supports a printer, modem, real time clock, and AC remote interface, as well as OS-65D V3.0 development disk operating system.

C2-4P: $598! The professional portable that has over 3-times the display capability of 1P’s. Features 32 x 64 character display capability, graphics, full computer type keyboard, audio cassette port, and 4 slot BUS (only two used in base machine). It has 8K BASIC, 4K RAM, and can be expanded to 32K RAM, dual mini-floppies and a printer.

C2-4P MF: $1533! It’s a big personal computing mini-floppy system at a special package price. Contains the famous C2-4P microcomputer with 20K static RAM, 5” mini-floppy unit for instant program and data loading, RS-232 circuitry (for optional modem and printer), and diskettes featuring exciting games, personal, business and education applications.

C2-8P DF: $2599! A full business system available at a personal computer price! The system includes the powerful C2-8P microcomputer (32K RAM expandable to 48K), dual 8” floppy unit (stores 8-times as much information as a mini-floppy), and 3 disks of personal, educational and small business applications software. Has all the capabilities of a personal system including graphics plus the ability to perform Accounting, Information Management, and Word Processing tasks for small business.

C2-8P: $799! The personal class computer that can be expanded to a full business system. Has all the features of the C2-4P plus an 8 slot BUS (3-times greater expansion ability than the C2-4P), Can be expanded to 48K RAM, dual floppy, hard disk, printer and business software.

I'm interested in OSI Computers. Send me information on:

- Personal Computers
- Small Business Computers
- Educational Systems
- Industrial Development Systems

Name
Address
City    Zip
Phone

COMPUTER SHOP
Boston
530 Comm. Ave
(Received from B.U.)
267-0700

Shrewsbury
2538 Main St
(Received from M.I.T.)
247-0700

Cambridge
209 Massachusetts Ave
(Received from M.I.T.)
247-0700

603-473-2323

(1-800) 269-0700

(1-617) 661-2670

Boston,
247-0700

Cambridge
268 Norfolk St
(Received from M.I.T.)
247-0700

(1-617) 661-2670
SALES FORECAST provides the best forecast using the four most popular forecasting techniques: linear regression, log trend, power curve trend, and exponential smoothing. Neil D. Lipson’s program uses artificial intelligence to determine the best fit and displays all results for manual intervention. $9.95

CURVE FIT accepts any number of data points, distributed in any fashion, and fits a curve to the set of points using log curve fit, exponential curve fit, least squares, or a power curve fit. It will compute the best fit or employ a specific type of fit, and display a graph of the result. By Dave Garson. $9.95

PERPETUAL CALENDAR may be used with or without a printer. Apart from the usual calendar functions, it computes the number of days between any two dates and displays successive months in response to a single keystroke. Written by Ed Hanley. $9.95

STARWARS is Bob Bishop’s version of the original and best game of intergalactic combat. You fire on the invader after aligning his fighter in your crosshairs. This is a high resolution game, in full color, that uses the paddles. $9.95

ROCKET PILOT is an exciting game that simulates blasting off in a rocket ship. The rocket actually accelerates you up and over a mountain; but if you are not careful, you will run out of sky. Bob Bishop’s program changes the contour of the land every time you play the game. $9.95

SPACE MAZE puts you in control of a rocket ship that you must steer out of a maze using paddles or a joystick. It is a real challenge, designed by Bob Bishop using high resolution graphics and full color. $9.95

MISSILE ANTI-MISSILE displays a target on the screen and a three dimensional map of the United States. A hostile submarine appears and launches a pre-emptive nuclear attack controlled by paddle 1. As soon as the hostile missile is fired, the U.S. launches its anti-missile controlled by paddle 0. Dave Moteles’ program offers high resolution and many levels of play. $9.95

MORSE CODE helps you learn telegraphy by entering letters, words or sentences, in English, which are plotted on the screen using dots and dashes. Ed Hanley’s program also generates sounds to match the screen display, at several transmission speed levels. $9.95

POLAR COORDINATE PLOT is a high resolution graphics routine that displays five classic polar plots and also permits the operator to enter his own equation. Dave Moteles’ program will plot the equation on a scaled grid and then flash a table of data points required to construct a similar plot on paper. $9.95

UTILITY PACK 1 combines four versatile programs by Vince Corsetti, for any memory configuration.

POSTAGE AND HANDLING
Please add $1.00 for the first item and $.50 for each additional item.

• Programs accepted for publication
• Highest royalty paid

TO ORDER
Send check or money order to:
P.O. Box 273
Plymouth Meeting, PA 19462
PA residents add 6% sales tax.

U.S. and foreign dealer and distributor inquiries invited
All programs require 16K memory unless specified
Name: Text Processing System (Editor and Assembler)
System: APPLE II
Memory: 24K
Language: Integer BASIC and Machine Language
Hardware: APPLE II, 24K and Disk II

Description: This disk based system allows you to create and edit Applesoft, Integer BASIC, assembly language, and APPLE DOS exec files. The text editor provides capabilities to create load, modify and save APPLE II disk operating system text files. Editing features include simple-to-use data entry, extensive character and string searches and replacement, block line movement, and simple single line macros. Text creation and modification is further simplified with such features as tabbing, specific search windows, file merging, and line deletion. The text editor supports systems equipped with a printer to create permanent listings of text files.

The assembler is a complete, disk-based, two pass symbolic assembler. You can assemble up to ten disk based text files at any one time. The assembler will generate disk based binary files that can be executed via the APPLE DOS “BLOAD” or “BRUN” commands. Eight character symbols allow for meaningful variable and routine names. The assembler supports all 56 standard 6502 opcodes and six additional pseudo-opcodes used to define constants, labels, program addresses, etc. Meaningful error messages are generated to help locate program mistakes. The assembler can generate both program and symbol table listings, with optional line printer output.

Copies: 200
Price: $55.00 plus $1.00 shipping and handling
   California residents add 6 per cent sales tax
   Includes: All programs on a diskette and a complete 60-page user’s manual.

Author: Jeffrey Gold
Available from:
    Software Concepts
    Box 1112
    Cupertino, CA 95014

Name: Household Finance Program
System: APPLE II
Memory 32K
Language: Integer BASIC and Machine Language
Hardware: APPLE II, 32K, and Disk II

Description: The household finance program is a comprehensive household record maintenance and budget management program. This disk based system provides the capability to maintain 175 records a month for 12 months (that's over 2000 records on a single diskette). With a simple to use data entry mode, a user can enter check transactions, deposits, and cash expenditures.

Error correcting is a simple matter with a complete set of editing features. Twelve user definable budget categories are available to allow a family to plan and analyze spending patterns. Check and cash expenditures can be assigned to any budget category. Both month-to-date and year-to-date budget summaries are available. Additionally, the program will provide data on how well the family is keeping to its established monthly budgets. Previously entered financial records can be retrieved via a comprehensive data listing mode.

Other program features include checkbook balancing, tax deductible classification, and single disk drive copy (backup) to protect against data loss. The program supports systems equipped with a printer and can provide user selected permanent listing via a unique page print mode. This software package is the most complete, easy-to-use home financial program available today.

Copies: 20
Price: $31.95 plus $1.00 postage and handling
   California residents must add 6 per cent sales tax
   Includes: All software supplied on a program diskette with a complete 32-page user’s manual.

Author: Jeffrey Gold
Available from:
    Software Concepts
    Box 1112
    Cupertino, CA 95014
Name: Belais' Master Index to Computer Programs in BASIC
System: All
Memory: N/A
Language: BASIC (a few programs require machine language routines)
Hardware: N/A

Description: A directory of computer programs written in BASIC. The programs are ones that have appeared in ten major home computer magazines. They cover both business and personal applications. All major computer systems are included. Many of the programs are written specifically to take advantage of the capabilities of such 6502-based computers as the PET and the APPLE II. The reviews provide detailed information about what each program does, and what hardware and software it needs. Program listings are not provided, but information is given on where to get them.

Price: $9.95 plus $1.00 shipping
   California residents must include 6 per cent sales tax.
   Includes: Book
Order Info: VISA/Master Charge accepted (give account number, expiration date, and signature).

Author: Paul Belais
Available from:
   Falcon Publishing
   Dept. Y
   P.O. Box 688
   Ben Lomond, CA 95005

Name: Mailing List Program
Memory: 48K with DOS and Applesoft ROM
Language: Applesoft II
Hardware: APPLE II, disk drive, printer

Description: The mailing list program is a disk based, menu driven program written in Applesoft II. In order to use the program, a 48K system with Applesoft II on firmware along with one disk drive and DOS 3.2 is required. If your system does not have Applesoft II on firmware, the mailing list program can still be used but the number of entries will be greatly reduced.

The program is able to maintain a complete mailing list. The mailing list data base can be changed, sorted, searched, added, deleted and reformatted. There are five types of sort and five types of search. Labels can be printed out on a 40, 80 or 132 character printer and also viewed on the screen for rapid editing. The program accommodates zip codes with seven digits for use outside the U.S.A.

There is a routine for lining up the labels and for setting the spaces between the labels. Provision has also been made to make a backup copy of the data with a single disk drive. The mailing list program makes generation and maintenance of a mailing list very quick and simple.

Price: $34.95 for diskette plus $1.25 shipping
   Includes: User manual and documentation.

Author: Gary E. Haffer
Available from:
   Software Technology for Computers
   P.O. Box 428
   Belmont, MA 02178

Name: Black Box
System: APPLE II
Memory: 16K
Language: Integer BASIC
Hardware: Cassette

Description: The program Black Box is based on the Parker Brother's game of the same name. The object of the game is to guess the positions of marbles that are hidden on an eight by eight board. To help you find the marbles, rays are sent into the box. These rays can hit a marble, be deflected by a marble, be absorbed into the box, or any combination of these! There are full instructions inside the program, and a sample game to get you going. Test your reasoning power against the mystical Black Box!

Price: $8.00
   Includes: Verified cassette, postage and handling

Author: Robin Hodgson
Available from:
   The AppleCorp
   103 Horizon 14
   723 14th St. N.W.
   Calgary, Alberta
   Canada
   T2N 2A4

Name: APPLE—DOC
System: APPLE II
Memory: 3.5 to 5.8K depending on options.
Language: Applesoft II

Description: Set of three programs—VARDOC, LINEDOC, and REPLACE.

VARDOC produces a list of every variable used in your program and all the lines each is used on. Screen and/or printer output can include optional descriptors of each variable.

LINEDOC produces a list of every line called by a GOTO, GOSUB, etc, and all the lines each is called from. You are even alerted to calls to lines no longer in the listing. Optional descriptors are for each line number.

REPLACE allows you to easily rename any or all occurrences of any variable in your program. Even change variables types! Can also be used to replace constants or referenced line numbers within the listing. The Literal Mode allows you to replace any set of characters or BASIC statements with any other set. This program is especially useful when appending subroutines with conflicting variable use.

Price: $9.95 for cassette, $13.95 for diskette.
   California residents must add 6 per cent sales tax.
   Includes: Three programs plus documentation.

Author: Roger Wagner
Available from:
   Local Computer Stores or
   Southwestern Data Systems
   P.O. Box 582
   Santa Fe, CA 92071
   (714) 562-3670, SASE for free information
Name: **Roger's Easel**
System: **APPLE II**
Memory: 16K for Integer and Applesoft ROM, 20K for Applesoft RAM

Description: Set of three programs: Roger's Easel, Lo-Res Link-Integer, and Lo-Res Link-Applesoft. A paddle oriented sketching program using the color graphics of the APPLE II. The unique features of this set include the ability to store and retrieve user created pictures from tape or disk, ability to erase with a single keystroke, returning original color when done, and immediate access to a detailed help list while in the program. The most outstanding feature is the option of permanently linking up to 41 pictures to any Integer or Applesoft program for instant recall at any time. Besides being just plain fun, applications range from putting more creative screen images in your game programs to educational programs for younger children involving shape or color recognition.

Price: **$9.95** on cassette, **$13.95** on diskette
Includes: Three program set with ten-page manual.

Author: **Roger Wagner**
Available from:
- Local APPLE dealers or:
  - Southwestern Data Systems
    - P.O. Box 582-MC
    - Santee, CA 92071
    - (714) 562-3670

Name: **Programmer's Utility Pack**
System: **APPLE II**
Memory: 4K to 6K (for the program itself) depending on the program used.
Language: **Integer and Applesoft**
Hardware: **APPLE II** with cassette or disk drive

Description: Set of 11 programs. Appends, STR$( ) and VAL( ) are on printed documentation with the tape version. Programs include: Renumber — Integer & Applesoft, Append — Integer and Applesoft, Line Find — Integer and Applesoft, Address/HEX Converter, Screen Find, Memory Move, and the STR$( ) and VAL( ) function simulations for Integer.

By using the various programs one can renumber Integer and Applesoft programs with all GOTO's, etc, being renumbered and the user alerted to unusual situations in the program. These include reference line numbers not in the program, lines referenced by a variable or expression, and a number of others.

Line Find allows the user to locate the actual address range of a line in memory so as to be able to insert CLR, HIMEM;,... etc. It can also be used on occasion to recover programs garbled by dropped bits. Address/HEX Converter converts between the HEX, Integer, and Applesoft address formats. It also provides the two byte breakdown of numbers greater than 256 for use in pointers, etc.

Screen Find is used for printing directly on the screen by POKEing appropriate values into the proper locations in memory. Screen Find gives these values and locations when the characters desired and the horizontal and vertical screen positions are input. Memory Move allows one to move blocks of memory up or down any number of bytes from Integer or Applesoft. The Monitor has a routine similar to this, but it cannot be used to move blocks up a small distance and it is not possible to use it directly from Applesoft.

**STR$( )** simulates the function of this name in Applesoft for use in Integer programs. **STR$( )** in Applesoft converts a number to a string. **VAL( )** is similar but converts strings to numbers.

Copies: **Just Released**
Price: **$16.95**. Calif. residents add 6 per cent sales tax.
Includes: Two cassettes or one diskette plus documentation.

Author: **Roger Wagner**
Available from:
- Local Apple dealers or:
  - Southwestern Data Systems
    - P.O. Box 582-MC
    - Santee, CA 92071
    - (714) 562-3670

Name: **Softtouch Utility Pac II**
System: **APPLE II**
Memory: 24K with DOS
Language: **Integer and Applesoft BASIC**
Hardware: **Disk drive**

Description: Set of nine programs on disk. Programs include: checkbook update to DOS, update electronic index III, auto-write instructions, find hidden control characters, slow/stop list, disk space, listing headers and exec reader. A complete listing is provided for all programs and programming.

Checkbook update rewrites your original checkbook program for use with the disk drive. Routines have been added to change accounts or list bank names with account numbers, etc. Index update rewrites Bob Bishop's electronic index file for complete automation. A printing routine has been added for hard copy.

Auto write appends subroutines to existing programs, converts integer BASIC listings to Applesoft or vice versa. Auto write documentation gives detailed instructions for using the program to patch in lines in any part of a program or delete illegal lines such as 65535, etc. Find hidden control character displays any control character buried in a catalog name, or any listing for both integer or Applesoft BASIC. Disk space is written in Applesoft and gives sectors and bytes left on a diskette. No text files are created by the program and operating time is three seconds. Slow/stop list may be loaded in and used continuously after switching disks or languages. Exec reader will read text files for all of the above with the exception of index file.

Price: **$13.95**
Includes: One diskette plus documentation.

Author: **Dr. Nick Romano**
Available from:
- Softtouch
  - P.O. Box 511
  - Leominster, MA 01453
A Warning:
The **MacroTeA™**
is for Professional
Programmers — and Very
Serious Amateurs — Only

Now: a machine language pro-
gramming powerhouse for the
knowledgeable programmer who
wants to extend the PET’s capa-
bilities to the maximum. The
MacroTeA, the Relocating Macro
Text Editor:Assembler from Skyles
Electric Works.

The Skyles MacroTeA is a super
powerful text editor. 26 powerful
editing commands. String search and
replace capability. Manuscript feature
for letters and other text. Text loading
and storage on tape or discs. Supports
tape drives, discs, CRT, printers and
keyboard.

The Skyles MacroTea is a relocating
machine language assembler with true
macro capabilities. A single name
identifies a whole body of lines. You
write in big chunks, examine, modify
and assemble the complete program.
And, when loading, the MacroTeA goes
where you want it to go. Macro and
conditional assembly support. Auto-
matic line numbering. Labels up to 10
characters long.

The Skyles MacroTeA is an enhance
Monitor. 11 powerful commands to
ease you past the rough spots of
program debugging.

The Skyles MacroTeA is a warm
start button. Over 1700 bytes of
protected RAM memory for your object
code.

There’s no tape loading and no
occupying of valuable RAM memory
space: The Skyles MacroTeA puts 10K
bytes of executable machine language
code in ROM (from 9800 to BFFF —
directly below the BASIC interpreter).
2K bytes of RAM (9000 to 97FF).

Like all Skyles Products for the PET, it’s practically plug in
and go. No tools are needed. And, faster than loading an
equivalent size assembler/editor from tape, the MacroTeA is
installed permanently.

The Skyles MacroTeA: 13 chips on a single PCB. Operates
interfaced with the PET’s parallel address and data bus or with
the Skyles Memory Connector. When ordering, indicate if the
MacroTeA will interface with a Skyles Memory Expansion
System. You can save $20. Specifications and engineering are
up to the proven Skyles quality standards. Fully warranted for 90
days. And, as with all Skyles products, fully and intelligently
documented.

VISA, Mastercharge orders call (800) 227-8388 (Except Calif.)
California orders please call (415) 494-1210.

Skyles Electric Works
10301 Stonydale Drive, Cupertino, CA 95014, (408) 735-7891

---

**Hypocycloids**

E.D. Morris
3200 Washington Street
Midland, MI 48640

A modification to John Sherburne’s original program
plots hypocycloids quite a bit faster, on the OSI, by
reducing the number of revolutions required. The
technique may be used on any micro.

---

I had just added the extra 2K of
memory to my Ohio Scientific 440 video
board to implement the graphics option,
and was wondering what to do with
those 16,384 dots (128 x 128) staring
out from my monitor. I happened to pick
up the March 79 issue of MICRO and was
intrigued by John Sherburne’s article on
plotting hypocycloids. A hypocycloid, if
you don’t remember, is what you get
when one circle rolls inside another as in
the “Spirograph” toy. I immediately ac-
cepted the challenge that if it can be
done on a PET, I could do it better on my
micro.
The original hypocycloid program suffered greatly from lack of speed since each point was calculated using four trigonometric functions. Approximately 300 points per revolution were required. Even then, some gaps appeared in the resulting pattern. I was able to reduce the number of points calculated per revolution to 30 by drawing straight line segments between calculated points. This makes the resulting curves not quite as smooth, but very acceptable as the accompanying photos illustrate. The number appearing in the lower left corner indicates the number of revolutions required to complete the figure.

Below is the subprogram I used to fill in the space between calculated points (X1,Y1) and (X2,Y2). A different procedure is used depending whether the slope of the plotted line is nearer the X axis or Y axis. Lines 1060-1065 and 1160-1165 store the bit in memory and are specific to my graphics board. I would be happy to provide a copy of the full program to anyone who is using the OIS 440 board with graphics.

1000 IF X1= X2 THEN 1100
1010 k = (Y2-Y1)/(X2-X1)
1015 IF ABS(k)>1 THEN 1100
1020 B=Y1-A*X1+0.5
1030 FOR X3=X2 TO X1 STEP SGN(X1-X2)
1040 Y3=INT(B+A*X3)+0.5
1060 M=54272+16*Y3+INT(X3/8)
1065 P0KEM,PEEK(M)ORS(X3AND7)
1070 NEXTX3:RETURN
1100 IF Y1=Y2 THEN RETURN
1110 A=(X2-X1)/(Y2-Y1)
1120 B=X1-A*Y1+0.5
1130 FOR Y3=Y2 TO Y1 STEP SGN(Y1-Y2)
1140 X3=INT(B+A*Y3)
1160 M=54272+16*Y3+INT(X3/8)
1165 P0KEM,PEEK(M)ORS(X3AND7)
1170 NEXTY3
1180 RETURN

To Order PROGRAMMER'S Toolkit or MACROTeA —
Custom designed to plug into your PET. So, when ordering, please indicate if your Toolkit:

... will be used with the Skyles Memory Expansion System, or...
... will be used with the ExpandaPet, or Expandmem
... will be used with the PET 2001-8 alone
... will be used with the PET 2001-16, -32 (chip only)

Your MacroTeA, Custom designed for your PET. So specify your PET model when ordering. $395.00*

(Important Savings: If it's to be used with a Skyles Memory Expansion System, the MacroTeA can plug directly into the Skyles connector. So, you save $20. The Skyles MacroTeA is only $375.00 when interfaced with the Skyles Memory Expansion System.)

Send your check or money order to Skyles Electric Works, VISA, 227-8398. (California residents; please phone (415) 494-12 0.)
Ten Day Unconditional Money-Back Guarantee on all products sold by Skyles Electric Works, except chip only.

Your MacroTeA is a product of Harry Saal and his associates at Palo Alto ICs.

So, if you really want to be into BASIC programming — and you want to have fun while you're doing it, order your BASIC Programmer's Toolkit now. We guarantee you'll be delighted with it.

Is Programming Fun?

Have More Fun, Make Fewer Errors, Complete Programs Much Faster...with the BASIC PROGRAMMER'S Toolkit

Now you can modify, polish, simplify, add new features to your PET programs far more quickly while reducing the potential for error. That all adds up to more fun... and the BASIC Programmer's Toolkit.

The magic of the Toolkit: 2KB of ROM firmware on a single chip with a collection of machine language programs available to you from the time you plug in your PET to the time you shut it off. No tapes to load or to interfere with any running programs. And the Programmer's Toolkit installs in minutes, without tools.

Here are the 10 commands that can be yours instantly and automatically guaranteed to make your BASIC programming a pleasure:

AUTO RENUMBER DELETE
HELP TRACE STEP
OFF APPEND DUMP
FIND

Every one a powerful command to insure more effective programming. Like the HELP command that shows the line on which the error occurs... and the erroneous portion is indicated in reverse video:

HELP

... Or the TRACE command that lets you see the sequence in which your program is being executed in a window in the upper corner of your CRT:

READY

The Programmer's Toolkit is a product of Harry Saal and his associates at Palo Alto ICs.

So, if you really want to be into BASIC programming — and you want to have fun while you're doing it, order your BASIC Programmer's Toolkit now. We guarantee you'll be delighted with it.

Send your check or money order to Skyles Electric Works, VISA, 227-8398. (California residents; please phone (415) 494-12 0.)
Ten Day Unconditional Money-Back Guarantee on all products sold by Skyles Electric Works, except chip only.

Skyles Electric Works 10301 Stonydale Drive, Cupertino, CA 95014, (408) 735-7891
DAM SYSTEMS by ImC
A complete system of modules to let you computer listen to the real world.

DAM SYSTEMS PRICE LIST

<table>
<thead>
<tr>
<th>DAM SYSTEMS components</th>
<th>CABLE &quot;A&quot; - Interconnect Cables TBA</th>
<th>CABLE A24 - Interconnect Cable TBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIM16I - Analog Input Module</td>
<td>$179.00</td>
<td>$19.95</td>
</tr>
<tr>
<td>POWI - Power Module</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td>ICON - Input Connector</td>
<td>$9.95</td>
<td></td>
</tr>
<tr>
<td>OCON - Output Connector</td>
<td>$9.95</td>
<td></td>
</tr>
<tr>
<td>MANMOD1 - Manifold Module</td>
<td>$59.95</td>
<td></td>
</tr>
<tr>
<td>ANAMANI - Analog Manifold Module</td>
<td>TBA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SENSORS</th>
<th>TBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDES: Sensors for temperature, pressure, flow, humidity, level, motion, etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMPUTER INTERFACES</th>
<th>TBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDES: Connects DAM SYSTEMS SENSORS to the AIM16 without soldering - sensor cables are supplied.</td>
<td></td>
</tr>
</tbody>
</table>

| PETMOD - PET Interface Module | $49.95 |
| SIMMOD - KIM Interface Module | $39.95 |
| MANUAL AND DISPLAY MODULE | |
| EXPANDER MODULE | |

<table>
<thead>
<tr>
<th>DAM SYSTEMS sets</th>
<th>TBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIM16: Starter Set 1</td>
<td>$189.00</td>
</tr>
<tr>
<td>AIM16: Starter Set 2</td>
<td>$239.00</td>
</tr>
<tr>
<td>PETSET1</td>
<td>$295.00</td>
</tr>
<tr>
<td>KINSET1</td>
<td>$285.00</td>
</tr>
</tbody>
</table>
SYM-1 6532 Programmable Timer

The 6532 interval timer is useful as a backup timekeeper or as a loop controller. It can be accessed in two ways, independent of the interrupt system, and employed to meet a variety of realtime program requirements.

In addition to the programmable ports and interval timers located in the 6522s, the SYM-1 has an interval timer in the 6532. The 6532-style device is also used on the KIM-1, and so knowing how to use the SYM timer properly will help in understanding KIM programs and enable the SYM programmer to adapt KIM programs for use on his SYM more easily.

The 6532 timer does not have its IRQ line connected to the IRQ input of the 6502. Therefore, lacking direct access to the interrupt structure, we are unable to get as precise a level of timing as with the onboard 6522s. However, if an extra timer or loop controller is required, the 6532 may prove to be useful.

Before using the timer in the 6532, one must first clear the interrupt flags. Since all of the features we intend to use are part of the write-protected memory, we must first of all allow access to this area. This is accomplished by:

```
20 86 8B JSR ACCESS
```

Then, to clear the interrupt flag (PA7 flag), we will read the interrupt flag register. This may be accomplished by reading any one of four locations: A405, A407, A41D or A41F, typically by executing the instruction:

```
AD 05 A4 LDA INTREG
```

After this instruction is executed, the interrupt flag register will contain "80". This register will be cleared to "00" when we write a value into the timer register. We may then go back occasionally during program execution, test to see if the flag register is still zero, and branch if it is not zero.

As another alternative, we can do a BIT test on the flag location, checking only the timer flag for the branch condition. This method has been used in the sample program. If the BIT test is used, it is not necessary to read the interrupt register in order to clear the PA7 flag because this flag will not be tested. The initial read instruction then becomes redundant.

At this point, we must decide how many clock cycles are to elapse before the timer flag becomes set. We will write the selected value into the counter.

There are four different points at which to enter data into the counter, A41C, A41D, A41E and A41F. These are indicated in the manual as 1T, 8T, 64T and 1024T. These multiples mean that any data which is entered into the counter will begin at that particular point and decrement with the rate of the clock frequency (1T), or at one decrement for each eight clock cycles (8T), one decrement for each 64 clock cycles (64T) or one decrement for each 1024 clock cycles (1024T).

There is only one timer register, but the four addresses mentioned above are the means by which the frequency pre-divider is set. For example, if we write "01" into location A41E, the timer flag is reset and, 64 clock cycles later, the timer flag is set again. If we write "7F" into location A41F, instead, then the timer flag will not be reset until 1024 clock cycles have elapsed.

Just as an example, let's say we wanted 800 clock cycles to elapse before the timer flag is set. We will be reading the flag register periodically to see if it is non-zero, determine whether the flag gets set, and branch on the non-zero condition. Writing decimal 00 (hex 64) into location A41D sets the pre-divider to 8 then, 8 x 100 = 800 ticks later, the timer reaches zero and the flag is set.

While the counter is independently decrementing, we can determine the current timer contents at any time by reading one of these four locations: A404, A405, A41C, A41E. There are four readable locations due to "don't care" addressing modes or incomplete address decoding.

One might be tempted to look at the timer contents, occasionally, and branch when the counter reaches zero. This does not offer a good chance for success as the following example will show.

Let's say we've written "0A" (decimal 10) into location A41D (8T) so that 80 cycles later the timer will count down to zero. Suppose we do the following during the counting period:

```
(A) Increment a memory location
(B) Test timer contents
(C) Branch back if non-zero
```

If the sequence of operations takes seven machine cycles, then after 77 cycles the timer will still be at "01" and after 77 + 7 = 84 cycles the timer will contain a count of zero since more than 80 cycles have elapsed. Right? Wrong! Unfortunately, it will contain "FC" instead! The limitation of this counter is that, as soon as zero is reached and the flag is set, the counter continues to decrement, but it no longer matters which counter multiple was being used because as the counter immediately begins to free-run decrement at the 1T rate.

To overcome this limitation, since we do not use the IRQ and since we only sample occasionally, we will generally read the interrupt register, testing for a non-zero figure, rather than reading the timer and looking for zero contents as shown above.

Now we come to an example program which ties everything together and demonstrates the use of this timer. Location 20D may be set for any desired timer value. Location 20F may be set to 1, 1C, 1D, 12E, or 1F depending upon whether you want to operate the timer with a prescale of 1T, 8T, 64T, or 1024T. You will notice that the loop of instructions between locations 211 and 224 takes a total of 28 machine cycles to execute.

Begin program execution at location 200. The display will light, upon completion indicating how many times the program was able to traverse the loop before the timer flag became set.

Robert A. Peck
1276 Riesling Terrace
Sunnyvale, CA 94087

October, 1979
MICRO -- The 6502 Journal
Having trouble running mnemonically-entered programs on your AIM-65? This might be one source of the problem.

According to the AIM-65 User's Guide, indirect indexed addressing mode may be entered by using either "(HH,Y)" or "((HH)Y)" where "HH" is a hexadecimal byte. The AIM-65 Summary Card lists the alternatives "(HH,Y)" or "((HH,Y)"

However, only the format "((HH)Y)" will assemble correctly.

The formats (HH,Y) and ((HH,Y) will be assembled incorrectly as indexed indirect instructions, "(HH,X)"

Don Stein
6012 Chatsworth Lane
Bethesda, MD 20014

DON LANCASTER'S INCREDIBLE SECRET MONEY MACHINE

A cookbook for creating your own computer or tech venture.

SYNERGETICS BOX 1077 M
THATCHER, AZ 85552

象征符号表2000 2:2A
ACCESS 8FB6  DSCAN  023B  OUTBYT 82FA  OUTDSP 89C1
SCAND 8906  TMIN  0211  TMOUT 0227

While working on a leasing rate calculation program in Kim BASIC I found the need for a list of variables available so that I could cross out the ones I used in my program. I found such a list in MICRO 4:4 and decided to write a program, in BASIC, to print it when needed.

Henri Reiher
4236 Madison
Montreal, CANADA H4B 2T9

100 REM PROG TO SHOW NUMERICAL AND STRING VARIABLES AVAILABLE IN
110 REM MICROSOFT BASIC AS USED IN PET-APPLE-TRS80 AND OTHERS
115 REM REF: MICROSOFT BASIC APRIL-MAY 78 PAGE 4
120 FOR X = 65 TO 90
125 PRINT
130 PRINT CHR$(X);" "
135 FOR Y = 0 TO 9
140 IF X = 65 TO 90
145 Y$ = CHR(X) + NUM(I) + " "
147 REM INSTEAD OF NUM(Y) YOU CAN USE STR$(Y)
150 PRINT Y$;
155 NEXT Y
160 NEXT Z
170 PRINT
175 PRINT
180 NEXT Z
190 END

17:56 MICRO -- The 6502 Journal October, 1979
More LETTERS

I have a SYM-1. While debugging a program that uses the timer in the 6532 I found out that the IRQ pin is not connected to the IRQ bus. Rather than spend a lot of time finding the neatest way to connect the 6532 IRQ pin to the IRQ bus, I simply ran a piece of wire wrap stock between the IRQ pin on the 6532 to the nearby 6522. Now I can use the interrupt feature of the 6532. I do not know whether Synertek did this for a particular reason but I have not had any problems since making this little modification. Perhaps you are already aware of this. I just thought I would pass it along, for what it is worth.

Keith Le Baron
1260 S. Blackhawk
Freeport, IL 61032

There is a useful, but unadvertised, display subroutine in the AIM-65 Monitor. It is labeled OUTDD1, and can be called by a JSR instruction to hex address EF7B.

The subroutine displays the ASCII character which is in the accumulator, at the relative position (0 - 19 decimal, or 0 - 13 hexadecimal) indicated by the X register. It returns with A and X contents intact.

Before calling the subroutine, be sure to ORA #80, or else the hardware cursor will be displayed.

Don Stein
6012 Chatsworth Lane
Bethesda, MD 20014

[Editor’s Note: Marvin De Jong demonstrated the use of this subroutine in an earlier issue of MICRO. Since, however, Don Stein independently “found” it and thinks that it is important enough to point out to other AIM users, we are printing his letter.]

[Editor’s Note: If you have some small bit (byte?) of information that you wish to pass on to fellow computerists, a short letter to MICRO is one simple way to “pass the word along”.]

DISK DRIVE WOES? PRINTER INTERACTION?
MEMORY LOSS? ERRATIC OPERATION?
DON’T BLAME THE SOFTWARE!

ISO-1
ISO-2
Power Line Spikes, Surges & Hash could be the culprit!
 Floppies, printers, memory & processor often interact!
Our unique ISOLATORS eliminate equipment interaction
AND curb damaging Power Line Spikes, Surges and Hash.
*ISOLATOR (ISO-1A) 3 filter isolated 3-prong sockets;
integral Surge/Spike Suppression; 1875 W Maximum load,
1 KW load any socket ............................ $54.95
*ISOLATOR (ISO-2) 2 filter isolated 3-prong socket banks;
(6 sockets total); integral Spike/Surge Suppression;
1875 W Max load, 1 KW either bank .......... $54.95
*SUPER ISOLATOR (ISO-3), similar to ISO-1A
except double filtering & Suppression ........ $79.95
*ISOLATOR (ISO-4), similar to ISO-1A except
unit has 6 individually filtered sockets .... $93.95
*ISOLATOR (ISO-5), similar to ISO-2 except
unit has 3 individually filtered sockets .. $76.95
*CIRCUIT BREAKER, any model (add-CB) Add $ 6.00
*CKT BRKR/SWITCH/PILOT any model (-CBS) ............................................. Add $11.00

Phone Orders 1-617-655-1532
Electronic Specialists, Inc.
171 South Main Street, Natick, Mass. 01760

TEXTCAST®

Turn your PET into a WORD PROCESSOR comparable to large systems for a fraction of the cost!

CREATE-REVIEW-EDIT FILES ON TAPES OR DISKS
PRINT TEXT-LETTERS-FORMS-TABLES

TEXTCAST FITS YOUR PET SYSTEM, OLD AND NEW ROMS

CLEAR INSTRUCTIONS!

Prices: Tape plus manual, $60.
Diskette plus manual, $65.
Write: Cognitive Products
P.O. Box 2592
CHAPEL HILL, N.C. 27514

PET Trademark of Commodore Business Machines, Inc.
FREE! up to $170 in merchandise with the purchase of PET-CBM item!!!

PET 16K Large Keyboard  $ 995  $130
PET 32K Large Keyboard  $1295  $170
PET 8K  $ 795  $100
PET 2040 Dual Disk (343K)  $1295  $170
PET 2023 Printer (pres feed)  $849  $110
PET 2022 Printer (trac feed)  $995  $130

KIM-1  $159  (Add $20 for Power Supply)  SYM-1  $222.00
6500 Programming Manual  6.50
2144 L 450 ns  5.90  24/5.15  100/4.45
2716 EPROM (5 Volt)  42.00
6550 RAM (for 8K PET)  12.70
6502 Microprocessor Chip  9.75
6522 VIA  9.75
6520 VIA  5.50
PET 4 Voice Music Board (MTUK-1002-2)  $ 49.00
Music Software (K-1002-3C) for PET  $ 19.00
Programmers Toolkit - PET ROM Utilities  $ 45.00
Microchess 2.0 for PET or APPLE  17.90
PET Word Processor - Machine Language  24.00
3M "Scotch" 8 disks  10/$31
3M "Scotch" 5 diskettes  SALE  10/$39
Verbatim 5 diskettes  10/$27

Cassettes (all tapes guaranteed)
Premium quality, high output low noise in 5 screw housing with labels:
C-10  10/5.95  50/25.00  100/48.00
C-30  10/7.00  50/30.00  100/57.00
WRITE FOR 6502 AND 8100 PRODUCT LIST
A B Computers
115-B E. Stump Road
Montgomeryville, PA 18936
(215) 699-8386

Put Yourself in Control with the

APPLETHROTTLE

That's right! The APPLETHROTTLE will turn your micro paddles into a speed controller. By simply pushing a button, you can stop your computer for as long as you want. Release the button, and your computer enters a slow-motion mode with your paddles controlling the speed. And if that isn't enough, look at these additional features:
- Plug into any slot
- Works with BASIC, Integer, and Applesoft
- Normal - slow - stop
- Use to LST, TRAC, RUN, etc.
- No SOFTWARE to load
- Unveil program secrets
And the's more! No more multiple LST commands to view small program sections. With the APPLETHROTTLE, you'll be able to list or trace long programs while watching your program flow in slow motion. So get in control with the APPLETHROTTLE and order yours today!

APPLETIME: Real Time Clock for the Apple II. Plugs directly into any slot and keeps time even when computer is off. Features 12/24 Hour, ECD/ASCII data format, and AC/Crystal time base selection. Includes software examples for machine, language and BASIC programming. Completely assembled and tested. APT-1 Real Time Clock $79.95

SOFTOUCH

PRESENTS IT'S
UTILITY PAC II

- CHECKBOOK UPDATE TO DISK
  AN EXEC FILE WRITES OVER YOUR CHECKBOOK
  PROGRAM TO AUTOMATICALLY UPDATE IT TO DISK
- INDEX FILE UPDATE
  AUTOMATES BISHOP'S INDEX FILE
- FIND CONTROL CHARACTER
  WILL DISPLAY CONTROL CHARACTERS ON ANY
  CATALOG OR PROGRAM LISTING
- SLOW LIST
  FULL STOP & START CONTROL WITH EXIT. WORKS
  WITH APPLESOFT OR INTEGER BASIC
- LIST HEADERS
  PUT HEADERS ON YOUR LISTINGS WITH NO LINE
  NUMBERS OR REM STATEMENTS. AP II
- AUTO WRITE
- AUTO WRITE INSTRUCTIONS
  USE EXEC FILES TO APPEND, ADD SUBROUTINES,
  OR EDIT PROGRAMS. CONVERT INTEGER TO APPLESOFT
  DELETE ILLEGAL LINE NUMBERS ETC. ETC.
- EXEC READER
  READS EXIT FILES FOR ABOVE
- DISC SPACE
  COMPLETELY WRITTEN IN APPLESOFT. WORKS
  IN 3 SECONDS. GIVES FREE SECTORS AND BYTES
  WITH LISTINGS AND DOCUMENTATION. PRICE $19.95

DISC MANAGEMENT SYSTEM

EIGHT PROGRAMS ON DISK TO PROVIDE THE USER WITH A COMPLETE UNDERSTANDING OF THE DISK DRIVE COMMANDS PLUS A UTILITY PACKAGE TO INDEX & CATEGORIZE ALL PROGRAMS WRITTEN FOR THE APPLE II COMPUTER. THE SYSTEM PROVIDES FULL SEARCH, EDITING AND DATA TRANSFER CAPABILITIES.
A TWENTY-SIX PAGE BOOKLET PROVIDES DETAILED, EDUCATIONAL TECHNIQUES GIVING A THROUGH UNDERSTANDING OF THE DISK COMMANDS.
SYSTEM REQUIREMENTS: DISK II A APPLESOFT TAPE OR ROM
PRICE $19.95 ON DISK FOR EITHER OF ABOVE
(PROCESSED & SHIPPED WITHIN 4 DAYS)
SEND CHECK OR MONEY ORDER TO:
SOFTOUCH
P.O. BOX 511
LEOMINSTER, MASS. 01453

WEST SIDE ELECTRONICS
P.O. Box 636, Chatsworth, CA 91311
We pay all shipping in Continental U.S.
Others add 10%. California residents add 8% tax

SYNERGISTIC SOFTWARE
5221 120th Ave. S.E.
Bellevue, WA 98006

The Adventure games combine the exciting graphics and sound effects capabilities of the APPLE II with the fascinating complexity of a mythical adventure game. Monsters, hazards, obstacles, weapons, magical devices and evil powers confront the player at every turn as you gather treasure and try to reach your goal. Two adventures are now available:

DUNGEON CAMPAIGN - Full color graphics
subterranean adventure. 16K required.
CASS, $12.50 DISK $15.00

WILDERNESS CAMPAIGN - HiRes graphics
surface adventure. 48K required.
CASS, $15.00 DISK $17.50

GET BOTH ON ONE DISK FOR $30.00
(WA Res. add 5.3% sales tax)
A Real-Time Clock for OSI Disk Systems

Did you know that your OSI disk-based system has most of the hardware you need for a real-time clock already built in? Here is information on how to use it.

Robert T. Kintz
104 Council Rock Avenue
Rochester, NY 14610

For most personal and business applications, the need for keeping track of time is either not very great or can be handled by special software routines for particular applications. Where microcomputers are involved in process control operations, however, such as in the real-time control of laboratory experiments, precise timekeeping is a must. Here the initiation and sequencing of most computer-controlled events must be held in tight lock-step with a real time clock.

Owners of Ohio Scientific Challenger II and III disk-based systems may not be aware that provision for a real-time clock already exists on their 470 disk controller board. The bottom middle section of this board contains the PC foils to mount three 74390 decade counter IC's. These divide the on-board 1 MHz crystal clock to provide pulses ranging from 1 to 100,000 per second, selectable at the user's option.

Timing pulses may be fed into the NMI or IRQ lines of the OSI bus (pins 2 or 3) where the 6502 will see them as interrupt signals. The software to handle an interrupt-driven, time keeping routine must have been loaded into memory prior to turning the clock on, or it may be permanently located in PROM at a convenient memory address.

One example of how the hardware may be implemented is shown in Figure 1. A 0.1 Hz clock pulse from the third 74390 is fed into both inputs of a two-input nand gate (7400) after passing through a switch located on the front panel. The 7400 may be conveniently located in the prototyping area just below the three 74390's on the 470 board.

The second input to the two nand gates is taken from bit "0" of a 6821 PIA located on the 500 or 510 CPU board. The outputs of the two 7400 gates are fed to the NMI bus line and a front panel LED, respectively. The brightly flashing LED serves as a reminder that the clock is running, following turning the switch "on" and setting bit "0" high.

The actual interrupt handling and clock routines have been written in machine language, as shown, where they have been assembled to start at $6900 (26880). Of course, relocation of these routines, as well as the clock counters, is entirely optional. Be sure, however, that they are located above the workspace occupied by BASIC or other applications programs.

A BASIC demonstration program incorporating the clock is also shown. Lines 50-70 set up the PIA on the CPU board (63232) so that ports A and B are configured as inputs and outputs, respectively. Since OSI's PROM monitor vectors to $0130 on receipt of an NMI interrupt, lines 90-100 POKE a jump to the start of the interrupt handling routine.

Next, in lines 120-140, the machine language object code is read as data and POKEd into high memory. The decimal equivalents of the object code are represented as DATA in lines 9010-9110. Lines 200-220 now set the clock counter locations to "0" and we are ready to turn the clock switch "ON".

Once this is accomplished, the clock is under program and/or keyboard control via POKEs to the PIA PORT B, bit "0." Applications programs inserted at line 300 may use the clock by PEEKing at the appropriate clock counter locations.

Figure 1
MOVING?

Please notify MICRO of any change of address so that you will not miss any issues. If we receive the change of address information by the 10th of the month, then the next issue of MICRO will be sent to the new address. We can not be responsible for replacing issues which are missed due to changes of address which you do not send in time. The Post Office does NOT return the undeliverable copies - so we lose both the postage and the magazine.

Send change of address to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824

Please include old label or your subscription number.
6502 Bibliography: Part XIII

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

478. The Cider Press 2 No. 1 (April, 1979)
Scribblemonger, John, “FORTH, Ver 1.6″, pg. 1.
Forth for the APPLE is 20 times faster than BASIC.
Silverman, Ken, “Computer Terms”, pg. 2.
APPLE terms defined and explained.
Nareff, Max J., “Max your APPLE”, pg. 2.
Another in a series of articles designed to simulate
the various MAtrix functions on the APPLE.
This BSTAT offers choice of hex or decimal and gives you
CATALOG so you can enter the name of the program
exactly with the cursor and save the program with another
cursor move on the APPLE.
Bernard, Phil, “Storing Strings on Tape, or, Is Disk
Necessary?”, pg. 3.
Anon, “Disk of the Month”, pg. 3.
Twenty-five programs on disk.
Vrooman, Gerry, “The APPLE II Memory Map De-Fogger”,
p. 4.
Explanation of where various functions are in memory.
Another version of the Hello program for the APPLE
disk combining a few new gimmicks.

479. Byte 4 No 4 (April, 1979)
Campbell, Richard, “Cross Pollinating the APPLE II”, pgs.
20-25.
A serial I/O port based on an Intel 8251 with RS-232
output.
Zimmerman, Mark, “Simulating Physical Systems — The
Two-Dimensional Ideal Gas”, pgs. 26-41.
Use your PET to experiment with physical models.
Meushaw, Robert V., “The Standard Data Encryption
Algorithm”, pgs. 110-126.
Using the KIM-1 in encryption.

480. KB Microcomputing (formerly Kilobaud) No 29 (May,
1979)
Lindsay, Len, “PET-Pourri”, pgs. 6-7.
New PET versions of the Microtechnology Unlimited
KIM music board and visible memory are in the offing.
More on tape head alignment on the PET. A TAPE TEST
program from Jim Butterfield is listed.
Anon., “OSI Small Systems Journal”, pgs. 8-11
The OSI Small Systems Journal is now published as a
section of Microcomputing.
A new control board for PET. An ADC Adapter module
for PET, and Superchip for the APPLE.
Knox, Thomas; Brazil, Ray H.; and Richardson, Robert M.
Letters discuss advantages and disadvantages of APPLE
II and TRS-80.
Pepper, Clement S., “KIMCTR”, pgs. 34-38.
This KIM-1 frequency counter/timer can be used with any
CPU with comparable features.

481. Southeastern Software Newsletter No 8 (April 1979)
2.
On the APPLE Use $FDOC, RDKEY. With several exam-
plgs, a good tutorial. Also explains exclusive OR.
McClelland, Geo., “Program to Print Applesoft Tokens”,
p. 4.
Listing of a program to supplement an earlier program to
print integer BASIC tokens.
McClelland, Geo., “Searching for a Small String Embodied
in a Larger String”, pg. 5.
Simple listing to use with files or data statements,
etc.
McClelland, Geo., “Running Disk Programs the Easy Way”,
pg. 6.
Use of the cursor makes reading in those program titles
easy.

482. 6:02 User Notes No 14 (April 1979)
Designed for a 40-column printer.
Larrabee, Robert D., “Check-Out”, pgs. 9-14.
How to check out a new program on the KIM without
having to continually hit the plus key. Back up feature.
An ability to ADD some material in the middle
of a program.
Schilling, Heinz Joachim, “BASIC Mod and Programming
Hirt”, pg 12.
A modification to correct a problem of reloading
programs on the KIM using Microsoft BASIC.
How to limit program listing to 16 lines at a time on the
KIM using Microsoft BASIC.
McKenna, Sean, “Automatic Line Number Entry Prompt for
BASIC”, pg 13.
An automatic line numbering input routine for 9-digit KIM
BASIC.
Herman, Harvey, “Renumber Addendum and Some Mods”,
p. 13.
Hints for KIM Microsoft BASIC.
Implementing the GET command in KIM BASIC.
Notes and discussions of FOCAL, Tiny BASIC, FORTH
and XPLO.
Mackay, A.M., “Accessing the SYM Displays”, pg. 18.
A program to output characters on the display.

Interfacing details for these two units.

Adams, Jim, “Wumpus and Music Box for SYM”, pg. 20.
Modifications to implement these two programs on the SYM.

Corrections for the AIM User’s Manual.

Merhar, Milan, “TVT-6 Notes and RAM Expansion”, pgs. 24-25.
TVT-6 discussion and a way to fill the lower 4K in KIM.

Prints your tape directory on your TTY or terminal.

483. Stems from Apple 2 No 4 (April, 1979)
Gustafson, Gus, “INT/FP Stop List Program”, pg. 4.
BASIC programs for Stop List.
Gustafson, Gus, “Apple Disk Copy Program”, pg. 5.
Modified program to permit using two cards and multiple drives.
Sittel, Randy, “Program FRE(0)”, pg. 5.
Routines for free bytes no matter what the memory.

484. Circuit News, April 15, 1979
Anon., “Microcomputers Monitor Oil Well Operation”
APPLE II is used in monitoring off shore oil well drilling processes, displaying information continuously on a silent 700 printer and an H-P X-Y plotter.

485. The Pet Gazette, Spring, 1979
Micro Technology Unlimited is coming out with a PET version of the KIM music board (DAC) and the visible memory.
Butterfield, Jim, “Routines from PET BASIC”, pgs. 2-6.
A listing of a large number of routines from PET BASIC.
Anon., “PET Tokens”, pg. 8.
A listing of the 255 PET Tokens.
Butterfield, Jim, “Thoughts on PET BASIC”, pgs. 10-12.
Hints for PET users. GET statements, the PET timer, precautions for amateur mechanics, print suppression, etc.
A BASIC program which loads a machine language routine into the PET.
A machine language program for tracing the progress of a BASIC program.
Instructions and discussion of the keyboard installation.
How to protect your program listing.
Aubrecht, Bob and Karl, “PET BASIC for Parents and Teachers”, pgs. 24-25.
Part 6 of this continuing tutorial.
Listing of a large number of key locations and functions.
Butterfield, Jim, “Tape Head Alignment”, pg 32.
Procedure and program listing of a tape test to help solve this important problem.

486. Design News, April 23, 1979

487. Byte 4 No 5 (May, 1979)
Pfeiffer, Erich A., “Aids for Hand Assembling Programs”, pgs. 238-244.
The article’s assembly method is used for program development on a KIM-1 microcomputer.

488. Southeastern Software Newsletter Issue No 9 (May, 1979)
A list which works with Applesoft.
Hartley, Tim, and McClelland, Geo., “Character Set”, pg. 2.
A machine code program to print the entire character set. Also a discussion of how the program works and the use of the disassembler.
Anon., “Applesoft II Merge Program”, pg. 3.
For disk or tape versions or ROM version AS II.
Hartley, Tim, “Hi-Res Drawing Program”, pgs. 4-5.
Written for a disk system with the AS II ROM card but mcds are given to change it for use on other combinations.
Anon., “Correcting Disk Files”, pgs. 5-7.
Another addition to the NAMES FILES program given in earlier issues.

489. Mi:RO No. 12 (May, 1979)
No es on getting started with the AIM 65.
Machine or assembly language coding is as easy as BASIC with this assembler.
You can look at your BASIC in ROM or other internal codes in machine language.
Hi-LO with a new twist to the game.
How real-time games can be written for the OSI Challenger systems which use a serial terminal run from the ACIA.
Good news is that only two minor hardware changes improve the high-speed cassette read/write. The KIM read/ou line is also improved. New uses for the BREAK command are given, and now the register name is displayed during the R command.
Rowe, Mike (Staff), “The MICRO Software Catalog: VIII”, pgs. 37-38.
Eleven new programs are described.
How to operate the KIM TTY link at 9600 baud.
How APPLE stores characters. A mighty article showing how to exercise considerably more control over the BASIC interpreter in your microcomputer.
Carpenter, Chuck, “Renumber Applesoft”, pgs. 45-46.
Append and renumber routines.
Index is broken down by system — APPLE, OSI, General, KIM/TIM, SYMAIM, and so on.

490. 73 Magazine No 12 (May, 1979)
Schmidt, Bill and Shattuck, Bob, “RTTY Transceiver for the KIM-1”, pgs. 78-82.
This program requires a video terminal and AFSK generator.
Introducing SEAWELL's

Little Buffered Mother
The ultimate Motherboard for any KIM-1, SYM-1, or AIM-65 system

Features:
- 4K Static RAM on board
- +5V, +12V, and -12V regulators on board
- 4 + 1 buffered expansion slots
- Accepts KIM-4 compatible boards
- Full access to application & expansion connector
- LED indicators for IRQ, NMI, and power-on
- Also compatible with SEA-1, SEA-16, the PROMMER, SEA-PROTO, SEA-ISDC, and more
- Onboard hardware for optional use of (128K addressing limit)
- Mounts like KIM-4 or with CPU board standing up
- 10 slot Motherboard expansion available - SEAWELL's Maxi Mother

Standard ...................... ..$139
w/4K RAM ..................... ..$189
Assembled Only

For further information contact:
SEAWELL Marketing Inc.
P.O. Box 17006
Seattle, WA 98107

SEAWELL Marketing Inc.
315 N.W. 85th
Seattle, WA 98117
(206) 782-9480
APPLESOFT II UTILITY

The Applesoft II Utility program provides the user with the following features. a) Complete automaticrenumbering ofany Applesoft II program. b) The creation ofan EXEC File for subroutine file creation. This feature allows you to incorporate the same subroutine in various programs. c) No modification ofthe program in machine memory (RAM). d) Automatic running of the program. No programmer should be without this excellent utility program. REQUIREMENTS: Disk II, Applesoft II, 16K of memory.

REAL ESTATE ANALYSIS PROGRAM

The Real Estate Analysis Program provides the user with three features. a) A powerful real estate investment analysis for buy/sell decisions and time to hold decisions for optimal rental/commercial investments. b) Generation of complete amortization schedules consistent with banking practices and schedules. c) Generation of depreciation schedules for selecting the best depreciation schedule for your use and a determination of optimal switch over points to straight line to maximize depreciation. All three features are designed for video screen or printer output. In addition, the program will plot: cash flow before taxes vs. years, cash flow after taxes vs. years, adjusted basis vs. years, capital gains vs. years, pre-tax proceeds vs. years, post-tax proceeds vs. years, and return on investment (%) vs. years. REQUIREMENTS: Applesoft II, 16K of memory without DOS or 32K of memory with DOS (Disk II).

ADDRESS FILE GENERATOR

A professional piece of software which allows the user to create four different types of address files: a) Holiday File, b) Birthday File, c) Home Address File, and d) Commercial Address File. The program contains a menu of seven major commands: 1) Create a File, 2) Add to File, 3) Edit File, 4) Display File, 5) Search File, 6) Sort File, and 7) Reorganize File. Most of the major commands have subordinate commands which adds to the flexibility of this powerful software system. We doubt you could buy a better program for maintaining and printing address files. REQUIREMENTS: Disk II, Apple Printer Card, 32K of memory with Applesoft ROM Card or 48K of memory without Applesoft ROM Card.

SUPER CHECKBOOK

A totally new checkbook program with a unique option . . . Bar Graphs. These bar graphs, outputted to a printer or video screen, provide trend analysis data on code expense, income, expenses, or gain/loss on a month by month basis. The program contains a total of fourteen options: 1) Check/Deposit Entry & Modification, 2) Reconciliation of Checks or Deposits, 3) Sort by Check Number, 4) Sort by Code for Year, 5) Sort by Code for Month, 6) Output Year to Date, 7) Output Month Activity, 8-11) Printer/Video Plot Trend Analysis-Bar Graphs, 12) Account Status, 13) Reconciled Check Status, and 14) Quit. An excellent program for maintaining your checkbook, or that of a small business. REQUIREMENTS: Disk II, 32K of memory with Applesoft ROM Card or 48K of memory without Applesoft ROM Card.

FUNCTION GRAPHS AND TRANSFORMATIONS

This program uses the Applesoft II high resolution graphics capabilities to draw detailed graphs of mathematical functions which the user defines in Basic syntax. The graphs appear in a large rectangle whose edges are X and Y scales (with values labeled by up to 6 digits). Graphs can be superimposed, erased, drawn as dashed (rather than solid) curves, and transformed. The transformations available are reflection about an axis, stretching or compressing (change of scale), and sliding (translation). The user can alternate between the graphic display and a text display which lists the available commands and the more recent interactions between user and program. Expected users are engineers, mathematicians, and researchers in the natural and social sciences; in addition, teachers and students can use the program to approach topics in (for example) algebra, trigonometry, and analytic geometry in a visual, intuitive, and experimental way which complements the traditional, primarily symbolic orientation. REQUIREMENTS: 16K of memory with Applesoft ROM Card or 32K of memory without Applesoft ROM Card.

Available at your local computer store

Call or write for our free SOFTWARE & ACCESSORIES CATALOG

DEALER INQUIRIES INVITED

POWERSONFT, INC.

P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

• Check or Money Order
• Include $1.00 for
Whipping and handling
• C.O.D. ($1.00 add'l. charge)
• Master Charge and VISA
orders accepted
• New Jersey residents add
5% sales tax

Programs Available on Diskette
at $5.00 Additional
1. **GRAPHICS PAC 2**  
   New Version  
   Quadruple your PET's graphic resolution. Why be stuck with the PET's cumbersome 25 x 40 1000 point display. With Graphics Pac you can directly control (set and clear) 4000 points on screen. It's great for *graphing, plotting, and gaming*. Graphics Pac allows you to plot in any combination of two modes: Quadrant graphing with (0,0) center screen, and Standard graphing with (0,0) plotted in the upper left hand corner. Complete documentation shows how you can *merge* this useful routine with any of your own programs *without retyping* either one! All this on a high quality Microsette for only $9.95.

2. **ASSEMBLER 2001**  
   A full featured assembler for your PET microcomputer that follows the standard set of 6502 mnemonics. Now you can take full advantage of the computing abilities of your PET. *Store and load* via tape, run through the SYS or USR functions. *List and edit* too with this powerful assembler. No other commercial PET assembler gives you all these features plus the ability to *look at* the PET'S secret Basic ROMs all in one program. This valuable program is offered at $15.95.

3. **BIKE**  
   An exciting new simulation that puts you in charge of a bicycle manufacturing empire. Juggle inflation, breakdowns, seasonal sales variations, inventory, workers, prices, machines, and ad campaigns to keep your enterprise in the black. *Bike is dangerously addictive*. Once you start a game you will not want to stop. To allow you to take short rest breaks. Bike lets you store the data from your game on a tape so you can continue where you left off next time you wish to play. Worth a million in fun, we'll offer BIKE at $9.95.

4. **PINBALL**  
   Dynamic usage of the PET's graphics features when combined with the fun of the *number one* arcade game equals an action packed video spectacle for your computer. Bumpers, chutes, flippers, free balls gates, a jackpot, and a little luck guarantee a great game for all. $9.95.

---

**SPECIAL**  
*☆☆☆☆☆☆ MUSICAL MADNESS ☆☆☆☆☆☆*  
add an exciting new dimension to your PET computer with Soundware's soundsational music box and sonicsound software from Soflside & Soundware

**☆THE SOUNDFORKS☆**  
The Soundware music box for your PET comes complete with controllable volume, an earphone jack, a *demo tape* with two programs, an instruction book, and a *one year warranty*. This sturdy unit is enclosed in an attractive plastic case. Notes tell how to program your own sound effects. All this during our musical madness for just ........................................ 29.95

**WORD FUN:** Speller: fun ways to practice spelling + Scramble + Flashcards 9.95

**☆MUSICAL SOFTWARE☆**  
*ACTION PACK:* Breakthru + Target + Caterpillar: non stop graphic action 9.95  
*PINBALL:* a video action spectacle with real time flippers, chutes gates, bumpers, tags etc. ........................................ 9.95  
*CLASS CS:* Checkers + Backgammon Board + Piano Player: checkers vs. computer or friend. Piano plays Minute Waltz 9.95  
*MUSIC MANIA:* Try to repeat a growing sequence of tones. With graphics. Challenge to the best ear ............................ 9.95
You love your PET, but you'll love it more with this BigKeyboard?

The Skyles Big KeyBoard™. More than 15 inches wide. A layout nearly identical to the PET Keyboard and with all functions—alpha, numeric, graphics, special symbols, lower case alpha—on full-sized, almost plump, key-tops double-shot to guarantee lifetime durability.

Would you like to turn on your PET . . . and see this

- **COMMODORE BASIC**
- **31743 BYTES FREE**
- **READY**

74KB Big KeyBoards @ $125.00 (Plus $5.00 shipping & handling)

Skyles Electric Works
10301 Stonydale Drive
Cupertino, CA 95014
(408) 735-7891