QUALITY DISK SOFTWARE

BACKED BY ON-GOING APPLICATIONS SUPPORT

APPLE II Ⓐ *TRS•80* Ⓣ

HOME FINANCE PAK I: Entire Series $49.95 Ⓐ Ⓣ

CHECK REGISTER AND BUDGET: This comprehensive **CHECKING ACCOUNT MANAGEMENT SYSTEM** not only keeps complete records, it also gives you the analysis and control tools you need to actively manage your account. The system provides routines for **BUDGETING INCOME AND EXPENSE, AUTOMATIC CHECK SEARCH,** and **BANK STATEMENT RECONCILING. CRT** or printer reports are produced for **ACTUAL EXPENSE** vs **BUDGET, CHECK SEARCH DISPLAY RECONCILIATION REPORT** and **CHECK REGISTER DISPLAY** by month. Check entry is prompted by user-defined menus of standard purposes and recipient codes, speeding data entry and reducing disk storage and retrieval time. Six fields of data are stored for each check: amount, check no., date, purpose, recipient and **TAX DEDUCTIBLE REMINDER. CHECK SEARCH** routines allow searching on any of these data fields. Up to 100 checks/mo. storage $39.95

SAVINGS: Account management system for up to 20 separate Savings accounts. Organizes, files and displays deposits, withdrawals and interest earned for each account. $14.95

CREDIT CARD: Get Control of your credit cards with this program. Organizes, stores and displays purchases, payments and service charges for up to 20 separate cards or bank loans. $14.95

UNIVERSAL COMPUTING MACHINE: $49.95 Ⓐ

A user programmable computing system structured around a 50 row x 50 column table. User defines row and column names and equations forming a unique computing machine. Table elements can be multiplied, divided, subtracted or added to any other element. Hundreds of unique computing machines can be defined, used, stored, and recalled, for later use. Excellent for sales forecasts, budgets, inventory lists, income statements, production planning, project cost estimates-in short for any planning, analysis or reporting problem that can by solved with a table.

COLOR CALENDAR: $29.95 Ⓐ

Got a busy calendar? Organize it with Color Calendar. Whether it's birthdays, appointments, business meetings or a regular office schedule, this program is the perfect way to schedule your activities.
The calendar display is a beautiful **HI-RES** color graphics calendar of the selected month with each scheduled day highlighted in color. Using the daily schedule, you can review any day of the month and schedule an event or activity in any one of 20 time slots from 8:00 A.M. to 5:30 P.M.

BUSINESS SOFTWARE: Entire Series $159.95 Ⓐ Ⓣ

MICROACCOUNTANT: The ideal accounting system for small businesses. Based on classic T-accounts and double-entry booking, this efficient program provides a journal for recording posting and reviewing up to 1,000 transactions per month to any one of 300 accounts. The program produces **CRT** and printer reports covering:

TRANSACTION JOURNAL	BALANCE SHEET
ACCOUNT LEDGERS	INCOME AND EXPENSE STATEMENT

Includes a short primer on Financial Accounting. (48K) $49.95

UNIVERSAL BUSINESS MACHINE: This program is designed to **SIMPLIFY** and **SAVE TIME** for the serious businessman who must periodically Analyze, Plan and Estimate. The program was created using our Universal Computing Machine and it is programmed to provide the following planning and forecasting tools.

CASH FLOW ANALYSIS	SALES FORECASTER
PROFORMA PROFIT & LOSS	SOURCE AND USE OF FUNDS
PROFORMA BALANCE SHEET	JOB COST ESTIMATOR
REAL ESTATE INVESTMENT	INVENTORY ANALYSIS

Price, including a copy of the Universal Computing Machine $89.95

BUSINESS CHECK REGISTER AND BUDGET: Our Check Register and Budget programs expanded to include up to 50 budgetable items and up to 400 checks per month. Includes bank statement reconciling and automatic check search (48K) $49.95

ELECTRONICS SERIES VOL I & II: Entire Series $259.95

LOGIC SIMULATOR: SAVE TIME AND MONEY. Simulate your digital logic circuits before you build them. **CMOS, TTL,** or whatever, if it's digital logic, this program can handle it. The program is an interactive, menu driven, full-fledged logic simulator capable of simulating the bit-time response of a logic network to user-specified input patterns. It will handle up to 1000 gates, including **NANDS, NORS, INVERTERS, FLIP-FLOPS, SHIFT REGISTERS, COUNTERS** and user-defined **MACROS.** up to 40 user-defined random, or binary input patterns. Accepts network descriptions from keyboard or from **LOGIC DESIGNER** for simulation .... $159.95 Ⓐ Ⓣ

LOGIC DESIGNER: Interactive **HI-RES** graphics program for designing digital logic systems. Draw directly on the screen up to 10 different gate types, including **NAND, NOR, INVERTER, EX-OR, T-FLOP, JK-FLOP, D-FLOP, RS-FLOP, 4 BIT COUNTER** and **N-BIT SHIFT REGISTER.** User interconnects gates using line graphics commands. Network descriptions for **LOGIC SIMULATOR** generated simultaneously with the **CRT** diagram being drawn .... $159.95 Ⓐ

MANUAL AND DEMO DISK: Instruction Manual and demo disk illustrating capabilities of both program (s) .... $29.95 Ⓐ Ⓣ

ELECTRONIC SERIES VOL III & IV: Entire Series $259.95

CIRCUIT SIMULATOR: Tired of trial & error circuit design? Simulate & debug your designs before you build them! With **CIRCUIT SIMULATOR** you build a model of your circuit using **RESISTORS, CAPACITORS, INDUCTORS, TRANSISTORS, DIODES, VOLTAGE** and **CURRENT SOURCES** and simulate the waveform response to inputs such as **PULSES, SINUSOIDS, SAWTOOTHS,** etc. . .all fully programmable. The output is displayed as an **OSCILLOSCOPE-STYLE PLOT** of the selected waveforms (Apple only) or as a printed table of voltage vs time. Handles up to 200 notes and up to 20 sources. Requires 48 **RAM** .... $159.95 Ⓐ Ⓣ

CIRCUIT DESIGNER: Interactive **HI-RES** graphics program for designing electronic circuits. Draw directly on the screen up to 10 different component types, including those referenced above. Components interconnect list for **CIRCUIT SIMULATOR** generated automatically. Requires $159.95

MATHEMATICS SERIES: Entire Series $49.95

STATISTICAL ANALYSIS I: This menu driven program performs **LINEAR REGRESSION** analysis, determines the mean, standard deviation and plots the frequency distribution of user-supplied data sets. Printer, Disk, I/O routines $19.95

NUMERICAL ANALYSIS: HI-RES 2-Dimensional plot of any function. Automatic scaling. At your option, the program will plot the function, plot the **INTEGRAL,** plot the **DERIVATIVE,** determine the **ROOTS, MAXIMA, MINIMA, INTEGRAL VALUE** $19.95

MATRIX: A general purpose, menu driven program for determining the **INVERSE** and **DETERMINANT** of any matrix, as well as the **SOLUTION** to any set of **SIMULTANEOUS LINEAR EQUATIONS.** $19.95

3-D SURFACE PLOTTER: Explore the **ELEGANCE** and **BEAUTY** of **MATHEMATICS** by creating **HI-RES PLOTS** of 3-dimensional surfaces from any 3-variable equation. Disk save and recall routines for plots. Menu driven to vary surface parameters. Hidden line or transparent plotting $19.95

ACTION ADVENTURE GAMES: Entire Series $29.95 Ⓐ

RED BARON: Can you outfly the **RED BARON?** This fast action game simulates a machine-gun **DOGFIGHT** between your **WORLD WAR I BI-PLANE** and the baron's. You can **LOOP, DIVE, BANK** or **CLIMB**-and so can the **BARON.** In **HI-RES** graphics plus sound. $14.95

BATTLE OF MIDWAY: You are in command of the **U.S.S. HORNETS' DIVE-BOMBER** squadron. Your targets are the Aircraft carriers, Akagi, Soryu and Kaga. You must fly your way through **ZEROS** and **AA FIRE** to make your **DIVE-BOMB** run. In **HI-RES** graphics plus sound. $14.95

SUB ATTACK: It's April 1943. The enemy convoy is headed for the **CONTROL SEA.** Your sub, the **MORAY,** has just sighted the **CARRIERS** and **BATTLESHIPS**' Easy pickings. But watch out for the **DESTROYERS** - they're fast and deadly. In **HI-RES** graphics plus sound $14.95

FREE CATALOG-All programs are supplied on disk and run on Apple II w/Disk & Applesoft **ROM** Card & **TRS-80** Level II and require 32K **RAM** unless otherwise noted. Detailed instructions included. Orders shipped within 5 days. Card users include card number. Add $1.50 postage and handling with each order. California residents add 6½% sales tax. Foreign orders add $5.00 postage and handling.

SPECTRUM SOFTWARE
142 Carlow, P.O. Box 2084
Sunnyvale, CA 94087

FOR PHONE ORDERS: (408) 738-4387
DEALER INQUIRIES INVITED.

VISA master charge

```
>CALL-151

*CB

00CB- 90
*CA

00CA- C3
*90C3

90C3- 08
*
 00 00 62 B3
*

90C8- 00 0C 01 1C 0A 00 64 36
*90C3:08 00 00 11

*
```

your computer) and pressing (return) two or three times you will see a series of numbers representing line zero of your Basic program.

The first eight two-digit numbers you should see are the following: 08 00 00 62 B3 00 0C 01. The number "08" means there are eight numbers for line zero; the numbers "00 00" mean the first line is line zero; the "62" means PRINT; "B3 00 0C" stands for 3072 and the "01" at the end means "end of this line." By replacing the number 62 with the number 11 you can change the PRINT in line zero to LOMEN:.

To change the 62 to 11, just type in the address you found in CB and CA again (90C3 in fig. 3, but different in your computer), and put a colon immediately after it – but *don't* press return just yet! The colon tells the computer that you are going to enter some new numbers in here. Type the following four numbers: 08 00 00 11, and press return. Type control-C to get back into Basic, and list line zero. It will now read: 0 LOMEM: 3072.

Copy And Flip Subroutine

The workhorse of this program is the subroutine at line 60. It does the job of copying what is on graphics page one to page two. The display on page one is not erased by the copying, and will stay there until your program changes it. Here's how the page-flipping routine works: While your subroutines are drawing or erasing on page one, the viewer is looking at a finished picture on page two. When the drawing on page one is complete, the viewer is switched to page one where he sees the next step in the movement of the figure. He hasn't seen any of the drawing or erasing, so it only appears that the figure has made a slight shift in position. While the viewer is looking at page one, the computer is busy copying page one to page two – invisibly. The subroutine then switches to page two, but the viewer is unaware that anything has happened since page two is now an exact copy of page one. Now, while the viewer is looking at page two, the cycle repeats and a new drawing is begun "behind the scenes" on page one.

The POKE-16300,0 at the beginning of subroutine 60 causes page one to be displayed, showing the viewer the drawing you made while he was looking at page two. The next six pokes specify that it is page one of graphics that is to be copied to page two. The CALL-468 does the actual copying. The last Poke, POKE-16299,0 causes page two to be displayed again, and the subroutine returns to the main program so that the next step in the drawing can be made out of sight on page one.

The speed of the animation can be slowed down if necessary with delay loops such as those in lines 50, 51, and 52. For large or very complicated drawings delay loops won't be necessary, but for a small object which is quickly drawn you may want them.

Main Program

The main program is found from line 1000 to 2000. You can see that it consists mostly of GOSUB's. There is a FOR...NEXT loop, and several X=X+1 commands, which cause the figure to move to the right. Line 1000 clears the graphics and text display and calls the page-flip routine so that the viewer will be looking at a blank screen for a few microseconds while the first drawing is made. Lines 1010 and 1020 draw a sidewalk and place the girl on the sidewalk at the left of the screen, standing still (drawing A). Note that X was set to zero to place her at the left edge, and Y was set to 10 to put her down on the sidewalk. At the end of line 1020 the page flip subroutine is called, followed by a delay subroutine, so that the viewer looks at the standing girl for a brief time before she begins to walk. Lines 1030 through 1050 contain a FOR...NEXT loop during which the drawings B, C, D, A will be made seven times. (The loop starts at B since A was made before entering the loop, and it was desirable to finish with the girl in standing position). Note that the first GOSUB in line 1030 erases the girl. X is then incremented by one before the next drawing is made, so that she will appear to have shifted one space to the right. Each time a new drawing is made it must be erased before incrementing X and making the next drawing, or the figure will leave a trail of itself behind on the screen. In this example program the girl will move across the screen from left to right one square at a time. The last step of the program is to do a POKE-16300,0 at the end, so that the viewer will be left on page one when the program ends.

Final Hints

When you are debugging your program and using the page-flipping subroutine, you may occasionally hear the ominous "syntax error" beep, but be unable to see a message on the screen, and also see no cursor. This probably means you have been caught on page two, and the error message that stopped your program and left you stranded on page two is being displayed on page one. Just type POKE-16300,0 (return), and you will see page one displayed, where the error message and cursor will be visible.

It is hoped that you have a lot of fun with animating your graphics routines, despite the extra effort involved. The same general methods apply to Applesoft programs, but different "tricks" are required to gain access to page two and to do the memory move required. These "tricks" will be described in a future article.

```
>LIST
  0 LOMEM:3072
 10 POKE -16300,0: TEXT : CALL
    -936: GR : GOTO 1000
 49 REM  ** DELAY ROUTINES **
 50 FOR J=1 TO 50: NEXT J: RETURN

 51 FOR J=1 TO 1000: NEXT J: RETURN

 52 FOR J=1 TO 2000: NEXT J: RETURN

 59 REM  ** PAGE FLIPPER **
 60 POKE -16300,0: POKE 60,0: POKE
    61,4: POKE 62,255: POKE 63,
    7: POKE 66,0: POKE 67,8: CALL
    -468: POKE -16299,0: RETURN

 98 REM  ** GRAPHICS FOR GIRL **
 99 REM  ** DRAW TOP HALF **
100 COLOR=8: HLIN X+3,X+4 AT Y:
     VLIN Y,Y+2 AT X+3: PLOT X+
    2,Y+2
105 COLOR=13: VLIN Y+1,Y+2 AT X+
    4: COLOR=6: VLIN Y+3,Y+6 AT
    X+3: VLIN Y+3,Y+6 AT X+4
110 COLOR=3: VLIN Y+7,Y+8 AT X+
    3: VLIN Y+7,Y+8 AT X+4: HLIN
    X+2,X+5 AT Y+9: RETURN
119 REM  ** STANDING LEGS **
120 COLOR=13: VLIN Y+10,Y+11 AT
    X+3: COLOR=8: HLIN X+3,X+4 AT
    Y+12: RETURN
129 REM  ** STEP FORWARD **
130 COLOR=13: VLIN Y+10,Y+11 AT
    X+3: PLOT X+4,Y+10: PLOT X+
    5,Y+11: COLOR=8: HLIN X+3,X+
    4 AT Y+12: PLOT X+6,Y+12: PLOT
    X+7,Y+11: RETURN
139 REM  ** MIDDLE OF STEP **
140 COLOR=13: HLIN X+3,X+4 AT Y+
    10: PLOT X+2,Y+11: PLOT X+5
    ,Y+11: COLOR=8: HLIN X+2,X+
    3 AT Y+12: HLIN X+5,X+6 AT
    Y+12: RETURN
149 REM  ** END OF STEP **
150 COLOR=13: VLIN Y+10,Y+11 AT
    X+4: PLOT X+3,Y+10: PLOT X+
```

```
     2,Y+11: COLOR=8: VLIN Y+11,
     Y+12 AT X+1: HLIN X+4,X+5 AT
     Y+12: RETURN
 159 REM  ** ERASE GIRL **
 160 COLOR=0: FOR J=X+1 TO X+7: VLIN
     Y,Y+12 AT J: NEXT J: RETURN
 999 REM  ** MAIN ROUTINE **
1000 GR : CALL -936: GOSUB 60: TAB
     14: PRINT "GIRL WALKING"
1010 COLOR=5: HLIN 0,39 AT 23
1020 X=0:Y=10: GOSUB 100: GOSUB
     120: GOSUB 60: GOSUB 51
1030 FOR I=1 TO 7: GOSUB 160:X=X+
     1: GOSUB 100: GOSUB 130: GOSUB
     60: GOSUB 50
1040 GOSUB 160:X=X+1: GOSUB 100:
      GOSUB 140: GOSUB 60: GOSUB
     50: GOSUB 160:X=X+1: GOSUB
     100: GOSUB 150: GOSUB 60: GOSUB
     50
1050 GOSUB 160:X=X+1: GOSUB 100:
      GOSUB 120: GOSUB 60: GOSUB
     50: NEXT I
1060 CALL -936: TAB 17: PRINT "THE EN
     D": POKE -16300,0: GOSUB 52
     : GOSUB 160: COLOR=0: HLIN
     0,39 AT 23
2000 TEXT : CALL -936: POKE -16300
     ,0: END
```

©

Oscilloscope

Rob Smythe
Ontario, Canada

Here is a program for physics teachers that makes good use of the Apple's high resolution graphics.

Unless your school's equipment is better than mine, you probably find it tricky to demonstrate waveforms in class. Stabilizing the pattern on an oscilloscope can require painstaking adjustment when the frequency of the inputted sound is changed. You wish to show how the shape of the wave is altered as overtones are added, but textbooks don't show enough. Demonstrating the effect of different separations in the frequencies of two notes requires diagrams, tedious to produce.

With this program you can show effects of varying amplitude and frequency on sine waves, add up to five overtones (each with their own amplitude) and show the resultant wave pattern for up to six different notes. This last facility is useful for demonstrating the cause of beats.

When you run this Applesoft program you will be presented with a table showing that there are no notes presently in memory and a menu prompting you for single Keystroke selection of commands. Touch 1, 2, 3, 4, 5 or 6 and you will be able to set the amplitude and frequency of a note. Enter as many notes as you wish, or change them one by one. Touch P to plot the resultant waveform. After the oscilloscope pattern is drawn and you have finished studying it, return to the menu by pressing any key.

Touching S will enable you to alter the plotting speed, which is initially set at 4. This determines the increment along the x-axis (time axis) between plotted points. When using frequencies over about 500 Hz, you might have to set speed at 1 or 2 (because at coarser settings significant change to the shape might occur between points and be missed: try 800 Hz at speed 4 and speed 1 to see this).

To clear all notes from the table, touch C and confirm with a Y.

Try notes of amplitude 10 to 20 in a frequency range of 100 to 500. Create a complicated note using all overtones, with amplitudes 10 or less (so that you don't go off the top of the "scope"). Beat patterns look nice when you play notes of frequency 1000 and 1050 together.

The Program:

1000's	**print table and menu routine**
1030	**format numbers in display**
1100	**wait for single keystroke input**
1110	**input data**
1120 on	**process data and reject invalid input**
2000's	**plot routine**
2000-2110	**draw axes**
2150-2160	**pick X value in radians**
2170	**sum the waves**
2190	**scale X and Y to fit screen**
2200	**check for off scale values**
2210	**plot**
3000's	**subroutine to check that points are not off scale**

Variables (in order of appearance)

G$	**bell**
SP	**speed (1 = slow to 5 = fast)**
I	**counter, local pointer**
A$	**local input variable**
AMP(I)	**amplitude of the I-th note**
FR(I)	**frequency of the I-th note careful: don't use FRE(I)**
F(I)	**frequency after scaling for plotting**
TIME	**a measure of length of X-axis**
S	**scaling factor**
J	**loop counter**
X	**horizontal coordinate of point**
Y	**vertical coordinate of point**

Suggested Modifications:

1. Very small changes are required to allow for more overtones.

2. Changing TIME in line 2120 will allow for a different range of suitable frequencies. You might add TIME input to the menu, so that beats can be shown effectively with frequencies that are very close together.

3. Of course, adding routines which would produce the sound of the note you have created on the Apple's speaker would make this program tremendously useful. Any volunteers?

```
]LIST

10  REM   OSCILLOSCOPE

          R.M. SMYTHE
          1522 RUSHOLME CRES.
          BURLINGTON, ONTARIO
          CANADA
```

```
 20  REM

        COPYRIGHT (C) 1981
        BY SOFTWARE UNLIMITED

 50 G$ =  CHR$ (7): REM ERROR BEEP

 100 SP = 4: REM  PLOTTING SPEED
      FROM 1 (SLOW=MOST ACCURATE)
      TO 5
997 :
998  REM  DATA INPUT
999 :
1000  TEXT : HOME
1010  PRINT "    NOTE    AMP    F
     REQ": PRINT
1020  FOR I = 1 TO 6: PRINT  TAB(
     7);I;"      ";
1030 A$ =  RIGHT$ ("       " +  STR$
     (FR(I)),6): IF AMP(I) < 10 THEN
      PRINT " ";
1040  PRINT AMP(I);"  ";A$
1050  PRINT : NEXT I
1060  PRINT : PRINT : PRINT "SPEE
     D - ";SP
1070  VTAB 21
1080  PRINT "CHANGE NOTE: 1/2/3/4
     /5/6    PLOT: P"
1090  PRINT "CLEAR NOTES: C   EXI
     T: E   SPEED: S"
1100  POKE  - 16368,0: WAIT  - 16
     384,128
1110  GET A$:I =  VAL (A$): IF I >
     6 THEN  PRINT G$: GOTO 1000
1120  IF I = 0 THEN 1180
1130  VTAB 21: CALL  - 958: PRINT
     "NOTE ";I;": ";: INPUT "AMPL
     ITUDE (1-10) ";A$:AMP(I) =  VAL
     (A$): IF AMP(I) = 0 THEN 113
     0
1140  IF AMP(I) > 20 THEN  PRINT
     G$;: GOTO 1130
1150  PRINT  TAB( 9);: INPUT "FRE
     QUENCY - ";FR(I): IF FR(I) <
     0 OR FR(I) > 99999 THEN  PRINT
     G$;: VTAB 22: CALL  - 868: GOTO
     1150
1160 F(I) = FR(I) / 27.75
1170  GOTO 1000
1180  IF A$ = "E" THEN  END
1190  IF A$ = "P" THEN 2000
1200  IF A$ = "C" THEN 1240
1210  IF A$ <  > "S" THEN  PRINT
     G$: GOTO 1000
1220  VTAB 21: CALL  - 958: INPUT
     "ENTER SPEED (1-5) - ";SP: IF
     SP < 1 OR SP > 5 OR  INT (SP
     ) <  > SP THEN  PRINT G$: GOTO
     1220
1230  GOTO 1000
1240  VTAB 21: CALL  - 958: PRINT
     "CLEAR ALL NOTES IN MEMORY?
     (Y/N) ": GET A$: IF A$ <  >
     "Y" THEN 1000
1250  FOR I = 1 TO 6:F(I) = 0:FR(
     I) = 0:AMP(I) = 0: NEXT : GOTO
     1000
1997 :
1998  REM  PLOTTING ROUTINE
1999 :
2000  HOME
2010  VTAB 24
2020  HGR
2030  HCOLOR= 3
2040  HPLOT 0,80 TO 279,80
2050  HPLOT 0,16 TO 0,143
2060  FOR I = 0 TO 279 STEP 70
2070  HPLOT I,78 TO I,82: HPLOT 2
     79,78 TO 279,82
2080  NEXT I
2090  FOR I = 16 TO 144 STEP 16
2100  HPLOT 0,I TO 4,I
2110  NEXT I
2120 TIME = 400
2130 S = 280 / TIME
2140  HPLOT 0,80
2150  FOR I = 0 TO TIME STEP SP
2160 X = I * 3.14159 / 180
2170 Y = 0: FOR J = 1 TO 6:Y = AM
     P(J) / 5 *  SIN (F(J) * X) +
     Y: NEXT J
2180 Y = 80 - Y * 16
2190 X = I * S
2200  GOSUB 3000
2210  HPLOT  TO X,Y
2220  NEXT I
2230  POKE  - 16368,0: WAIT  - 16
     384,128
2240  GET A$
2250  GOTO 1000
2997 :
2998  REM  SUBROUTINE CHECK RANGE

2999 :
3000  IF X < 0 THEN X = 0
3010  IF X > 279 THEN X = 279
3020  IF Y < 0 THEN Y = 0
3030  IF Y > 159 THEN Y = 159
3040  RETURN
```

©

Apple Authors

COMPUTE! is looking for applications articles aimed at beginners and intermediate programmers. We're specifically interested in programming hints, tutorials, articles written to help users get more out of their machine.

The Apple Hi-Res Shape Writer

Doug Hennig
Dallas, TX

These days a lot of people are writing their own games or applications software rather than paying for someone else's labor. After all, designing and writing software really is the best part about owning your own microcomputer. Games especially are fun to design because they tax not only your programming skills, but your imagination as well.

Many of today's popular computer games are of the "arcade" type where you are required to fire missles to destroy any number of different kinds of objects. Usually these objects are drawn and manipulated on the high resolution screen using hi-res shape tables that are stored in memory to define the object. Other types of games also use hi-res shape tables, including a number of recently developed high resolution adventure games. As you have no doubt already observed, the uses of shape tables in graphics programs is almost limitless. However, that is where the problem arises.

Anyone who has ever designed a high resolution shape table knows that it is not a lot of fun to do. You must draw the shape dot by dot, convert each dot into a number, convert the numbers into bytes by an obscure process, type in the long list of bytes, and hope that you have not made an error somewhere along the way (I invariably do!). After several attempts at such nonsense, I thought that there must be an easier way. Why not do just the fun part – drawing the shape – and leave the Apple to do the hard part? Thus the Apple Hi-Res Shape Writer was born.

The Apple Hi-Res Shape Writer allows you to draw any shape on the low resolution screen using the game paddles and then convert the lo-res shape into a high resolution shape table. You will soon see your graphics ideas take shape (no pun intended) quickly and painlessly. Create space ships, alien creatures, even new and exotic character sets.

Operation

Operation of the Apple Hi-Res Shape Writer is simple. First you tell the program how many shapes you want in the shape table (the maximum is ten). It will then draw a dark blue background in low resolution graphics and ask you for the number of vertical and horizontal elements in the shape that is to be drawn next; since the low resolution screen is 40x40, these are the limits for the shape. A black area that is the size of your shape will appear, which can be filled in as you wish. The location of the current plotting position, indicated by a flashing light blue "cursor," is controlled by the game paddles. To plot a point, press "P" and to erase a previously plotted point, press "E" (note that the REPT key can be used along with either of these to give "speed" drawing). The shape does not have to be drawn in any particular order; you can "doodle" if you want, trying different designs and erasing the parts that you do not want. Once the shape is drawn to your satisfaction, press "S" to construct the shape table. After the table is done, you may see the high resolution shape before starting on the next shape. Once you have drawn the desired number of shapes, you have the option of saving the entire shape table on disk.

Hi-res shapes will appear "skinnier" than the lo-res shapes drawn because low resolution blocks are rectangular rather than square. However, a little practice will allow you to easily visualize how your hi-res shape will look and to plan the lo-res shape accordingly.

Since the hi-res shape table is stored in the second page high resolution buffer (starting at 16384 or $4000), The Apple must have at least 24K and Applesoft in ROM.

The Program

There are few "fancy tricks" used in this program. If you are not familiar with the way shape tables are created or used, you should read Chapter Nine of the latest Applesoft manual before trying to follow how this program works. To help explain its operation, I have included a list of major variables and subroutines used in the program, with comments about the uses of each.

Variables Used (In Order of Appearance)

A$ Used for all input from the user.

SH The number of shapes to be drawn (a maximum of ten). This number is POKEd into the first two locations of the shape table.

BASE The starting address of the data for the current shape. It is initially set to 16384 + 2*SH + 2 because the table starts at 16384 and the table index consists of two bytes for the number of shapes and two bytes to point to each shape.

NU The number of the current shape.

TABLE () Holds the plot status of every point in the shape area of the screen. The status is based on the following system:

move right (no plot) = 1	plot and move right = 5
move down (no plot) = 2	plot and move down = 6
move left (no plot) = 3	plot and move left = 7

BLACK, AQUA, RED Hold the values of the corresponding low resolution colors.

COL A dual purpose variable: first it holds the number of horizontal elements in the shape and later it is used as a loop counter for the columns.

C1, C2 Contain the horizontal low resolution screen limits.

R1, R2 Contain the vertical low resolution screen limits.

X, Y Represent the coordinates of the current plot position.

OLD Holds the color of the screen under the "cursor" (black if nothing was plotted, red if a point was plotted).

MOVE Stores the plot value: either move right (1) or move left (3), depending on which row is being scanned (see the variables START, LAST).

DOWN Stores the plot value to move down (2).

START, LAST Hold the loop limits for the column counter. The first row is scanned left to right, the second right to left, and so on (always moving down one row when the end of the current row is reached), so START and LAST are switched at the end of each row.

INC Another dual purpose variable: initially, it holds the step value for the column loop (1 means scanning left to right and -1 means scanning right to left); later it is used to hold the number of bytes in the current shape.

B Holds the octal representation of the current byte to be put into the shape table. Octal is used because in hi-res shape tables, eight possible moves are allowed (see the Applesoft manual).

W, X, Y, Z Store intermediate results in the conversion of octal to decimal. X also holds the final (decimal) result.

D Holds the length of the shape table.

Sections And Subroutines Used

10–37 Introduction.
40–400 Print instructions.
410–450 Initial shape table setup.
460–500 Variable initialization.
510–700 Low resolution screen setup.
710–830 Draw the lo-res shape.
840–1130 Create the hi-res shape table.
1140–1240 Display the hi-res shape.
1250–1500 Save the shape table on disk.
11000–11080 Input user response (avoids the problems of using INPUT).
12000–12100 Print heading.
13000–13040 Go on to next screenful.

Changes And Modifications

A number of changes to the program are possible. The colors of the background, the flashing "cursor," and plotted points may be changed to suit your taste by changing the values in lines 490 and 530 (AQUA is the color of the "cursor" and RED is the color of a plotted point). To allow more shapes per table (up to a reasonable limit, of course), change the upper limit in line 425.

Perhaps the most important modification would concern the speed of execution. Creation of the shape table can take up to several minutes for large shapes because of the amount of data involved. If the time involved seems unreasonable to you, perhaps a machine language version of lines 840 to 1130 would be in order. Since I am not proficient in machine language, I chose to let Basic do the job for me, but I am sure that some enterprising soul can come up with a faster version. If you want to tackle this problem, feel free to contact me with any questions that you have about the program.

Using Shape Tables With Your Own Programs

There are two ways that you can use a shape table in your own program. The program can read in the table from disk (using BLOAD) or the program can POKE in the values for the table from data in DATA statements. The first method is obviously easier to program, but you must ensure that the disk containing the shape table is inserted in the drive or the user will get a nasty DOS error message.

The question with the second method is: how do I convert all those bytes in memory into numbers in DATA statements? One way (the hard way) is to write a short program which reads the shape table one byte at a time, prints the value, and lets you write it down before going on to the next one. Then you have to type all the values into DATA statements and POKE them into memory. The easier way is to EXEC a text file which will do all that for you. Listing 2 contains a program which will set up such a text file. To use this, set the variable LINE in line 90 to the line number that you want the POKE routine to start at, set the variable B in line 90 to the last memory location of the shape table (lines 1460 and 1470 in the Apple Hi-Res Shape Writer print this value for you), RUN the program and EXEC the text file. For example, for a shape table that the computer tells you ends at 17000, type (in the direct mode):

```
LOAD POKE WRITER
90 LINE = 5000 : B = 17000
RUN
```

Now just load your program, EXEC POKE ROUTINE, and you have added a routine, starting at line 5000 in this example, that will POKE your shape table into memory every time the program is run.

I hope that you enjoy Apple Hi-Res Shape Writer. I have found it extremely useful in designing some of my own games and educational software, and I am sure that you will find many uses for it too!

```
?SYNTAX ERROR
]LIST

10  REM ********************************
12  REM ** APPLE HI-RES SHAPE WRITER **
14  REM **       BY DOUG HENNIG       **
20  REM ********************************
25  HIMEM: 16384
30  GOSUB 12000
35  PRINT "DO YOU WANT INSTRUCTIONS? ";: GOSUB 11000
37  IF A$ = "N" THEN  HOME : GOTO 400
40  VTAB 10: CALL  - 868: HTAB 3
50  PRINT "THIS PROGRAM WILL ALLOW YOU TO CREATE"
60  PRINT "UP TO TEN DIFFERENT SHAPE TABLES FROM"
70  PRINT "DESIGNS THAT YOU HAVE DRAWN ON THE"
80  PRINT "SCREEN IN LOW-RESOLUTION GRAPHICS."
90  VTAB 15: HTAB 3
100  PRINT "THE PROGRAM WILL ASK FOR THE NUMBER"
110  PRINT "OF HORIZONTAL AND VERTICAL ELEMENTS "
120  PRINT "THAT YOU WANT. THESE NUMBERS MUST BE"
130  PRINT "BETWEEN 1 AND 40."
140  GOSUB 13000
150  HTAB 3
160  PRINT "A BLUE 'CURSOR' WILL BE DISPLAYED TO"
170  PRINT "INDICATE THE CURRENT PLOT POSITION. USE"
180  PRINT "THE GAME PADDLES TO MOVE THE CURSOR TO"
190  PRINT "THE DESIRED LOCATION AND PRESS ";: INVERSE : PRINT "P";: NORMAL
    : PRINT " TO"
200  PRINT "PLOT AT THE CURRENT POSITION; A PLOTTED"
210  PRINT "POINT IS INDICATED BY A RED SQUARE. TO"
220  PRINT "ERASE A PLOTTED POINT, PRESS ";: INVERSE : PRINT "E";: NORMAL
    : PRINT ". TO"
230  PRINT "BEGIN CREATION OF THE SHAPE TABLE,"
240  PRINT "PRESS ";: INVERSE : PRINT "S";: NORMAL : PRINT ". "
250  GOSUB 13000
260  HTAB 3
270  PRINT "THE SHAPE TABLES WILL BE STORED IN"
280  PRINT "THE HIGH-RESOLUTION SECONDARY PAGE AREA"
290  PRINT "(STARTING AT LOCATION 16384, OR $4000)."
300  PRINT "YOU WILL BE GIVEN THE OPTION TO SAVE "
310  PRINT "THE SHAPE TABLE ON DISK."
320  PRINT : HTAB 3
330  PRINT "TO USE THE SHAPE TABLE IN A PROGRAM,"
340  PRINT "SIMPLY USE A 'BSAVE' COMMAND WITHIN"
350  PRINT "THE PROGRAM. NOTE THAT THE HIGH-"
360  PRINT "RESOLUTION SECONDARY PAGE WILL BE"
370  PRINT "UNAVAILABLE FOR USE."
390  GOSUB 13000
400  POKE 34,0
410  PRINT "HOW MANY SHAPES WILL THERE BE? ";: GOSUB 11000
420 SH =  INT ( VAL (A$))
425  IF SH < 1 OR SH > 10 THEN  VTAB 10: CALL  - 958: GOTO 410
430  POKE 16384,SH: POKE 16385,0: REM PUT NUMBER OF SHAPES INTO START OF T
    ABLE INDEX
440 BASE = 16384 + 2 * SH + 2
450  POKE 232,0: POKE 233,64: REM TELL APPLE WHERE SHAPE TABLE IS
460 NU = 0
480  DIM TABLE(1600)
490 BLACK = 0:AQUA = 6:RED = 1
```

```
500 NU = NU + 1: IF NU > SH THEN 1250
510  GR
520  HOME : VTAB 21
530  COLOR= 2
540  FOR I = 0 TO 39: VLIN 0,39 AT I: NEXT I: REM DRAW BACKGROUND
550  PRINT "NUMBER OF HORIZONTAL ELEMENTS - ";: GOSUB 11000
560 COL =  INT ( VAL (A$))
570  IF COL < 1 OR COL > 40 THEN  VTAB 21: CALL  - 958: GOTO 550
580  HOME : VTAB 21
590  PRINT "NUMBER OF VERTICAL ELEMENTS - ";: GOSUB 11000
600 ROW =  INT ( VAL (A$))
610  IF ROW < 1 OR ROW > 40 THEN  VTAB 21: CALL  - 958: GOTO 590
620 C1 =  INT ((40 - COL) / 2):R1 =  INT ((40 - ROW) / 2)
630 X =  INT (COL / 2 -  INT (COL / 2) + 0.5):Y =  INT (ROW / 2 -  INT (RO
    W / 2) + 0.5)
640 C2 = 39 - C1 - X:R2 = 39 - R1 - Y
650  COLOR= BLACK: FOR I = C1 TO C2: VLIN R1,R2 AT I: NEXT I: REM CLEAR SP
     ACE FOR SHAPE
660  HOME : VTAB 21
670  INVERSE : PRINT "P": PRINT "E": PRINT "S": NORMAL
680  VTAB 21: HTAB 3: PRINT "- PLOT POINT"
690  VTAB 22: HTAB 3: PRINT "- ERASE POINT"
700  VTAB 23: HTAB 3: PRINT "- CREATE SHAPE TABLE"
710 X = C1:Y = R1
720 OLD =  SCRN( X,Y)
730  COLOR= AQUA: PLOT X,Y: REM FLASH CURSOR
740  FOR I = 1 TO 100: NEXT I
750  COLOR= OLD: PLOT X,Y
760 X =  INT ((C2 - C1 + 1) / 255 *  PDL (0)) + C1:Y =  INT ((R2 - R1 + 1)
     / 255 *  PDL (1)) + R1: REM GET NEW COORDINATES
770  IF X > C2 THEN X = C2: REM DON'T GO OUT OF BOUNDS
780  IF Y > R2 THEN Y = R2
790 KEY =  PEEK (49152):A =  PEEK (49168)
800  IF KEY < 128 THEN 720
810  IF KEY = 208 THEN  COLOR= RED: PLOT X,Y: REM PLOT POINT
820  IF KEY = 197 THEN  COLOR= BLACK: PLOT X,Y: REM ERASE POINT
830  IF KEY <  > 211 THEN 720
835  REM CREATE SHAPE TABLE
840  HOME : VTAB 21: PRINT "CREATING SHAPE TABLE"
850 MOVE = 1:N = 0:DOWN = 2
860 START = C1:LAST = C2:INC = 1
865  REM STARTING AT THE UPPER LEFT CORNER, SCAN BACK AND FORTH ACROSS ROW
     S
870  FOR ROW = R1 TO R2
880 : FOR COL = START TO LAST STEP INC
890 ::TABLE(N) = MOVE
900 :: IF COL = LAST THEN TABLE(N) = DOWN: REM IF END OF ROW GO DOWN
910 :: IF  SCRN( COL,ROW) = 1 THEN TABLE(N) = TABLE(N) + 4: REM A PLOT POI
     NT
920 ::N = N + 1
930 : NEXT COL
940 :TEMP = START:START = LAST:LAST = TEMP
950 MOVE = MOVE + INC * 2
960 :INC =  - INC
970  NEXT ROW
980 COL = 0:INC = 0
990 B = 0: FOR I = 0 TO 2: REM CONVERT MOVES TO BYTES
1000 :A =  INT (10 ^ I * TABLE(COL))
1010 :B = B + A
```

```
1020 : IF B > 199 THEN B = B - A: GOTO 1050
1030 :COL = COL + 1: IF COL = N THEN 1050
1040  NEXT I
1050 W =  INT (B / 100):X =  INT (B / 10)
1060 Y = X - W * 10:Z = B - X * 10
1070 X = 64 * W + 8 * Y + Z
1080  POKE BASE + INC,X:INC = INC + 1: REM PUT BYTE INTO TABLE
1090  IF COL <  > N THEN 990
1100  POKE BASE + INC,0:INC = INC + 1: REM END OF THIS SHAPE
1110  POKE 16384 + 2 * NU,BASE - 16384 - 256 *  INT ((BASE - 16384) / 256)
    : REM POKE POINTERS TO THIS SHAPE INTO INDEX
1120  POKE 16385 + 2 * NU, INT ((BASE - 16384) / 256)
1130 BASE = BASE + INC
1140  HOME : VTAB 21
1150  PRINT  CHR$ (7)"WANT TO SEE HI-RES SHAPE #"NU"? ";: GOSUB 11000
1160  IF A$ = "N" THEN 1250
1170  HGR : SCALE= 1: ROT= 0
1180  HCOLOR= 3
1190  DRAW NU AT 140,80
1200  HOME : GOSUB 13000
1240  GOTO 500: REM NEXT SHAPE
1250  REM   SAVE ON DISK
1260  GOSUB 12000
1270  VTAB 10: PRINT "DO YOU WANT TO SAVE THE SHAPE TABLE ON"
1280  PRINT "DISK? ";: GOSUB 11000
1290  IF A$ <  > "Y" THEN 1480
1300 D = BASE - 16384: REM  LENGTH OF TABLE
1340  PRINT : PRINT "FILE NAME - ";: GOSUB 11000
1350  IF  LEN (A$) > 30 OR  VAL ( LEFT$ (A$,1)) <  > 0 THEN  VTAB  PEEK (3
    7) - 1: CALL  - 958: GOTO 1340
1355  FOR I = 1 TO  LEN (A$): IF  MID$ (A$,I,1) = "," THEN  VTAB  PEEK (37
    ) - 1: CALL  - 958: GOTO 1340
1357  NEXT
1360  PRINT : PRINT "INSERT THE DATA DISK INTO THE DRIVE"
1380  PRINT "AND THEN PRESS ANY KEY."
1390 KEY =  PEEK (49152): IF KEY < 128 THEN 1390
1400 KEY =  PEEK (49168)
1410  PRINT : PRINT "SAVING SHAPE TABLE"
1415  ONERR  GOTO 5000
1420  PRINT  CHR$ (4)"BSAVE"A$",A$4000,L"D
1430  POKE 216,0
1460  PRINT : PRINT "THE LAST LOCATION IN THE SHAPE TABLE"
1470  PRINT "IS "BASE - 1"."
1480  POKE 34,0
1490  PRINT : PRINT "GOOD LUCK WITH YOUR NEW SHAPE TABLE!"
1600  END
4999  REM CONVERT DECIMAL TO HEX
5000  PRINT : PRINT "THERE WAS A DISK I/O ERROR.": POKE 216,0: END
10999  REM "INPUT" SIMULATOR
11000 A$ = ""
11010  GET B$
11020  IF B$ =  CHR$ (13) THEN  PRINT : RETURN : REM IF RETURN PRESSED GO
    BACK
11025  IF B$ =  CHR$ (21) OR B$ =  CHR$ (10) THEN 11010
11030  IF B$ <  >  CHR$ (8) THEN  PRINT B$;:A$ = A$ + B$: GOTO 11010
11040  IF  LEN (A$) = 0 THEN 11010: REM IF NO CHARS ENTERED IGNORE BACKSPA
    CE
11050  PRINT B$" "B$;
11060  IF  LEN (A$) = 1 THEN 11000
```

```
11070 A$ =  LEFT$ (A$, LEN (A$) - 1)
11080  GOTO 11010
11999  REM PRINTHEADING AND SET TEXT WINDOW
12000  TEXT : HOME
12010  VTAB 2: INVERSE
12020  FOR I = 1 TO 40: PRINT " ";: NEXT I: PRINT
12030  VTAB 4: HTAB 10: NORMAL
12040  PRINT "HI-RES SHAPE WRITER"
12050  HTAB 13: PRINT "BY DOUG HENNIG"
12060  VTAB 7: INVERSE
12070  FOR I = 1 TO 40: PRINT " ";: NEXT I: PRINT
12080  POKE 34,9
12090  NORMAL : VTAB 10
12100  RETURN
12999  REM NEXT SCREEN ROUTINE
13000  VTAB 23: HTAB 8
13010  INVERSE : PRINT "PRESS RETURN TO CONTINUE": NORMAL
13020 KEY =  PEEK (49152): IF KEY < 128 THEN 13020
13030 KEY =  PEEK (49168)
13040  HOME : RETURN

]
```

©

THE ATARI® GAZETTE

Adding A Voice Track To ATARI Programs II

Mike Doleman
Richfield, MN

Deja vu? Not really. Those of you interested in making your ATARI speak, a la a pre-recorded voice track in "sync" with your program may recall an article in the July/Aug. '80 issue of **COMPUTE!** which shows one method of doing this. Since what will be explained in this article results in the same effect, it is titled the same. Hopefully however, you will agree that this new method has advantages.

Some problems do exist with using the other method since, in that situation, the two elements — the voice track and the program — advance independently of each other and in some situations are vulnerable to falling out of sync. Obviously if the two are to be in sync some care must be taken to see that they start out together. Not so obviously, if you happen to put the cassette into a cassette player that runs at a slightly different speed than the one used to obtain the time values, presto — your program and voice track gradually get separated, after all that work you put into synchronizing them. (Not to mention the whole sync procedure being a cumbersome hassle even if it works right.)

Take heart, you can do it the way the "pros" do, and you don't need any "special hardware" to accomplish it with your ATARI system. You probably already have it, an ordinary run-of-the-mill stereo cassette tape recorder with two microphones.

With the stereo cassette recorder and the aid of your ATARI you can put a signal on the digital track of a cassette tape which controls the advancement of the program, and do it at the same time you're recording the voice track — thereby eliminating the need for doing all that other "stuff" when using the other method.

Here's the nuts and bolts, or should I say peeks and pokes of how it's done.

Memory location 53775 (refer to Appendix I, Basic Reference Manual) can be made to return several values by the ATARI program recorder as it plays. These values are governed, as I mentioned before, by a signal on the digital track of the cassette tape, so the first thing to know is which signal does what to 53775 and how to put it on the tape at the proper time. The signal itself is simply an audio frequency and can be easily generated by the SOUND statement on your ATARI. Specifically just two tones are needed, one caused by a SOUND 0,5,10,15 statement and the other by a SOUND 0,8,10,3 statement.

If the signal is the audio frequency produced by a SOUND 0,5,10,15 being recorded on the tape, then on playback in the program recorder it will cause 53775 to return a 255, if the signal is changed to a sound 0,8,10,3 then 53775 will return a 239.

Now! If, in the program which is being used with the voice track, a simple subroutine is placed wherever you want the program to stop and wait for a cue from the tape (the subroutine monitoring 53775 and holding if 53775 = 255 or continuing if 53775 = 239), then you have total control, by the tape, over the progression of the program.

Now we're ready for the step-by-step procedure to actually make a tape that does the job! Presumably you already know where, on the script you have written for your voice track, you want the program to stop running and wait for the voice track to cue it to begin running again. If you haven't done so already, mark these places.

STEP 1. When you set up to record your voice track (in the left channel of your recorder) also put a right channel microphone directly in front of your TV speaker. (The TV you use with your ATARI).

STEP 2. Load the following program into your ATARI:

```
10 SOUND 0,5,10,15
20 FOR X=1 TO 500 : NEXT X
30 IF STRIG(0)=1 THEN 30
40 SOUND 0,8,10,3
50 FOR X=1 TO 100 : NEXT X
60 GOTO 10
```

STEP 3. Plug a joystick into controller jack 1 of the ATARI. (Or a paddle and change the 'STRIG' in statement 30 to a 'PTRIG'. If you have neither then use:

```
30 IF PEEK(53279)<>7 THEN 30
```

and press any console switch rather than a trigger button.)

STEP 4. RUN the program and adjust the record level of the right channel so the signal is strong. Also adjust the left channel appropriately for recording the voice track.
STEP 5. Position the tape you are using for the voice track to the proper place for recording. (This will probably be immediately behind the program it goes with.)
STEP 6. Begin recording the voice track and every time you come to a place previously marked in the script for the program to continue simply press the trigger button. (Remember to release it fairly quickly as a safeguard to putting more than one signal change on the tape.)

The only thing you now need in the main program is a little subroutine that stops the program and then only allows it to continue on the cue (signal) you have just put on the tape along with your voice track. It is as follows:

```
0000 GOSUB 6000 (Placed wherever you want to stop
     the program.)
6000 FOR X=1 TO 500 : NEXT X
6010 X=PEEK(53775):IFX=255 OR X=127
     THEN 6010
6020 RETURN
```

Obviously the subroutine does not have to be numbered 6000; number it whatever you need for your own program. The delay at 6000 is needed in case the program execution to the next place you want stopped is so fast that it catches the tape still signaling for a change, which is not an improbability. And the number 127 needs to be included in 6010 because sometimes the ATARI decides to return it instead of 255.

And that is it! You have just eliminated any need to collect all those time values and go back to your program to put them in the proper place. Along with that, there is now no chance that the voice track and program will go out of 'sync'.

A word about an inconvenience which may manifest itself in the form of the TV speaker interfering with the voice track recording while making the signals for the digital track. It may be a good idea to make a direct hookup from the ATARI monitor jack to the right channel of the cassette recorder. Pin hole #3 (upper left) is the audio and a 1 mega ohm resistor should be used to prevent cross channel recording. Also note that the procedure for allowing the program to turn the program recorder on and off, a necessary function for making voice track/program combinations, is simply POKE 54018,52 for 'on' and 60 for 'off'. ©

Atari Tape Techniques

Richard M. Kruse
Wichita, Kansas

The Atari 400/800 personal computers have more built-in capabilities than most users realize. This article expands on one such area – the tape-handling functions. No special knowledge or equipment beyond the Atari computer and tape recorder is required, and the Atari user who is so equipped can try the new procedures immediately. Included is the listing for a BASIC program called "Hex Tape Dump," which allows display and examination of Atari tape records.

It is a safe assumption that nearly all owners of Atari 400/800 computers have also acquired the Atari model 410 cassette tape recorder. This peripheral device was originally shipped with each Atari 800 computer, but is now sold as an optional accessory useable with either model. The 410 is the basic mass storage device in the Atari system. It's simple, reliable, and economical (roughly one-eighth the cost of a single disk drive).

Yet, it's also a good bet that most Atari owners are not using their tape systems at anywhere near full potential. Now we Atari fans are, in general, no more lazy than the rest of the computerists out there, but, alas, we have been a little short on information concerning our machines. We recognize, though, that the Atari folks out in Sunnyvale have struggled mightily to correct this situation, and we applaud their efforts. The Atari Basic Reference Manual is, in fact, a model of organization that others would do well to emulate. Still, the Atari is a complex machine, and even this excellent manual barely scratches the surface of Atari's capabilities. This is the case with the more advanced cassette functions.

(An aside: In case you've heard about the new Atari tech manuals ... yes, they're real, and they're crammed with data. They are also VERY heavy reading. Unless you're on easy speaking terms with the 6502 and advanced machine-language programming techniques, stick with the material presented in these pages. The same high-class stuff will soon be showing up here; a little at a time and with enough explanation to make it useable to all.)

If you have had the feeling that maybe you were missing something along these lines, then read on. The information in this article will get you past "CSAVE" and "CLOAD", and show you how to get more 'bang per buck' out of your 410 recorder while sticking with BASIC language.

Atari fans ... to your keyboards!

THERE'S LIFE BEYOND CLOAD!

First, let's examine some tape functions whose commands are built-in; that is, no POKES or PEEKS are required. Take a few moments here to dust off your Atari Basic Reference Manual and reread the descriptions in the "Input/Output Commands" section for "ENTER", "INPUT", "LOAD", "OPEN/CLOSE", "PRINT", "PUT/GET", and "SAVE". Each of these commands is applicable to a tape function, and there are important differences between them which seem, on the surface, to be redundant.

THE LONG AND SHORT OF IT

All data written to tape by the Atari OS (operating System) is first formatted into 128-byte blocks called RECORDS. The OS then appends four additional bytes of control data to each record, forming 132-byte FRAMES. Since 128 bytes is a fairly small amount of data, several frames will normally be recorded sequentially to make a FILE. Adjacent frames in a file are never contiguous, however, but are separated by non-data spaces called INTER-RECORD GAPS (IRG's). Refer to Figure 1, which is a pictorial diagram that should help clarify the relationship between these entities.

We now have enough information to understand and appreciate the difference between CSAVE and SAVE"C". Both of these BASIC commands save a BASIC source program to the cassette, but CSAVE uses short IRG's (less than a

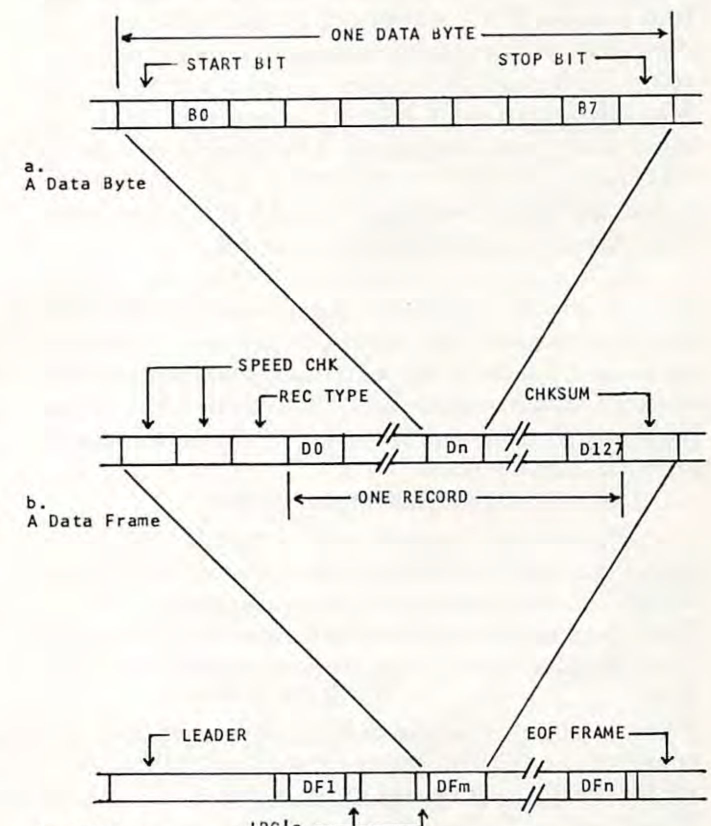


Figure 1: 128 data bytes (a) are combined with four control bytes to make a data frame (b). A file (c) consists of a leader, one or more data frames, and an End-Of-File frame.

second) while SAVE"C" uses long gaps (about three seconds).

What practical difference does this make? And why would I ever use SAVE"C" anyway, since it makes the snail-like pace of program loading even slower? Okay, here we go. The significance of long IRG's, and the reason for providing the option to use them, is this: The long gap provides just enough space between records to stop the tape drive motor, restart it when ready, and bring the tape back up to speed before the beginning of the next sequential frame of data. This cannot be accomplished with short gaps, and under some conditions the computer cannot accept data fast enough to accomodate non-stop operation. (In all fairness, there is another reason for the apparent redundancy of CSAVE and SAVE"C". While CSAVE applies specifically and only to the tape recorder, SAVE"C" is one form of a more general command that also applies to disk files.)

Now, I can almost hear you saying, "Well, that's nice to know, sort of, but the information doesn't really seem too useful." Right? You would be right, of course, except for a function called RUN"C" which, according to the Reference Manual, loads a BASIC program and immediately begins execution. The only problem is, it doesn't work with a CSAVE'd program ... Aha! You're way ahead of me. Yes, indeed, try SAVE"C" and you'll find that RUN"C" works exactly as advertised. Atari has, in fact, given you a hint on this without really explaining the reasons. Look at the Reference Manual section on "Chaining Programs." Makes more sense now, doesn't it? The specific rule is that CLOAD, LOAD"C" and RUN"C" will all read source programs written with SAVE"C", but only CLOAD will read CSAVE'd material.

By the way, I should point out that the correct form is actually SAVE"C:" with a colon following the device name. The Atari OS, however, allows only one cassette drive, and does not support file names (ouch!) in tape files. This being the case, no further information following "C" in a command string is actually used.

ONLY TOKENS WILL BE ACCEPTED...

There is still another way to save BASIC programs, much different from either CSAVE or SAVE"C", and serving a unique purpose. The Atari personal computers, like most of their cousins, store BASIC source programs internally in a kind of shorthand called the TOKENIZED form. A token is a one-or two-byte (typically) code that represents a BASIC keyword such as GOSUB or PRINT. Source programs are encoded into this shorthand form in order to conserve memory space in the computer.

The significance of tokenization to our discussion is that both CSAVE and SAVE"C" write the shortened, tokenized form of your BASIC program to the cassette. Not surprisingly, CLOAD, LOAD"C", and RUN"C" all recognize only this abbreviated code. There exists, however, a method of storing the original, expanded ASCII source code. And, just as with long IRG's, there are a couple of good reasons to use this method.

CAUTION: MERGING TRAFFIC

Every BASIC programmer is familiar with the LIST command, with which he can review previously-entered source code on the CRT or produce a permanent program listing on a printer. Atari's treatment of I/O devices, however, allows you to LIST your program to the cassette as well. The command format is LIST"C", to save all source lines; or LIST"C",X,Y to save only those from line X to line Y, inclusive. Either way, program lines will be stored in full ASCII form, including line numbers, REMarks, and (most) spaces. There has to be a way, of course, to retrieve such a listing, and this is done with the command ENTER"C".

The LIST"C"/ENTER"C" sequence has one particular characteristic that makes it indispensible in preparing BASIC programs. Unlike CLOAD and LOAD"C", which both clear any BASIC source lines from memory before loading, ENTER"C" does not necessarily disturb a resident program. This means that you can merge often-used routines into a BASIC program without having to retype them each time. Once you become familiar with this process, it can save lots of time and effort.

Actually, ENTERing a source program from tape is exactly equivalent to typing source lines at the keyboard, and the same rules generally apply. If a line is entered from tape, for example, having the same line number as a line already in memory, the old line will be replaced.

THE INCREDIBLE SHRINKING CODE

Another interesting and potentially useful effect of the LIST"C"/ENTER"C" sequence is that it can actually reduce the size of your BASIC programs. I stumbled across this undocumented characteristic while doing my "homework" for this article, and I do not at present know the reasons for it. Here's how to try it for yourself:

> CLOAD a BASIC program, preferably one that's undergone lots of editing. When it's loaded, type "PRINT FRE(0)", and write down the number of unused memory bytes. Now store the source back to tape, using LIST"C". Clear the computer's BASIC program area by typing "NEW," and reload the code, using ENTER"C", from the tape just made. Finally, type "PRINT FRE(0)" again, and compare the result with the one you write down.

I have tried this on several existing programs, and have gotten memory savings anywhere from a

Now for adults.

Textwizard™ transforms Atari into a powerfully serious word processor.

Textwizard™ is no kids game. It's a sophisticated and complete word processing program for the **Atari 800 computer.

Here is instant control over the creation, editing and formating of any writing. Insert words. Replace phrases. Delete sentences. Move paragraphs. Automatically repositions surrounding text. *Textwizard™ does it all with over 50 simple commands.

On your command, Textwizard™ will search out and correct mistakes throughout your document. And it protects you from common operating errors by warning you before you make them.

Textwizard™ will print out as many original "copies" as you need. In boldface, elongated or condensed lettering, upper or lowercase, with proportional spacing. And you can store your work on a diskette.

So, whether you write legal briefs, computer programs, or The Great American novel, Textwizard™ can make your work easier. And though it's no toy, it's fun to use.

At $99.95, you don't have to write a best-seller to afford Textwizard.™ Textwizard™ is another of the many creative products from Datasoft.

Datasoft Inc.™
COMPUTER SOFTWARE

Software for people who aren't easy to please.

19519 Business Center Drive / Northridge, California 91324 / (213) 701-5161

Check your local software dealer or
Send check or money order with $2.00 postage/handling. California residents, add 6% sales tax.

*Textwizard will perform on a 32K system with one or more disc drives. It is compatible with the Atari® 825, Centronics® 737 and Epson® MX-80 printers.
**Atari is a registered trademark of Atari Computers Inc.

few bytes (who cares!) to over 500 bytes (impressive!). As I said earlier, I haven't delved into the reasons for this effect, but an observation of the results quickly shows one repeating pattern. The memory saving seems to be greatest when applied to programs that have been heavily "massaged" through editing, especially if variables have been used and then discarded. If true, the implication is that the Atari line-editing process is not one hundred percent effective in deleting program statements. Hopefully an inquisitive reader will determine exactly what it is worth for himself.

BASICALLY BASIC FILE HANDLING

As we have now seen, the Atari cassette system suffers from the worst drawbacks of all such systems. It is slow, file names are not supported, and only a single device can be used. Furthermore, except for the motor, the recorder is under manual control, and, in fact, the computer cannot even tell whether or not the tape is running! The combination of these characteristics pretty well precludes any but the most elementary file-handling functions. This section, therefore, may be seen mostly as an exercise with only minimum practical value.

It is possible to store and retrieve data via the cassette recorder from within a BASIC program using only built-in commands. Once again, the Atari BASIC Reference Manual alludes to this, but does not go into sufficient detail so that users can fully understand the methods. The problem is complicated, in this case, by an acknowledged OS "bug" which may thwart attempts to use the capability.

Due to the limited usefulness of the tape file functions, we will not spend much time on them, except to define how they are invoked.

The cassette recorder, like all Atari I/O devices, must be identified to the OS before it can be used. This is the function of the OPEN command, which links the cassette function to one of the INPUT/OUTPUT CONTROL BLOCKS (IOCB's). The command structure is OPEN#1,4,0,"C" to read from tape, or OPEN#1,8,0,"C" to write. You cannot open the cassette for simultaneous read and write, for obvious reasons. The first parameter in the OPEN command (#1, above) tells the system which of the eight IOCB's to assign. They are numbered from zero through seven, and as far as the OS is concerned they are all functionally identical. Stay away from #0 and #7, however, since these two IOCB's are "appropriated" by the OS for the screen (#0) and for its own tape functions (#7). The second parameter identifies the "direction" of data flow; either input to the computer ("4") or output from the computer ("8"). The third parameter is not used by the cassette system but must be present in the command ... use "0".

Once the cassette is opened, you may read or write data either as pure binary bytes or as ASCII-encoded character strings.

To write a single byte (any value from 0 to 255), use the command PUT#1,X. (X is a representative name for any variable which evaluates to an integer value in the range just stated.) GET#1,X is the corresponding command which will read these values back. It is important to realize that floating-point numerical values are represented internally as multi-byte groups, and, as such, cannot be handled directly by PUT or GET.

The other method of saving and retrieving tape data is to use ASCII representation, with one character per byte. This method is considerably less efficient with tape space, but does allow you to store anything that can be represented in a PRINT statement. The PRINT command, in fact, is what you use to write such information, using PRINT #1;data. The data can be numeric variables, string variables, literals, arrays, or any combination of these, just as in an ordinary PRINT command. (In this way floating point values can be saved. The value 2.45, for example, would be written as a series of four ASCII bytes: 32,2E,34,35 (hexadecimal representation).) Be aware that the system will append the End-of-Line character (9BH) to the last data item unless the PRINT#1 statement is terminated with a semicolon (;).

The corresponding command for reading ASCII data back is INPUT#1,var. Here is where the EOL character just mentioned becomes very important-the INPUT#1 command will attempt to keep reading until it finds this character. INPUT#1, like PRINT#1, uses the same general rules as does its keyboard-related counterpart.

Following the last PUT#1 or PRINT#1 command, a cassette file absolutely must be closed, using CLOSE#1. If you do not do this, you will lose some of the data that you thought was written.

At this point we need to look at some of the internal "mechanics" used by the Atari OS to format your data into the 128-byte records which are actually recorded. A 131-byte block of memory locations is reserved for use as a cassette buffer. (This is identified in your BASIC Reference Manual Memory Map as addresses 3FDH through 47FH.) Each time you execute a PUT#1 or PRINT#1 command, the resultant data is temporarily stored in the last 128 bytes of this buffer. Each time the 128th byte is stored, the Atari immediately suspends operation of your program, starts the cassette motor, writes the data block, and stops the motor. The internal buffer pointer is then reset to zero, and control is returned to your program (which could actually be in the middle of dumping data from a PRINT#1 statement!). In order for any of this to happen, of course, the cassette must first be OPENed as already described.

Opening the device for a write operation causes this sequence of events: (1) two beeps are emitted from the console speaker, and the computer waits for you to enable the recorder; (2) the cassette motor is started; (3) approximately 20 seconds of "leader" tone is written; and (4) the motor is stopped ... OOPS!!! Correction-the motor keeps on running! Here is the OS "glitch" mentioned earlier. The motor will not stop until the first data record has been written. The easiest way to get around this is to immediately write a "dummy" record using zeroes or spaces, or any other data. Here's how to do it:

```
.
.
100 OPEN#1,8,0,"C"
110 FOR I=0 TO 127:PUT#1,0:NEXT I
.
.
```

Your program will, of course, have to allow for this dummy record when retrieving the data. Once the first record has been written, the system is back on good behaviour ... the motor is started only when the buffer needs to be dumped, and stopped promptly after writing.

Realizing that there is an intermediate data buffer between your program and the cassette clarifies the need to close all files. The CLOSE command immediately causes the buffer to be dumped to the tape, even though it may not be full. (This is why you will lose data if a file is not closed properly). After the last data record is written, the OS (automatically) appends an End-of-File record before stopping the tape. NOTE: The END statement automatically closes all open files, but STOP does not.

Please note that the IOCB number need not be #1 as in these examples, but must be the SAME value in all OPEN, GET, PUT, INPUT, PRINT and CLOSE statements accessing this particular file.

BRINGING IT ALL TOGETHER

The BASIC program listing which follows accomplishes two things: it illustrates one possible method of reading a tape file from BASIC, and it provides a function that will prove indispensable to you when you try some of the procedures that we have explored. The program, called "Hex Tape Dump," reads any Atari-recorded file having long IRG's, one record at a time. After each record is read, the contents of that record are displayed on the screen in both hexadecimal and ASCII form.

This program makes use of a very important bit of information not yet discussed. One of the additional control bytes automatically appended to each record prior to writing contains a code describing the length and nature of that record. This byte shows up at location 1023 (decimal) following a read, and can have one of three values, with the following meanings: 252 (decimal) means that the record just read is a full record; that is, it contains 128 bytes of valid data. 250 (decimal) indicates that the record is only partly filled. In this case only, the last byte of the buffer will contain the actual number of valid data bytes. 254 (decimal) says that this is an End-of-File record, in which all data bytes are zero. When you find this control code, you have read past the end of valid data.

"Hex Tape Dump" is an ordinary BASIC program in all respects, and it is self-prompting when entered exactly as shown. Try it, and then spend some time playing with the BASIC cassette functions. You will end up with an even greater appreciation of the capabilities of this machine called ATARI.

*ATARI is a registered trademark of Warner Communications Inc.

LISTING 1: HEX TAPE DUMP

```
10   GRAPHICS 0:DIM BUF (8)
15   TRUE=-1:FALSE=0:FBT=TRUE:TOP=TRUE:PAGE=FALSE
20   PRINT CHR$(125):PRINT"     >>> ATARI HEX
     TAPE DUMP <<<"
25   PRINT:PRINT:PRINT"POSITION TAPE TO START OF
     FILE, THEN"
30   PRINT"PRESS M PLAY M : M RETURN M & STAND BY...";
35   REM OPEN THE CASSETTE FOR READ
40   TRAP 1000:REC=1:OPEN #1,4,0,"C"
45   REM DSPLY EIGHT BYTES PER LINE
50   FOR I=1 TO 8
55   REM FILL LINE IF OUT OF DATA
60   IF PAGE THEN PRINT"-- ";:BUF(I)=0:GOTO 100
70   GOSUB 500:BUF(I)=BYTE:IF NOT TOP THEN 90
75   REM PRINT TOP-OF-PAGE HEADER
80   TOP=FALSE:PRINT CHR$(125):PRINT"> RECORD #";
     REC;"...";NBTS+1;" BYTES"
85   PRINT:PRINT" ----------HEX----------   -ASCII
     --":PRINT
90   GOSUB 2000:PRINT" ";
100  NEXT I:PRINT"  ";
105  REM NOW PRINT ASCII FROM BUF
110  FOR I=1 TO 8:BYTE=BUF(I)
120  IF BYTE<32 OR BYTE>122 THEN BYTE=46
130  PRINT CHR$(BYTE);:NEXT I
150  PRINT:IF NOT PAGE THEN 50
160  PRINT:PRINT"PRES M SPACE M TO CONTINUE...";
170  REC=REC+1:PAGE=FALSE:TOP=TRUE
175  REM WAIT FOR USER RESPONSE
180  IF PEEK(764)<>33 THEN 180
190  POKE 764,255:GOTO 50
500  REM READ BYTE & SET FLAGS
505  GET #1,BYTE:IF NOT FBT THEN 600
510  FBT=FALSE:NBTS=1:STAT=PEEK(1023)
520  IF STAT=250 THEN NBTS=PEEK(1151)
530  IF STAT=252 THEN NBTS=128
600  NBTS=NBTS-1:IF NBTS>0 THEN RETURN
610  FBT=TRUE:PAGE=TRUE:RETURN
1000 REM DONE OR ERROR
1005 PRINT:IF PEEK(195)=136 THEN PRINT CHR$(28);
     "*** READ PAST END-OF-FILE ***":END
1010 PRINT CHR$(28);"--TASK ABORTED...ERROR NUMBER ";
     PEEK(195):END
2000 REM PRINT ONE BYTE IN HEX
2005 NYB=INT(BYTE/16):GOSUB 2100
2010 NYB=BYTE-NYB*16
2100 IF NYB<10 THEN PRINT CHR$(NYB+48);:RETURN
2110 IF NYB<16 THEN PRINT CHR$(NYB+55);:RETURN
2120 PRINT " ! ";:RETURN
```

©

Atari Graphics: 16 Colors!

Clyde H. Spencer
Mountain View, CA

Would you like to be able to have graphics displays with more than four colors on the screen simultaneously? Would you find it useful to be able to draw dotted, colored lines or fill shapes with textured color? If your answer is yes, then read on and I will tell you how to do something that not only isn't documented, but if you could find someone at ATARI to talk about it, they would probably tell you "it can't be done without the GTIA chip." That is the creation (or at least simulation) of a variation of playfield graphics modes 9, 10 and 11.

I can almost hear you mumbling to yourself now, "I didn't know there were graphics modes higher than 8! And what is this thing called a GTIA chip?" Let's talk about the GTIA chip first. As I have been able to unscramble the history of this little wonder called George's Television Interface Adapter chip, it started out life as a custom designed prototype that would do everything that the production CTIA chip (the chip in the computer that handles the graphics) does and a little more. That little more was GRAPHICS modes 9, 10 and 11. They have a resolution of 80 pixels horizontally by 192 pixels vertically with up to 16 different colors or luminescences on the screen simultaneously, out of the 128 possible. But alas, the lost little chips were never put into production. In order to meet marketing deadlines, the simpler, less powerful CTIA chips were installed in the production model computers. That is the bad news! The good news is that the Operating System and 8K Shepardson BASIC were written with the ability to implement these higher graphics modes, if the GTIA were installed. This was presumably done either as a hedge against a last-minute marketing decision to put it in, or with a view to offering this chip sometime in the future as a model upgrade like new chrome on a "Detroit behemoth." But you needn't wait for next year's model. With a little PEEKing and POKEing and a modification of the display list (see the article by Patchett in Vol. 1, issue 6 of **COMPUTE!**) you too may have colored icing on your cake.

The trick is accomplished with a short subroutine that modifies the GRAPHICS mode 8 display list to look like what a GRAPHICS mode 10 display list should look like. This is accomplished by replacing the graphics type (instruction operation code) for GRAPHICS mode 8, in the display list, with the graphics type for GRAPHICS mode 10. Finally, it is necessary to poke a 10 into one of the appropriate Operating System "shadow" registers in RAM to make the system expect to find more than four colors to display. Table 1 is a listing of the subroutine which I call "TEN,". The subroutine starts at line number 32000, so that it can be appended easily to almost any size program. It is liberally sprinkled with remarks. All essential statements are multiples of 10; the other line numbers are REMark statements, which may be deleted.

Unfortunately, it is difficult to select any given desired combination of colors because there is an interaction between the color registers; setting a particular value may affect the others. Generally, it will require considerable experimentation. I have as yet been unable to discover how exactly to predict color selection. What I have observed is as follows: Color registers 0, 1 and 2, loaded with the SETCOLOR command, create solid colors for "DATA" values (see "PALLETTE", Table 2) of 5, 10 and 15, respectively. The other "DATA" values will create colors that may be interspersed with the background color. However, SETCOLOR 0 also effects "DATA" values 1, 4, 6, and 9. Similarly, SETCOLOR 1 effects 2, 6, 8 and 9 and SETCOLOR 2 effects 3, 7, 11, 12, 13 and 14. By appropriate choice of color and luminescence values for the color registers, more than 3 solid colors may be created. There always seems to be some duplication of colors, but you can expect to get three to nine solid colors and an additional five or six colors that will create dotted lines. To get an idea of the colors that can be created, run "PALETTE." It will draw bands of color corresponding to "DATA" values of zero to 15, from the top down. Experiment with changing the constants A, B and C and see what happens! If anyone can shed more light on this subject, I would appreciate hearing about it.

A printout of the various modes' display lists would be instructive on how the ATARI does its graphics displays. But, with 198 addresses and their contents, it would consume too much space in this article. Alternatively, I have provided you with a short program called "DISPLAYLIST" (see Table 3) that will dump the standard display lists to your printer. If you wish to examine the "special" display list also, then first merge the subroutine "TEN" with "DISPLAYLIST." To merge the subroutine, first LOAD "TEN", then use the command LIST "C: or LIST "D:TEN". Then LOAD "DISPLAYLIST" and use the direct mode command ENTER "C: or ENTER "D:TEN" for cassette or disk respectively. Then delete lines 100 through 126, and replace line 130 with a GOSUB 32000. Then RUN as usual. For a detailed explanation of the display lists and the meaning of the

LETTER PERFECT T.M. LJK

WORD PROCESSING FOR THE *ATARI — 800 T.M.

MAIN - MENU

CURRENT DRIVE NUMBER #1

→ Editor ←
Change Drive #
Load
Save
Merge
Screen Format
Printer
Lock
Unlock
Delete
Format Disk
Data Base Merge
Quit

Press '<' or '>' to move cursor
Press (Return) for selection

USE: EPSON MX-80
and ATARI -825
PRINTERS

EASY TO USE: LETTER PERFECT is a character orientated word processor with the user in mind. The program (machine language) is very fast. It is a menu driven program that is very easy to operate. The program is a single load program and can work with one or more disk drives. It requires a minimum of 16K of memory and a single disk drive. With the Artari 825 printer you can print text with right hand justification. You may also use different type fonts (10 and 17 character per inch) within the body of the text itself. Boldface is printed as expanded print font. Underlining can be done as well as sending Escape characters within the body of the letter itself. All the formats are a default but you can change them all to desired values if you wish. Right Margin, left margin, top of form, line spacing, etc. are easily changed. Data Base Merge works with the sister program LETTER PERFECT—DATA BASE MANAGER. User may use this program to create mailing lists, and completely develop your own data base for your personal needs. All text packed before storage to diskette for greater storage capacity. Large Buffer allows you to pick up and move up to one full page of screen text and move it to any location in the text. Merge more than one file together for easy editing. Screen Format allows you to see on the video screen exactly how the text will appear on the printer. Automatic page numbering, headers and footers are easily accomplished. This program is easy to use because of its meaningful and easily mastered commands. Fully documented with a users manual that explains in simple language 'how to' completely use the program.

All this and more, for $149.95.

Features:

FULL CURSOR CONTROL
Home Cursor
Scroll Page Forward
Scroll Page Backward
Pause Scroll
Scroll Line at Time
Scrolling Speed Control
Move Cursor Down
Beginning of Text

MULTIFUNCTION FORMAT LINE
Standard Formats a Default
Formats Easily Changed
Right Justification
Left Margin
Page Width
Line Spacing
Lines Per Page
Form Stop
Set Page#
Top Margin
Bottom Margin

Delete a Character
Insert a Character
Delete a Line
Insert a Line
Headers and Footers
Shift Lock and Release
Global and Local Search and Replacement
Underlining and Boldface
Automatic Centering
Horizontal Tabs
Special Print Characters
Split Catalog
Page Numbering up to 65535
Prints up to 255 Copies of Single Text File
Non Printing Text Commenting

FUNCTIONS
Delete All Text
Delete All After Cursor
Delete All Before Cursor
Delete Next Block
Delete Buffer
Move Next Block to Buffer
Add Next Block to Buffer
Insert Block From Buffer
Merge Text Files

apple®

This program also available on the Apple in 40/80 Video (Super'R' Term, Smarterm, Videx, Bit-3). You may use any printer type. The Hays Micromodem II can be used to send files. Can be Reconfigured at any time to use different printer, 80 column board, or standard 40 column video. Much, Much, More!

DEALER INQUIRIES INVITED

LJK T.M.
COMPUTER BASED SOFTWARE
ENTERPRISES
Inc.

FREE CONTROL PAGE

LJK ENTERPRISES INC.,
P.O. Box 10827
St. Louis, MO 63129
(314) 846-6124

operation codes, I suggest that you refer to IRIDIS #1 by The Code Works.

The illustration program which I call "SPRAY" (Table 4), draws colored lines of different length which appear to radiate from a common point in the center of the screen. The color register values used were arrived at by experimentation with "PALETTE" and seem to give the greatest variety of solid colors. The colors are changed randomly by changing the values of color register 2 (SETCOLOR 2). If you allow the program to run long enough for the attract mode to be invoked by the Operating System, then all the color registers will be changed automatically. This will then also change the background color and lend additional interest to the patterns formed.

The resolution of GRAPHICS modes 9, 10, and 11 (192v X 80h) is a departure from the usual ATARI practice of having twice as many pixels horizontally as vertically. However, that very departure from the "standard" is itself an additional advantage in that it provides flexibility in what can be done with "direct" graphics modes. One does not have to resort to "mixed-modes" to achieve special effects. "Mixed-Modes" graphics, where one custom-tailors the display list, line by line, allows virtually unlimited flexibility in designing a screen format, but I would not recommend this to the novice. However, this ready-made subroutine approach for multiple colored and dotted lines is simple and straightforward and should further your appreciation of that wonderful and versatile machine, the ATARI.

TABLE 1. Subroutine TEN.

```
32000 REM : *TEN* a Subroutine to
32001 REM : create a simulated GR.10.
32002 REM : Written by Clyde Spencer.
32003 REM
32004 REM
32005 REM : Create GR.8 mode display
32006 REM : list without text.
32010 GRAPHICS 8+16
32013 REM
32014 REM
32015 REM : Locate address of display
32016 REM : list.
32020 LET DL=PEEK(560)+256*PEEK(561)
32023 REM
32024 REM
32025 REM : Turn off the ANTIC chip.
32030 POKE 559,0
32033 REM
32034 REM
32035 REM : Place new instruction op
32036 REM : codes in display list.
32040 POKE DL+3,78:POKE DL+99,78
32043 REM
32044 REM
32045 REM : Begin 1st insertion loop.
32050 FOR INSERT=DL+6 TO DL+98
32060 POKE INSERT,14
32063 REM
32064 REM
32065 REM : Increment insertion loop.
32070 NEXT INSERT
32073 REM
32074 REM
32075 REM : Begin 2nd insertion loop.
32080 FOR INSERT=DL+102 TO DL+198
32090 POKE INSERT,14
32093 REM
32094 REM
32095 REM : Increment insertion loop.
32100 NEXT INSERT
32103 REM
32104 REM
32105 REM : **Change timing.**
32110 POKE 87,10
32113 REM
32114 REM
32115 REM : Turn on ANTIC chip.
32120 POKE 559,34
32125 REM
32130 RETURN
```

TABLE 2. PALETTE Program.

```
50 REM : *PALETTE* a demonstration of
51 REM : multiple colors in GR.10.
52 REM : Written by Clyde Spencer.
53 REM
54 REM
95 REM : Go to subroutine to create
96 REM : simulated GR.10.
100 GOSUB 32000
103 REM
104 REM
105 REM : Assign  register variables.
110 LET A=1
120 LET B=5
130 LET C=10
133 REM
134 REM
135 REM : Set color registers.
140 SETCOLOR 0,A,C
150 SETCOLOR 1,B,C
160 SETCOLOR 2,C,C
163 REM
164 REM
165 REM : Begin drawing loops.
170 FOR DATA=0 TO 15
173 REM
174 REM
175 REM : Set color value.
180 COLOR DATA
183 REM
184 REM
185 REM : Begin inner drawing loop.
190 FOR BAR=10 TO 20
193 REM
194 REM
195 REM : Draw lines.
200 PLOT 0,10*DATA+BAR
210 DRAWTO 79,10*DATA+BAR
213 REM
214 REM
215 REM : Increment inner drawing loop.
220 NEXT BAR
223 REM
```

```
224 REM
225 REM : Increment outer 'data' loop.
230 NEXT DATA
233 REM
234 REM
235 REM : Enter an infinite loop to
236 REM : keep screen from clearing.
237 REM : Hit 'BREAK' key to rerun.
240 GOTO 240
```

TABLE 3. DISPLAYLIST Program.

```
50 REM : *DISPLAYLIST* A program to
51 REM : print out the address and
52 REM : operation codes of the dis-
53 REM : playlist.
54 REM : Written by Clyde Spencer.
55 REM
56 REM
95 REM : Reset TRAP and ask for INPUT
96 REM : again if input is not numeric.
100 TRAP 100
103 REM
104 REM
105 REM : Ask for GRAPHICS modes 0-40.
110 PRINT "WHAT GRAPHICS MODE";
113 REM
114 REM
115 REM : Input GRAPHICS mode numeric
116 REM : value.
120 INPUT MODE
123 REM
124 REM
125 REM : Set GRAPHICS mode and create
126 REM : new display list.
130 GRAPHICS MODE
133 REM
134 REM
135 REM : Locate starting address of
136 REM : display list.
140 DISPLAYLIST=PEEK(560)+PEEK(561)*256
143 REM
144 REM
145 REM : Begin loop to list contents
146 REM : of display list.
150 FOR ADDRESS=0 TO 200
153 REM
154 REM
155 REM : Assign new address to
156 REM : variable called CONTENTS.
160 LET CONTENTS=DISPLAYLIST+ADDRESS
163 REM
164 REM
165 REM : Print address and op codes.
170 LPRINT " ",CONTENTS,PEEK(CONTENTS)
173 REM
174 REM
175 REM : Check for JUMP op code (end
176 REM : of display list).
180 IF PEEK(CONTENTS)=65 THEN GOTO 200
183 REM
184 REM
185 REM : Increment print loop.
190 NEXT ADDRESS
193 REM
194 REM
195 REM : Remember to POP the stack if
196 REM : you will be doing anything
197 REM : else.
200 END
```

TABLE 4. SPRAY Program.

```
50 REM : *SPRAY* a demonstration of
51 REM : multiple colors in GR.10.
52 REM : Written by Clyde Spencer.
53 REM
54 REM
95 REM : Go to subroutine to create
96 REM : simulated GR.10.
100 GOSUB 32000
103 REM
104 REM
105 REM : Generate random number
106 REM : between 3 and 5.
110 LET C=INT(RND(0)*3)+3
113 REM
114 REM
115 REM : Set color registers.
120 SETCOLOR 0,1,8
130 SETCOLOR 1,10,8
140 SETCOLOR 2,C,8
143 REM
144 REM
145 REM : Set quadrant flag.
150 LET SIGN=1
153 REM
154 REM
155 REM : Begin nested drawing loops.
160 FOR DO=1 TO 2
170 FOR DATA=1 TO 15
173 REM
174 REM
175 REM : Assign drawing color.
180 COLOR DATA
183 REM
184 REM
185 REM : Pick random X&Y coordinates.
190 LET X=INT(RND(0)*40)
200 LET Y=SIGN*INT(RND(0)*96)
203 REM
204 REM
205 REM : Plot colored lines.
210 PLOT 40-X,96-Y:DRAWTO 40+X,96+Y
213 REM
214 REM
215 REM : Increment color loop.
220 NEXT DATA
223 REM
224 REM
225 REM : Reset quadrant flag.
230 LET SIGN=-1
233 REM
234 REM
235 REM : Increment symmetry loop.
240 NEXT DO
243 REM
244 REM
245 REM : Pause to appreciate.
250 FOR DELAY=1 TO 1000:NEXT DELAY
253 REM
254 REM
255 REM : Do it all again!
260 GOTO 110
```

©

Assembler Joystick Driver

James E. Korenthal
New York, NY

Tired of coding all those IF statements to separate a value returned by the STICK function into its X and Y components? This short assembler routine will do the trick, and give you faster executing, more readable code to boot!

In order to understand how the routine works, we have to look at values returned by STICK in binary notation:

Direction	Decimal	Binary E	W	S	N	X-val	Y-val (see below)
none	15	1	1	1	1	1	1
N	14	1	1	1	0	1	0
NE	6	0	1	1	0	2	0
E	7	0	1	1	1	2	1
SE	5	0	1	0	1	2	2
S	13	1	1	0	1	1	2
SW	9	1	0	0	1	0	2
W	11	1	0	1	1	0	1
NW	10	1	0	1	0	0	0

A glance at this table reveals that each direction is handled by one bit, with a zero value indicating that the joystick is pointed in that direction. The "E W S N" subheading in the binary column indicates the bit-direction correspondence. For example, a value of 9 is 1001 in binary, with 0's in the bits corresponding to south and west.

Now, how can we make use of this information? We can split up the east-west and north-south groups of two bits, and transform them into delta-x ("change in x") and delta-y values. Here's the assembler program:

```
STICKO  =  $0278              ;address of joystick 0 value
ANSLO   =  $D4                ;low byte of answer
ANSHI   =  $D5                ;high byte of answer
;
        * =     $0600         ;dummy origin (code is
                               relocatable)
        PLA                   ;discard number of argu-
                               ments
        PLA                   ;discard high byte of
                               stick number
        PLA                   ;get stick number (0-3)
        TAX                   ;use it as an index
        PLA                   ;discard high byte of
                               direction
        LDA     STICKO,X      ;get value for appropri-
                               ate direction
        PLP                   ;get direction indicator
                               in carry
        BCS     STICKY        ;does he want x (0) or
                               y (1)?
        LSR     A             ;he wants x, so shift down
                               2 bytes
        LSR     A
STICKY  AND     #3            ;mask off high bits
        SEC                   ;set carry for subtract
        SBC     #2            ;change 2 to 0, 3 to 1, 1 to -1
        BPL     SAVEIT        ;was it 1?
        LDA     #2            ;transform to 2 if so
SAVEIT  STA     ANSLO         ;save low byte of result
        LDA     #0            ;zap high byte
        STA     ANSHI
        RTS                   ;all done
        .END
```

Given a joystick number (0-3) and a direction (0 for X or horizontal or east-west and 1 for Y or vertical or north-south), this routine returns, for the correct joystick, a value shown the X-val or Y-val column of our table. You should verify that subtracting one from these values yields appropriate deltas (-1, 0, or 1) for screen positions. If standard deltas are desired for the y-axis, reverse the value (subtract it from one instead of the other way around).

The following demonstration program shows how to incorporate the joystick driver into your own BASIC programs. It simply monitors the status of all four joysticks, displaying standard deltas in x,y format:

```
10 REM Atari 800 Joystick Driver Demonstration
   Program
20 REM James E. Korenthal, 1981
30 GOSUB 1000 :REM load machine language code
40 X=0 : Y=1 :REM direction codes (for read-
   ability)
50 POKE 201,8 :REM use narrow columns
60 FOR NSTICK=0 to 3 :REM loop on all three
   sticks
70 DELTAX = USR(JOY,NSTICK,X)-1 :REM get
   horizontal change
80 DELTAY = 1-USR(JOY,NSTICK,Y) :REM get
   vertical change
90 PRINT DELTAX;",";DELTAY, :REM print
   values
100 NEXT NSTICK :REM end loop on sticks 0-3
110 PRINT :REM skip to next line
120 GOTO 60 :REM loop until break or reset is
    pressed
1000 REM subroutine to load joystick driver machine
     code
1010 DIM JOY$(29) : JOY=ADR(JOY$) :REM set up
    code & pointer
1020 FOR J=JOY to JOY+28 :REM read and store
     29 bytes
1030 READ BYTE : POKE J,BYTE : NEXT J :
     RETURN
1040 REM machine code goes here:
1050 DATA 104,104,104,170,104,189,120,2,40,176,2,
     74,74,41
1060 DATA 3,56,233,2,16,2,169,2,133,212,169,0,133,
     213,96
```

As an interesting exercise, try incorporating the joystick driver into Larry Isaacs' speed-up of Chris Crawford's Player-Missile Demo (Listing 2 on page 108, **COMPUTE!** Vol. 3, No. 4). Make sure you allow the ship to move in all directions (as is, the program only allows north, south, east, or west), and watch out for that RESTORE in line 1140! ©

Atari 800:

Assembly Language Routine To Eliminate DOS/FMS When They Are No Longer Needed

John Elliott
New York, NY

As all users of VERSION I of DOS will know, about 9K of RAM is taken up by the DOS/FMS routines. More than 4K of this RAM is needed only when you want to talk directly to DOS through the menu selection screen.

The DOS REFERENCE MANUAL describes a method (1) of releasing the RAM used for these functions. However, the BASIC program listed in the manual will run only when the BASIC cartridge is inserted. This presents a problem for those of us who are using the ASSEMBLER/EDITOR cartridge. How can we get the same RAM savings as our BASIC counterparts? By using assembler, of course!

The short assembly language program listed here will eliminate DOS and FMS when either the BASIC or ASSEMBLER/EDITOR cartridge is inserted. You can still use all the DOS functions that are controllable with BASIC and the ASSEMBLER /EDITOR keywords.

The routine is designed to be as general-purpose as possible, so I will describe an operational procedure which is independent of the cartridge being used. In fact, once the program is saved on diskette, it can be executed with no cartridge inserted.

Saving The Program To Diskette

Our first task is to get the object program into RAM. If you have the ASSEMBLER/EDITOR cartridge, use it to assemble the program, with the object program going to RAM (this is the assembler default). If you do not have the ASSEMBLER/ EDITOR cartridge, but you want to make use of the program, then you can use BASIC to POKE the program into RAM.

Our next task is to save the object program on diskette. Go to the DOS menu selection, by typing DOS (RETURN). Then follow the "BINARY SAVE" procedure described in the DOS manual (2). The session should proceed as follows:

SELECT ITEM
K (RETURN)
SAVE—GIVE FILE, START, END
DISOUT. OBJ, 600, 626 (RETURN)
SELECT ITEM

This SELECT ITEM prompt indicates that the program has been saved. Note that the file name is arbitrary.

Executing The Program

Now that we have the permanent copy on disk, we can load it into RAM and use it as and when we need it. To load it into RAM, go to the DOS menu selection, by typing DOS (RETURN). Then follow the "BINARY LOAD" procedure described in the DOS manual (3). The session should proceed as follows:

SELECT ITEM
L (RETURN)
LOAD FROM WHAT FILE?
DOSOUT. OBJ (RETURN)
SELECT ITEM

This SELECT ITEM prompt indicates that the program has been loaded into RAM. To execute it, follow the "RUN AT ADDRESS" procedure described in the DOS manual (4). The session should proceed as follows:

SELECT ITEM
M (RETURN)
RUN FROM WHAT ADDRESS?
600 (RETURN)
READY/EDIT

The computer will respond with either READY or EDIT, depending on whether you have the BASIC or the ASSEMBLER/EDITOR cartridge inserted. DOS & FMS have now been eliminated.

I hope you find this routine useful. If you find that you rarely use DOS directly, you may like to make use of the AUTO. SYS feature to automate the routine. Good luck!

Notes On The Listing

The program follows fairly closely the steps taken by the BASIC program listed in the DOS manual (1). You may find it interesting to compare the two.

Lines 140-160 : Define the origin and initialize the stack pointer.
Lines 170-280 : Adjust DOS/OS vectors.
Line 290 : Link to DOS.
Lines 300-320 : Jump to the cartridge initialization routine.

I have used PAGE 6 to hold the object program. However, the routine is relocatable, so you may locate it somewhere else in RAM if you need locations $600-$626 for some other purpose.

As the comment at line 120 states, eliminating DOS & FMS releases 5200 bytes of RAM. The DOS REFERENCE MANUAL (1) states that the 5384 bytes of RAM are released, but this includes the length of the BASIC routine — 184 bytes.

References.

(1) ATARI DISK OPERATING SYSTEM REFERENCE MANUAL, C015200 rev.1, appendix C.3, page 58.
(2) Ibid. Page 36.
(3) Ibid. Page 38.

(4) Ibid. Page 38.

```
LOC  OBJECT    SOURCE STATEMENT

               ;ROUTINE TO ELIMINATE
               ;      DOS & FMS
               ;RELEASES 5200 BYTES
               ;
0000                  *=    $600
0600 A2FF             LDX   #$FF
0602 9A               TXS
0603 A923             LDA   #$23
0605 850A             STA   $0A
0607 A9F2             LDA   #$F2
0609 850B             STA   $0B
060B A988             LDA   #$88
060D 850C             STA   $0C
060F A907             LDA   #$07
0611 850D             STA   $0D
0613 A930             LDA   #$30
0615 8D0C07           STA   $70C
0618 A912             LDA   #$12
061A 8D0D07           STA   $70D
061D 208807           JSR   $788
0620 A900             LDA   #$00
0622 8508             STA   $08
0624 6CFABF           JMP   ($BFFA)
0627                  .END
```

©

Atari Data

Robert W. Baker
Atco, NJ

While recently converting a simple program from the PET to run on the Atari, I came across a few new quirks in Atari BASIC that I believe have not yet been documented. The problems have to do with using strings in DATA statements on the Atari. There is a vast difference in the ways Microsoft and Atari handle this.

It appears that Atari uses commas to separate DATA elements regardless of where they appear. Even if a string is enclosed in quotes, commas are still recognized and create separate data elements. Adding quotes actually creates another problem since they are not optional. Any quotes in the DATA statement will actually be read as part of the string data. Here's a simple program that will quickly illustrate how the DATA statement works on the Atari:

```
10 TRAP 70
20 DIM A$ (40)
30 DATA "TEST, WITH, COMMAS"
40 READ A$
50 PRINT A$
60 GOTO 40
70 END
```

When you run this program you'll see:

```
"TEST
WITH
COMMAS"
```

These three lines show that the commas are still recognized and actually create three separate data elements instead of one. Also, notice that the quotes are still part of the data as are the spaces after the commas. Thus, whenever placing strings within DATA statements on the Atari, you cannot have commas as part of the data. Also, there's no real reason to use quotes unless they're actually wanted in the data. You do not need quotes at all, even when there are spaces within the string constant.

While I'm at it, here's a copy of the program I converted for the Atari. It's a program that I use to record birthdays, anniversaries, and other important dates. The program allows you to display or print the recorded dates for any month, or the entire list. It has an option to supress special dates unless specifically requested.

The information for any date is stored in separate DATA statements. The first five characters are the actual date in the form of "MM/DD". This is followed by two spaces and any specific data associated with that date. Special dates are identified by an asterisk as the first character in the data for that date (see program line 1000). The last DATA entry must be the word "END" to terminate the list correctly.

For convenience, I normally use a separate line for each DATA statement constructed from the date itself. This makes the line very easy to locate and avoids duplication. Typically I make the month the thousands digits (1000-12000) and the day of the month the hundreds and tens digits (010-310). This leaves the ones digits for multiple events on the same date, up to 10 maximum of course.

Data recorded in the program is not sorted or re-ordered in any way. Dates are listed in the exact order found in reading the DATA statements, so they should be stored in the correct order. Whenever you update the program remember to save a new copy. For added convenience, you may want to also include the date of the last update as well (see program line 50). Just remember to avoid using commas in the data strings as discussed earlier!

```
10 REM ----+------------------
20 REM DATE BOOK - FOR ATARI
30 REM BY: ROBERT BAKER
40 REM -----------------------
50 REM LAST UPDATE:  MM/DD/YY
60 REM -----------------------
65 OPEN #1,4,0,"K"
67 DIM R$(40),M$(2)
70 PRINT CHR$(125);"*** DATE BOOK *** IM
PORTANT DATES ***"
80 PRINT :PRINT
90 PRINT "DISPLAY MONTH"
95 PRINT "(1-12, A=ALL, S=SPCL)";
100 INPUT M$:IF M$="" THEN 800
105 IF M$<"A" THEN M=VAL(M$)
110 IF M$="A" OR M$="S" THEN 200
120 IF M<1 OR M>12 THEN 800
200 PRINT :PRINT
210 P=0:PRINT "WANT PRINTED COPY (Y/N)";
220 INPUT R$:IF R$="Y" THEN P=1
300 PRINT CHR$(125);:RESTORE :L=0
310 READ R$
320 IF R$="END" THEN GOSUB 900:GOTO 70
330 IF R$(8,8)<>"*" THEN 400
340 IF M$="S" THEN PRINT R$:L=L+1:IF P=1
 THEN LPRINT R$
350 GOTO 500
400 IF M$="A" OR VAL(R$(1,2))=M THEN PRI
NT R$:L=L+1:IF P=1 THEN LPRINT R$
500 IF L=20 THEN GOSUB 900:L=0
510 GOTO 310
800 CLOSE #1:END
900 PRINT :PRINT "PRESS ANY KEY TO CONTI
NUE";
920 GET #1,X
950 PRINT CHR$(125);:RETURN
1000 DATA 01/01  *D* GUESS WHO
12250 DATA 12/25  JANE DOE (1925)
12300 DATA 12/30  ME TWO (1950)
32000 DATA END
```

©

Memory Protection For Atari

Jim Clark
Seattle, WA

```
10 REM MEMPROT.BAS
20 REM by Jim Clark
30 REM 2415 East McGraw St.
40 REM Seattle, WA 98112
100 REM Load assembly language subroutine
110 PGMSIZ=24:DIM SUBR$(PGMSIZ)
120 FOR I=1 TO PGMSIZ
130 READ BYTE
140 SUBR$(I)=CHR$(BYTE)
150 NEXT I
200 REM Get amount of memory to protect
210 PRINT "How many bytes do you want to protect";
220 INPUT PROTECT
230 HI=INT(PROTECT/256):LOW=PROTECT-256*HI
240 SUBR$(6,6)=CHR$(LOW)
250 SUBR$(14,14)=CHR$(HI)
300 REM Reinitialize BASIC with the new low memory pointer
310 Z=USR(ADR(SUBR$))
400 REM Assembly language subroutine
410 REM MEMLO =$02E7 ;BOTTOM OF AVAILABLE USER MEMORY
420 REM WARMST=$08   ;WARM START FLAG
430 REM CARTA =$A000 ;BASIC CARTRIDGE ENTRY POINT
440 REM
450 REM ;THE PROGRAM IS COMPLETELY RELOCATABLE,
460 REM ;SO NO STARTING ADDRESS IS PROVIDED
470 REM
500 REM CLC                  ;INITIALIZE FOR ADDITION
510 DATA 24
520 REM LDA MEMLO            ;ADD LEAST-SIGNIFICANT BYTES
530 DATA 173,231,2
540 REM ADC #PROTECT&$FF
550 DATA 105,0
560 REM STA MEMLO
570 DATA 141,231,2
580 REM LDA MEMLO+1          ;ADD MOST-SIGNIFICANT BYTES
590 DATA 173,232,2
600 REM ADC #PROTECT/256
610 DATA 105,0
620 REM STA MEMLO+1
630 DATA 141,232,2
640 REM LDA #0               ;RESET THE WARM START FLAG
650 DATA 169,0
660 REM STA WARMST
670 DATA 133,8
680 REM JMP CARTA            ;START BASIC OVER AGAIN
690 DATA 76,0,160
999 END
```

A problem arises in applications which require that a portion of memory be protected from BASIC on the Atari Computer. For example, most assembly language subroutines need protection. The problem is that BASIC is likely to use memory anywhere within available RAM, thus writing over the assembly language subroutine and destroying it.

In many computers it is possible to protect memory at the "high" end, i.e., at the highest RAM address. The Atari uses high memory for the data which is displayed on the screen. If you attempt to protect memory above the screen display by reducing the high memory value that BASIC thinks it has, then you cannot clear the screen or scroll text in any of the split-screen modes because these actions affect memory *beyond* the screen display area. These actions cause no problem when the screen display is actually the last thing in memory, because they apply to nonexistent memory. However, if you want to use memory beyond the screen display for your own purposes, then your data will be damaged by any action in your program which clears the screen or scrolls text in a text window.

Another alternative is to protect low memory. The main problem with this approach is that the memory protection must be done before BASIC gets control, since BASIC starts saving any program you enter beginning at the low memory address. The program shown here solves this problem as follows: it takes control of the Atari in an assembly language subroutine and resets the system's low memory pointer. It then reinitializes BASIC — just as if you had pressed the SYSTEM RESET key — and BASIC takes control again, blissfully unaware that it now has less RAM to work with than it did before you ran this program.

To find the address of the memory you have protected, type ?PEEK(743) + 256*PEEK(744) *before* you run this program. The number printed can be used as the origin for an assembly language subroutine, or as the destination address for whatever data you want to store in the protected area.

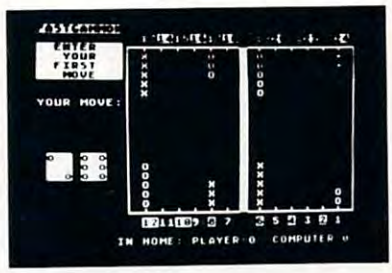
When you run the program, it asks how much memory you want to protect. Type in any positive number which is less than the amount of RAM you have available, as determined by typing ?FRE(0). The program reinitializes BASIC, and if you type ?PEEK(743) + 256*PEEK(744) again, the number printed will be greater than the value shown before running the program: the difference is the amount you requested to be protected. The memory area will remain protected until you turn the computer off, and the area can be used for assembly language subroutines, redefined character sets, player-missile graphic objects, or any other use you might wish. ©

SOFTWARE FOR THE ATARI 800* AND THE ATARI 400*

TARI TREK™
By Fabio Ehrengruber

Get ready for an exciting trek through space. Your mission is to rid the galaxy of Klingon warships, and to accomplish this you must use strategy to guide the starship Enterprise around stars, through space storms, and amidst enemy fire. Sound and color enliven this action-packed version of the traditional trek game. Nine levels of play allow the player to make the mission as easy or as challenging as he wishes. At the highest level you are also playing against time. Damage to your ship can be repaired in space at a cost of time and resources if you can't make it back to base. TARI TREK gives you a lot of trek at a low price. This program is written entirely in BASIC and requires at least 24K of user memory. For the Atari 800 only.

Cassette - $11.95 Diskette - $14.95



FASTGAMMON™
By Bob Christiansen

Play backgammon against a talented computer opponent. This is the latest and best version of the most popular backgammon-playing program for personal computers - FASTGAMMON. Roll your own dice or let the computer roll them for you. Adjust the display speed to be fast or slow. If you wish you can play a game using the same dice rolls as the previous game - a great aid in improving your skills at backgammon. Beginners find it easy to learn backgammon by playing against the computer, and even very good players find it a challenge to beat FASTGAMMON. The 12-page instruction booklet includes the rules of the game. Written in machine language. Requires only 8K of RAM and runs on both the Atari 400 and the Atari 800.

On cassette only - $19.95

TANK TRAP
By Don Ursem

A rampaging tank tries to run you down. You are a combat engineer, building concrete barriers in an effort to contain the tank. Use either the keyboard or an Atari joystick to move your man and build walls. If you trap the tank you will be awarded a rank based on the amount of time and concrete you used up. But they'll be playing taps for you if you get run over. There are four levels of play. Higher levels of play introduce slow curing concrete, citizens to protect, and the ability of the tank to shoot through any wall unless you stay close by. Music, color, and sound effects add to the excitement. Written in BASIC with machine language subroutines. Requires at least 16K of user memory. Runs on the Atari 800 and on an Atari 400 with 16K RAM.

Cassette - $11.95 Diskette - $14.95

QS FORTH™ By James Albanese. Step into the world of the remarkable FORTH programming language. Writing programs in FORTH is much easier than writing them in assembly language, yet FORTH programs run almost as fast as machine code and many times faster than BASIC programs. QS FORTH is based on fig-FORTH, the popular model from the FORTH Interest Group that has become a standard for microcomputers. QS FORTH is a disk-based system that can be used with up to four disk drives. There are five modules included:

1. The FORTH KERNEL (The standard fig-FORTH model customized to run on the Atari computer).
2. An EXTENSION to the basic vocabulary that contains some handy additional words.
3. An EDITOR that allows editing source programs (screens) using Atari type editing.
4. An IOCB module that makes I/O operations easy to set up.
5. An ASSEMBLER that allows defining FORTH words as a series of 6502 assembly language instructions.

Modules 2-5 may not have to be loaded with the user's application program, allowing for some efficiencies in program overhead. Full error statements (not just numerical codes) are printed out, including most disk error statements. QS FORTH requires at least 24K of RAM and at least one disk drive. For the Atari 800 only.

On diskette only - $79.95

★ ★ ★ ★ ★ ★ ★ ★ ★

ASSEMBLER by Gary Shannon. Write your own 6502 machine language programs with this inexpensive in-RAM editor/assembler. Use the editor to create and edit your assembler source code. Then use the assembler to translate the source code into machine language instructions and store the code in memory. Simple commands allow you to save and load the source code to and from cassette tape. You can also save any part of memory on tape and load it back into RAM at the same or at a different location. The assembler handles all 6502 mnemonics plus 12 pseudo-ops that include video and printer control. Commenting is allowed and error checking is performed. A very useful feature allows you to view and modify hexadecimal code anywhere in memory. Instructions on how to interface machine language subroutines to your BASIC programs are included. ASSEMBLER requires 16K of user memory and runs on both the Atari 800 and the Atari 400.

On cassette only - $24.95

★ ★ ★ ★ ★ ★ ★ ★ ★

6502 DISASSEMBLER by Bob Pierce. This neat 8K BASIC program allows you to disassemble machine code, translating it and listing it in assembly language format on the video and on the printer if you have one. 6502 DISASSEMBLER can be used to disassemble the operating system ROM, the BASIC cartridge, and machine language programs located anywhere in RAM except where the DISASSEMBLER itself resides. (Most Atari cartridges are protected and cannot be disassembled using this disassembler.) Also works as an ASCII interpreter, translating machine code into ASCII characters. 6502 DISASSEMBLER requires only 8K of user memory and runs on both the Atari 800 and the Atari 400.

Cassette - $11.95 Diskette - $14.95

QS QUALITY SOFTWARE
6660 Reseda Blvd., Suite 105, Reseda, CA 91335
(213) 344-6599

WHERE TO GET IT: Call us at (213) 344-6599 for the name of the Quality Software dealer nearest you. If necessary you may order directly from us. Mastercard and Visa cardholders may place orders by telephone. Or mail your check or bankcard number to Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91335. California residents add 6% sales tax. SHIPPING CHARGES: Within North America orders must include $1.50 for first class shipping and handling. Outside North America the charge for airmail shipping and handling is $5.00. Pay in U.S. currency.

*Indicates trademarks of Atari, Inc.

Program Development In Atari Basic

Paul E. Hoffman
Cambridge, MA

The Atari microcomputers are good machines on which to program, especially in Basic. The editor is easy to use and has many nice features. Unfortunately, developing large programs on the Atari is not as easy as one would like due to the lack of dynamic program chaining.

Optimally, you should be able to make many program fragments and be able to run a linking program that would look like:

10 ENTER 'D:PROG1'
20 ENTER 'D:PROG2'
30 ENTER 'D:PROG3'

and so on, that would bring all of the programs together. The problem is that the Atari operating system stops the program right after an ENTER, so that a running program can only have one ENTER command.

The way around this restriction is to use the screen as a device to give the ENTER commands to link the large program. Of course, you can simply type the commands in each time they want to link the programs. It is easier, however, to have the Atari display the ENTER commands; all you have to do is RETURN over them.

Breaking up a longer program into modules that are about two screens long makes debugging programs much easier. Small program fragments can be listed completely without having to guess which line range you want. Also, the fragments can be broken up into logical units of the program, such as the variable definitions, screen setup, and so forth.

Developing a program using fragments has some drawbacks, but they are easy to overcome. You must be careful about using GOTO's across segments, since you may decide to renumber some lines in the segment to which you are jumping. This problem can be avoided by using as few GOTO's as possible, and by always going to REM statements that tell you which lines to change if the line you are going to needs to be renumbered.

Subroutines can be treated the same as statements that are gone to, but there is a more efficient way to deal with them. GOSUB's that point to the beginning of a program are found by the program more quickly than those that point to the end, so putting all of your subroutines in the first module of a program will make your program run faster. If all you use the first module for is to define constants and subroutines, it is unlikely that you will need to renumber much, thus avoiding the problems with GOTO's.

There are two other restrictions for using this program. The first is that you can only have up to ten modules, due to the amount of space each ENTER takes on the screen. Second, if you renumber your program using a Basic renumbering program available from some third-party vendors, you should only renumber the full program to avoid incorrect GOTO's.

The program in Listing 1 can be used to display the ENTER commands on the screen to link together 10 programs called PROG1, PROG2, ... You can change the name of the program in lines 30 and 60 to any name you want. Once the program has been run, simply hit RETURN's over each line, and your program will be linked and in memory.

Line 10 clears the screen; lines 20 through 50 display the first five ENTER's (for PROG1 through PROG5). The CHR$(34) is a quote mark; using CHR$(n) instead of the character itself in program listings makes them easier to read and reproduce.

Lines 60 and 70 write a one-line program that clears the screen and displays the next five enters (for PROG6 through PROG10). If you have five or less program fragments to link, you can eliminate lines 50 through 70. Line 80 puts the cursor over the first ENTER (after the READY prompt appears), then line 90 clears the linking program from memory.

Using the above scheme for writing large programs should make them easier to edit and develop. We can always hope that future Atari versions of Basic (as well as other programming languages) give more flexibility in program linking so that programs like this one are not necessary.

```
10 ? CHR$(125):REM CLEAR THE SCREEN
20 FOR I=1 TO 5
30 ? :? :? "ENTER ";CHR$(34);"D:PROG";I;CHR$(34)
40 NEXT I
50 ? :?
60 ? "POKE84,1:POKE85,0:FORI=6TO10:?:?:?";CHR$(34);"ENTER ";CHR$(34);";CHR$(34);
";CHR$(34);"D:PROG";CHR$(34);
70 ? ";I;CHR$(34):NEXT I:POKE 84,1:POKE 85,0"
80 POKE 84,1:POKE 85,0
90 NEW
```

©

THE OSI® GAZETTE

OSI C1P Newspaper Route Listing Program Part Two

Charles L. Stanford
Cinnaminson, NJ

Running The Program

With all the preambles and caveats out of the way, let's take a look at the program. Line 0 through whatever contain the customer list. I found it easiest to enter the first time directly as DATA Statements. We devised a form (see Listing 2), and John filled it out from his collection cards and entered the data into the program over a period of several days. Note that Line 0 contains the number of customers. We vary that to match the route, but keep the number of DATA lines at 75. The program doesn't care as long as there are more DATA's than READ's. Be very careful, however, that every line has an identical number of characters.

When printing to the screen or to an external device, Line 425 can be used to select the format of the list. If you want only the Sunday route, use a line such as IF D(X) <> 1 AND D(X) <> 3 AND D(X) <> 6 THEN GOTO 460. Note that AND and not OR is used for <> IFs.

Listing 2

```
 10 INPUT"NUMBER OF CUSTOMERS";N
 20 PRINTTAB(12);"NAME";
 30 PRINTTAB(25);"No.";TAB(35);"St.";
 40 PRINTTAB(41);"D";TAB(45);"P"
 50 FORX=1TON
 60 PRINTRIGHT$("  "+STR$(X),2);
 70 PRINT"DATA";
 80 PRINT": : : : : : : : :,";
 90 PRINT": : : :,: : :,: :,: :"
 100 NEXT
OK

OK
RUN
NUMBER OF CUSTOMERS? 5
            NAME         No.       St.   D   P
 1DATA: : : : : : : : :,: : : :,: : :,: :,: :
 2DATA: : : : : : : : :,: : : :,: : :,: :,: :
 3DATA: : : : : : : : :,: : : :,: : :,: :,: :
 4DATA: : : : : : : : :,: : : :,: : :,: :,: :
 5DATA: : : : : : : : :,: : : :,: : :,: :,: :
OK
```

The BASIC Garbage Collector Bug

There is one more problem involved in getting this program to run successfully. That's the Garbage Collection Bug in BASIC ROM Number 3. Most OSI owners probably already know about this problem, but as an assist for those who are new to it, a short background. When the ROMs were programmed, there were two code errors within the Garbage Collector Routine starting at $B147 in ROM 3. The errors don't stop the routine from running, but they sure keep it from doing anything useful. The GC is necessary to undo the damage to memory caused by the accumulation of strings in upper RAM. When a string is concatenated, or even recreated, all versions are retained. If you say that A$ = A$ + A$, both versions stay in existence. You can see that an active routine such as the one at Line 900 of Listing 1 will soon use up all the RAM. But the GC is automatically called whenever RAM gets short, and all the redundant strings are discarded. On the OSI, not only doesn't this happen, the whole program hangs up, the screen pulses, and only a quick "BREAK" can save source code destruction.

There are several solutions. One is to buy one of the corrected PROMs available from several sources. But Rodger Olsen of Aardvark advised me that even a repaired OSI GC isn't perfect. He suggests the software fix in **The (Real) First Book of OSI**. I just ordered my copy, so I can't comment. What I have done is include a fix I devised, which puts a repaired OSI GC in Page 2 of RAM (the unused part starting at $0222). It is shown in Listing 3. You must remember that a BREAK will require that the Vector at locations 11 and 12 (Dec) be reset. The GC will not fit between $0222 and $02FF, so it

DOWN-TO-EARTH PRICES ON OUT-OF-THIS-WORLD PERSONAL COMPUTERS AND COMPONENTS.

Look at this!

Ohio Scientific Superboard II

$299

- It's the first complete computer system on a board.
- Superboard II uses the ultra powerful 6502 Microprocessor
- 8K Microsoft BASIC-in-ROM
- 4K static RAM on board, expandable to 8K
- Full 53-key keyboard, with upper and lower case. Plus user expandability.
- Video interface and audio cassette interface.

The Ohio Scientific Superboard II at $299 — in today's economy — has got to be the best buy by far. It will entertain you with spectacular graphics made possible by its ultra high resolution graphics and super fast BASIC. It will help you in school or industry, as an ultra powerful scientific calculator. Advanced scientific functions and a built-in "immediate" mode allow you to solve complex problems without programming.

The Superboard II can be expanded economically, for business uses, or to remotely control your home appliances and security. Even communicate with other computers.

Read what's been written about Superboard II:

"We heartily recommend Superboard II for the beginner who wants to get into microcomputers with a minimum cost. A real computer with full expandability."

—POPULAR ELECTRONICS, MARCH 1979

"The Superboard II is an excellent choice for the personal computer enthusiast on a budget."

—BYTE, MAY 1979

Look at these easy hardware prices:

610 Board For use with Superboard II and Challenger 1P. 8K static RAM. Expandable to 24K or 32K system total. Accepts up to two mini-floppy disk drives. Requires +5V @4.5 amps.	**$ 298**
Mini-Floppy Disk Drive Includes Ohio Scientific's PICO DOS software and connector cable. Compatible with 610 expander board. Requires +12V @1.5 amps and +5V @ 0.7 amps. (Power supply & cabinet not included.)	**299**
630 Board Contact us for important details.	**229**
AC-3P 12″ combination black and white TV/video monitor.	**159**
4KP 4K RAM chip set.	**79**
PS-005 5V 4.5 amp power supply for Superboard II.	**45**
PS-003 12V power supply for mini-floppies.	**45**
RF Modulator Battery powered UHF Unit.	**35**
CS-900B Metal case for single floppy disk drive and power supply. (While stock lasts.)	**49**
AC-12P Wireless remote control system. Includes control console, two lamp modules and two appliance modules, for use with 630 board.	**175**
AC-17P Home security system. Includes console, fire detector, window protection devices and door unit for use with 630 board.	**249**
C1P Sams C1P Service manual	**8**
C4P Sams C4P Service manual	**16**
C3 Sams Challenger III manual	**40**

Ohio Scientific and independent suppliers offer hundreds of programs for the Superboard II, in cassette and mini-floppy form.

Freight Policies All orders of $100 or more are shipped freight prepaid. Orders of less than $100 please add $4.00 to cover shipping costs. Ohio residents add 5.5% Sales Tax.

VISA

Hours: Call Monday thru Friday. 8:00 AM to 5:00 PM E.D.T.
TOLL FREE: 1-800-321-5805

Guaranteed Shipment
Cleveland Consumer Computers & Components guarantees shipment of computer systems within 48 hours upon receipt of your order. **Our failure to ship within 48 hours entitles you to $35 of software, FREE.**

To Order: Or to get our free catalog **CALL 1-800-321-5805 TOLL FREE.** Charge your order to your **VISA** or **MASTER CHARGE** account. Ohio residents call: (216) 464-8047. Or write, including your check or money order, to the address listed below.

CLEVELAND CONSUMER COMPUTERS & COMPONENTS
P.O. Box 46627
Cleveland, Ohio 44146

Order Form: CLEVELAND CONSUMER COMPUTERS & COMPONENTS P.O. Box 46627 Cleveland, Ohio 44146

☐ Superboard II $299.
☐ 610 Board $298.
☐ Mini-Floppy Disk Drive $299.
☐ RF Modulator $35.
☐ AC-3P 12″ B/W Monitor $159.
☐ C1P Sams Manual $8.

(Attach separate sheet for other items.)

NAME ______________________

ADDRESS: ______________________

CITY: ______________ STATE: ______________ ZIP: __________

PHONE: ______________________

Payment by: VISA ______ MASTER CHARGE ______ MONEY ORDER ______

Credit Card Account #______________________

Expires ______________ Interbank #(Master Charge) ______________

TOTAL CHARGED OR ENCLOSED $__________ (Ohio Residents add 6.5% Sales Tax)

Orders of less than $100, please add $4.00 to cover shipping costs. Orders will be accepted from U.S. and Canada only. All prices quoted are U.S., date of publication, standard UPS shipping FOB the factory. C uses part of the original code as subroutines. There are several other fixes available, but one or the other must be used or you'll have to omit the names, and reset the string pointer at 129 and 130 (Dec) each time a loop of the Routine at Line 900 is run.

As is often the case, this program concept can be extended into many other areas. How about a checkbook balancer with the purpose of each check printed out along with its number and amount? A Christmas card checklist? The names can be any length consistent with your RAM; just change the "8" spaces wherever they appear. Routines to add values of numeric variables can be easily added.

Table 1

```
10 DIMN$(2):DATA"A","BBB"
20 X=0:Y1=3:Z=487
30 READN$(1):READN$(2)

0300 00  START SOURCE       033E 58  ASCII - X
         CODE                 3F 00   2ND CHAR
  01 17  ADDRESS OF           40 00  VALUE OF X
  02 03   NEXT LINE           41 00   IN 32-BIT
  03 0A  LINE                 42 00   FLOATING POINT
  04 00   NUMBER              43 00   BINARY
  05 85  TOKEN - DIM          44 59  ASCII - Y
  06 4E  ASCII - N            45 31        - 1
  07 2A        - $            46 82  VALUE OF Y1
  08 28        - (            47 40
  09 32        - 2            48 00
  0A 29        - )            49 00
  0B 3A        - :            4A 5A  ASCII - Z
  0C 83  TOKEN - DATA         4B 00   2ND CHAR
  0D 22  ASCII - "            4C 89  VALUE OF Z
  0E 41        - A            4D 73
  0F 22        - "            4E 80
  10 2C        - ,            4F 00
  11 22        - "            50 4E  ASCII - N
  12 42        - B            51 80  TOKEN - $TRING
  13 42        - B            52 13  7+(DIM+1)*4
  14 42        - B            53 00  ALWAYS 00
  15 22        - "            54 01  ALWAYS 01
  16 00  END OF LINE          55 00  ALWAYS 00
  17 2A  ADDRESS OF           56 03  DIM + 1
  18 03   NEXT LINE           57 00  FOUR BYTES TO
  19 14  LINE                 58 00   EACH ELEMENT OF
  1A 00   NUMBER              59 00   AN ARRAY; START
  1B 58  ASCII - X            5A 00   WITH ELEMENT 0
  1C AB  TOKEN - =            5B 01  LEN OF $TRING
  1D 30  ASCII - 0            5C 0E  ADDRESS OF
  1E 3A        - :            5D 03   $TRING
  1F 59        - Y            5E 00  ALWAYS 00
  20 31        - 1            5F 03  LEN OF $TRING
  21 AB  TOKEN - =            60 12  ADDRESS OF
  22 33  ASCII - 3            61 03   $TRING
  23 3A          :            62 00  ALWAYS 00
  24 5A          Z          0363 UNUSED RAM
  25 AB  TOKEN - =
  26 34  ASCII - 4
  27 38        - 8
  28 37        - 7
  29 00  END OF LINE
032A THROUGH 033B
  HOLDS LINE #30
033C 00  END OF LINE
  3D 00  CODE FOR
  3E 00  "LAST LINE"
```

Sample Run-Entire List

```
                JOHN'S INQUIRER ROUTE
                ------ -------- -----

1. STANFORD  2903 GEORGETOWN RD  DAILY & SUN  GARAGE
2. JONES        2 BARTON CT      DAILY ONLY   REAR DOOR
3. SMITH      321 BRANCH PIKE    DAILY & SUN  UNDER RUG
4.              0
5.              0
```

Listing 1A

```
0 DATA75
1 DATA"STANFORD",2903, 1,1,1
2 DATA"JONES   ",   2,12,2,3
3 DATA"SMITH   ", 321, 2,1,6
4 REM-LINES 4 THRU 73 OMITTED
74 DATA"        ",   0, 0,0,0
75 DATA"        ",   0, 0,0,0
```

Sample Run-Sunday Customers Only

```
                JOHN'S INQUIRER ROUTE
                ------ -------- -----

1. STANFORD  2903 GEORGETOWN RD  DAILY & SUN  GARAGE
3. SMITH      321 BRANCH PIKE    DAILY & SUN  UNDER RUG
```

Listing 1B

```
200 READX:N=X:DIMN$(X),A(X),S(X),D(X)
201 DIMP(X),S1$(15)
202 GOSUB700
205 FORX=1TO9:PRINT:NEXT
210 PRINT"Newspaper Route":PRINT
215 PRINT"1.  Print the route to printe
r":PRINT
220 PRINT"2.  Print the route to screen
":PRINT
225 PRINT"3.  Add a customer":PRINT
230 PRINT"4.  Delete a customer":PRINT
240 INPUT"Enter your preference";X2
245 ONX2GOSUB300,400,500,600
250 GOTO205
300 REM-PRINT TO PRINTER
310 POKE517,1
320 GOSUB420
330 POKE517,0
340 RETURN
400 REM-PRINT ROUTE TO SCREEN
405 PRINT:PRINT:PRINT:PRINT
410 PRINTTAB(15);"John's Inquirer Route
415 PRINTTAB(15);"------ -------- -----
420 PRINT:PRINT:FORX=1TON
425 REM-SELECT LIST ON THIS LINE
430 PRINTRIGHT$("  "+STR$(X),2);".";
435 PRINTTAB(4);N$(X);TAB(14);RIGHT$("
  "+STR$(A(X)),4);
440 PRINTTAB(19);S1$(S(X));
445 PRINTTAB(35);D1$(D(X));
450 PRINTTAB(48);P1$(P(X))
460 X=USR(X)
470 NEXTX
480 RETURN
500 REM
510 PRINT"           Add a customer":PRINT
515 INPUT"Enter customer's code number"
;X3:PRINT
520 FORX=NTOX3+1STEP-1
525 N$(X)=N$(X-1):A(X)=A(X-1):S(X)=S(X-
1):D(X)=D(X-1):P(X)=P(X-1)
530 X=USR(X):NEXTX
535 INPUT"Enter customer's name";N$(X3)
:PRINT
540 INPUT"Enter house number";A(X3):PRI
NT
545 INPUT"Enter street name code number
";S(X3):PRINT
550 INPUT"Enter delivery code number";D
(X3):PRINT
555 INPUT"Enter special location code n
umber";P(X3):PRINT
560 N$(X3)=LEFT$(N$(X3)+"        ",8)
565 PRINTN$(X3);A(X3);S1$(S(X3));D1$(D(
X3));P1$(P(X3))
575 X=USR(X)
580 INPUT"Add another";I$
585 IFLEFT$(I$,1)="Y"THEN500
590 GOTO800
600 REM-DELETE
610 PRINT:PRINT:PRINT:PRINT
620 INPUT"Enter customer's code number"
;X4
630 FORX=X4TON-1
640 N$(X)=N$(X+1):A(X)=A(X+1):S(X)=S(X+
1):D(X)=D(X+1):P(X)=P(X+1)
645 X=USR(X)
650 NEXTX
660 N$(N)="        "
670 A(N)=0:S(N)=0:D(N)=0:P(N)=0
680 INPUT"Delete another";I$
685 IFLEFT$(I$,1)="Y"THEN600
690 GOTO850
700 REM-STARTUP SEQUENCE
710 FORX=1TON
720 READN$(X):READA(X):READS(X):READD(X
):READP(X)
730 NEXTX
750 S1$(1)="Georgetown Rd":S1$(2)="Bran
ch Pike":S1$(3)="Essex Ct"
755 S1$(4)="Somerset Dr":S1$(5)="Bergen
Dr":S1$(6)="Salem Dr"
760 S1$(7)="Brigham Ct":S1$(8)="Salem C
t":S1$(9)="Cooper Ct"
765 S1$(10)="Hunterdon Dr":S1$(11)="Riv
erton Rd":S1$(12)="Barton Ct"
768 S1$(13)="Carriage Way":S1$(14)="Mid
dlesex Dr"
775 D1$(1)="Daily & Sun":D1$(2)="Daily
only":D1$(3)="Sun only"
780 D1$(4)="Sat only":D1$(5)="M-F only"
:D1$(6)="Sat-Sun only"
790 P1$(1)="Garage":P1$(2)="Carport":P1
$(3)="Rear door"
795 P1$(4)="In door":P1$(5)="Mail box":
P1$(6)="Under rug"
799 RETURN
800 REM-SAVE DATA - ADD CUSTOMERS
805 B=782
810 FORX=NTO1STEP-1
815 B=782+(X-1)*28
820 GOSUB900:X=USR(X)
825 NEXTX
830 CLEAR:GOTO200
850 REM-SAVE DATA - DELETE CUSTOMERS
855 B=782
860 FORX=1TON
865 GOSUB900
870 X=USR(X)
875 B=B+6:NEXTX
880 CLEAR:GOTO200
900 REM-SAVE DATA
905 L$=LEFT$(N$(X)+"        ",8)
910 Q=8:POKEB,34:B=B+1:GOSUB995:POKEB,3
4:B=B+1
915 B=B+1:L$=RIGHT$("    "+STR$(A(X)),4
)
920 Q=4:GOSUB995
925 B=B+1:L$=RIGHT$("  "+STR$(S(X)),2)
930 Q=2:GOSUB995
935 B=B+1:L$=RIGHT$(" "+STR$(D(X)),1)
940 Q=1:GOSUB995
945 B=B+1:L$=RIGHT$(" "+STR$(P(X)),1)
950 Q=1:GOSUB995
955 RETURN
995 FORR=1TOQ:POKEB,ASC(MID$(L$,R,1)):B
=B+1:NEXTR:RETURN
999 END
OK
```

Listing 3

```
41000 REM-GARBAGE COLLECTION FIX
```

```
41010 POKE11,34:POKE12,2
41020 FORX=0TO139
41030 Y=PEEK(45383+X):POKE546+X,Y
41040 NEXTX
41050 FORX=0TO46
41060 Y=PEEK(45596+X):POKE696+X,Y
41070 NEXTX
41080 POKE613,4:POKE699,2:POKE700,24
41090 POKE629,177:POKE630,2
41100 POKE686,76:POKE687,211:POKE688,177
41110 POKE689,166:POKE690,157
41120 POKE691,208:POKE692,3
41130 POKE693,76:POKE694,19:POKE695,178
41140 POKE743,38:POKE744,2
OK
```

©

Voracious Butterfly

John Wright
Ottawa, Canada

The name came after seeing the program run. Voracious Butterfly was originally just a first exercise in using peeks, pokes and graphics, and as a visual check on how random is RND.

Display

A 24 x 24 section of the screen is filled with G187, the mini chequerboard, and G43, +, moves around one square at a time in a random direction. Each time it lands on a new square it 'eats' the G187 and replaces it with a G32 (Blank), G42 (*) or a character. When all the characters are displayed, the end routine pokes in another word and strips out the remaining G187s. A counter at the bottom of the screen increments by 100 every 100 steps.

Program

The program has 6 modules and a main line. The subroutines at 300, 500 and 1100 are called once each and could have been written in the main line. Conversely L70 to L150 could have been another module.

SUB 250 converts from X, Y coordinates to a POKE address.

SUB 300 to 480 reads in the word which is used in the end routine, puts 32 in all locations of the MA matrix to POKE blanks, replaces some of those 32s by 42s to sprinkle stars in the top and bottom thirds of the screen, and zeros counters.

SUB 500 to 560 reads character data into MA and puts a 1 in MB corresponding to each character in MA.

L70 to 155 picks the start point for the Butterfly and POKEs two zeros for the counter.

SUB 800 to 960 takes 100 steps. On each step the contents of MA are poked to the screen location, the contents of MB are added to TT (MB is 0 unless there is a display character in which case MB (X,Y) is 1. It is then reset to 0).

L840 checks conditions for a normal exit.

L860, 870 give the next step in the walk, with equal probability of staying still or moving to any of the eight adjacent squares.

L900, 910 stop the Butterfly from going off screen. Using SGN allows it to be done with one statement instead of separate IFs for 0 and 25. If the Butterfly goes off left, it reappears right as though there is a wrap-around. Similarly for top and bottom.

SUB 1000 to 1090 adjusts the base of the random number by incrementing the original input. This

OSI **AARDVARK** OSI

NOW MEANS BUSINESS!

WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

This is a line-oriented word processor designed for the office that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any text single, double or triple spaced and has text centering commands. It will make any number of multiple copies or chain files together to print an entire disk of data at one time.

MAXI-PROS has both global and line edit capability and the polled keyboard versions contain a corrected keyboard routine that make the OSI keyboard decode as a standard typewriter keyboard.

MAXI-PROS also has sophisticated file capabilities. It can access a file for names and addresses, stop for inputs, and print form letters. It has file merging capabilities so that it can store and combine paragraphs and pages in any order.

Best of all, it is in BASIC (OS65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.
MAXI-PROS — $39.95

THE EDSON PACK
ALL MACHINE CODE GAMES
FOR THE 8K C1P

INTERCEPTOR —You man a fast interceptor protecting your cities from Hordes of Yukky Invaders. A pair of automatic cannon help out, but the action speeds up with each incoming wave. It's action, action everywhere. Lots of excitement! $14.95

MONSTER MAZE — An Arcade style action game where you run a maze devouring monsters as you go. If one sees you first, you become lunch meat. Easy enough for the kids to learn, and challenging enough to keep daddy happy. $12.95

COLLIDE — Fast-paced lane-switching excitement as you pick up points avoiding the jam car. If you succeed, we'll add more cars. The assembler code provides fast graphics and smooth action. $9.95

SPECIAL DEAL—THE ENTIRE EDSON PACK— ALL THREE GAMES FOR $29.95

THE AARDVARK JOURNAL

FOR OSI USERS — This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modify the system, and reviews of OSI related products. In the last two years we have run articles like these!

1) A tutorial on Machine Code for BASIC programmers.

2) Complete listings of two word processors for BASIC IN ROM machines.

3) Moving the Directory off track 12.

4) Listings for 20 game programs for the OSI.

5) How to write high speed BASIC — and lots more —

Vol. 1 (1980) 6 back issues - $9.00
Vol. 2 (1981) 2 back issues and subscription for 4 additional issues - $9.00.

ACCOUNTS RECEIVABLE — This program will handle up to 420 open accounts. It will age accounts, print invoices (including payment reminders) and give account totals. It can add automatic interest charges and warnings on late accounts, and can automatically provide and calculate volume discounts.

24K and OS65D required, dual disks recommended. Specify system.
Accounts Receivable. $99.95

*** * * SPECIAL DEAL — NO LESS! * * ***

A complete business package for OSI small systems — (C1, C2, C4 or C8). Includes MAXI-PROS, GENERAL LEDGER, INVENTORY, PAYROLL AND ACCOUNTS RECEIVABLE — ALL THE PROGRAMS THE SMALL BUSINESS MAN NEEDS. $299.95

P.S. We're so confident of the quality of these programs that the documentation contains the programmer's home phone number!

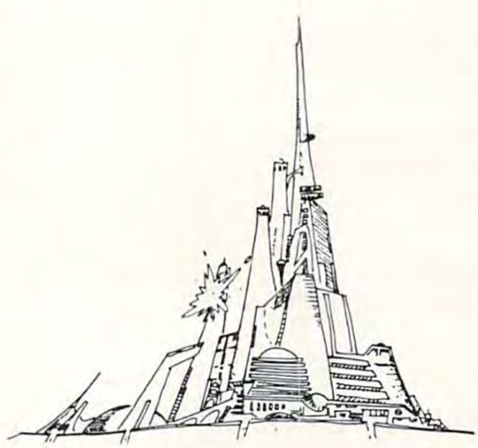
SUPERDISK II

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs — and sell the whole thing for — SUPERDISK II $29.95 (5 1/4") $34.95 (8").

AND FUN, TOO!



BOOKKEEPING THE EASY WAY — WITH BUSINESS I

Our business package 1 is a set of programs designed for the small businessman who does not have and does not need a full time accountant on his payroll.

This package is built around a **GENERAL LEDGER** program which records all transactions and which provides monthly, quarterly, annual, and year-to-date PROFIT AND LOSS statements. GENERAL LEDGER also provides for cash account balancing, provides a BALANCE SHEET and has modules for DEPRECIATION and LOAN ACCOUNT computation.
GENERAL LEDGER (and MODULES) $129.95.

PAYROLL is designed to interface with the GENERAL LEDGER. It will handle annual records on 30 employees with as many as 6 deductions per employee.
PAYROLL - $49.95.

INVENTORY is also designed to interface with the general ledger. This one will provide instant information on suppliers, initial cost and current value of your inventory. It also keeps track of the order points and date of last shipment.
INVENTORY - $59.95.

GAMES FOR ALL SYSTEMS

GALAXIAN - 4K - One of the fastest and finest arcade games ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. Specify system — A bargain at $9.95

MINOS - 8K - — Features amazing 3D graphics. You see a maze from the top, the screen blanks, and when it clears, you are in the maze at ground level finding your way through on foot. Realistic enough to cause claustrophobia. — $12.95

NEW — NEW — NEW

LABYRINTH - 8K - This has a display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you track down and shoot mobile monsters on foot. Checking out and testing this one was the most fun I've had in years! — $13.95.

TIME TREK - 8K - Real Time and Real graphics Trek. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! — $9.95

SUPPORT ROMS FOR BASIC IN ROM MACHINES — C1S/C2S. This ROM adds line edit functions, software selectable scroll windows, bell support, choice of OSI or standard keyboard routines, two callable screen clears, and software support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. C1P requires only a jumper change. — $39.95
C1E/C2E similar to above but with extended machine code monitor. — $59.95

Please specify system on all orders

This is only a partial listing of what we have to offer. We now offer over 100 programs, data sheets, ROMS, and boards for OSI systems. Our $1.00 catalog lists it all and contains free program listings and programming hints to boot.

OSI

AARDVARK TECHNICAL SERVICES, LTD.
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

OSI

was done to disturb any patterns. It also sets up an exit if the program runs too long and the Butterfly tires, and increments the display counter.

SUB 1100 to 1180 is the end routine. It POKES the top word and clears out the remaining G187s.

L170, 470, 530 and 840 could be changed to the handwritten version to make the display part easier to change.

Changing the Randomness

The original version did not have the routine at L1000, the Q loop at L800 and instead of wrap-around at L900 and 910 it had fold-back. This can be simulated by:

```
1030 REM DUMMY LINE
 900 IF X=0 OR X=25 THEN X=X-SX
 910 IF Y=0 OR Y=25 THEN Y=Y-SY
```

This version did not always work. On one occasion it ran for about 35 minutes and left two sizeable areas of the screen untouched.

Presumably the random number generator settled into a pattern, so to disturb this the 1000 routine was introduced, changing the base after 100 steps.

Variations

The listing includes many REM statements which can be omitted, and most statements are 1 to a line so compaction is possible.

Apart from 'wrap-around X and fold-back Y' (which I have left in my archive version) or vice-versa, the variations are in the display capability.

If you are on familiar terms with him,

```
 300 IB = 5
3000 82,79,78,78,89
```

may be acceptable.

If you change the main display data, you should use the handwritten version of L170, 470, 530 and 840 or recalculate. Remember that the display goes in bottom first, top last.

There is no reason why the display should not be a picture or a pattern. In this case the 'top' word may be better placed at the bottom by:

```
1100 Y = 1
```

It would be easier to have the display data as characters and blanks. They would then be read in by string variables and converted.

```
  45  BL$ =" 24 spaces "
 300  READ A$
 305  FOR I=I TO LEN(A$):TW(I)=ASC(MID$
      (A$,I,1)):NEXT I
 505  READ A$:A$=LEFT$(A$+BL$,24)
 520  MA(X,Y)=ASC(MID$(A$,X,1))
3010  DATA " ABC etc.
```

Use the " in the data statement if there are leading blanks.

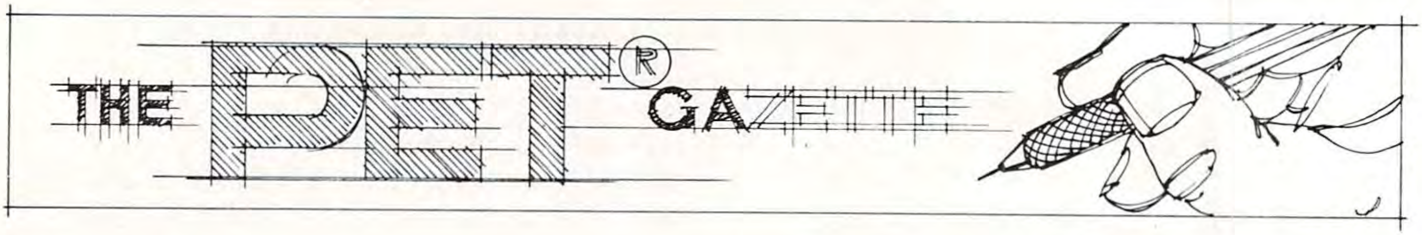
Side Benefits

The program is a good conditioner for a flabby waist. Judicious use of body English may guide the Butterfly to uneaten squares.

```
1 REM  VORACIOUS BUTTERFLY
10 REM  JOHN WRIGHT
15 DIM MA(24,24),MB(24,24)
20 REM  'RANDOM WALK' DEMONSTRATION
30 INPUT"RANDOM NUMBER";NR:RN=NR
40 OO=54116:REM THIS IS FOR 600 BOARD
50 GOSUB300
60 GOSUB500
70 REM  PICK START POINT
80 J=INT(570*RND(RN))+1
90 X=INT(J/24)+1
100 Y=J-24*(X-1)
110 GOSUB250
120 POKEZ,43
140 REM  PAUSE THEN REMOVE +
150 FOR J=1TO100:I=J:NEXTJ
155 POKEOO+6,48:POKEOO+7,48
160 GOSUB800
170 IFTT=61THEN200     IF TT=CH...
180 GOSUB1000
190 IFCT<>-1THEN160
200 GOSUB1100
210 IFCT=-1THENPRINT"EXHAUSTED BUTTERFLY"
220 END
247 REM
248 REM
249 REM  SCREEN POSITION FOR X,Y
250 Z=OO-32*Y+X
260 RETURN
296 REM
297 REM
298 REM  INPUT DISPLAY MATRIX
299 REM  TOP WORD
300 IB=6
305 FORI=1TOIB:READTW(I):NEXTI
310 REM  BLANK SCREEN,INPUT DISPLAY, SET UP
    COUNTER MATRIX
315 FORX=1TO30:PRINT:NEXTX
317 POKEOO+1,32:REM  ERASE CURSOR
320 FORX=1TO24
330 FORZ1=1TO3
340 FORZ2=1TO8
350 Y=8*(Z1-1)+Z2
360 GOSUB250
370 POKEZ,187
380 MA(X,Y)=32
390 REM  STARS AT TOP AND BOTTOM
400 IFZ1=2THEN420
410 IFRND(RN)<.15THENMA(X,Y)=42
420 MB(X,Y)=0
430 NEXTZ2,Z1,X
460 REM  ZERO COUNTERS
470 TT=0:CT=0 : CH=0
```

```
480 RETURN
497 REM
498 REM
499 REM READ IN DATA
500 FOR Y=11TO15
510 FORX=1TO23
520 READMA(X,Y)
530 IFMA(X,Y)<>32 THEN MB(X,Y)=1   1:CH=CH+1
540 NEXTX,Y
560 RETURN
797 REM
798 REM
799 REM  100 STEPS IN RANDOM WALK
800 FOR Q=1TO100
810 TT=TT+MB(X,Y)
820 MB(X,Y)=0
830 POKEZ,MA(X,Y)
840 IFTT=61THEN960   IF TT=CH...
850 REM  NEW PLACE FOR +
860 SX=INT(3*RND(RN))-1:REM  GIVES+1,0,-1
870 SY=INT(3*RND(RN))-1:REM  DITTO
880 X=X+SX
890 Y=Y+SY
900 IFX=0ORX=25THENX=X-24*SGN(SX):REM  TRY
    ...THENX=X-SX
910 IFY=0ORY=25THENY=Y-24*SGN(SY):REM  TRY
    ...THENY=Y-SY
920 GOSUB250
930 POKEZ,43
950 NEXTQ
960 RETURN
997 REM
998 REM
999 REM  ADJUST BASE OF RANDOM NUMBER
1000 CT=CT+1
1010 IFCT=100THENCT=-1:GOTO1070
1020 NR=NR+1
1030 RN=NR
1040 AC=INT(CT/10)+48
1050 IF AC=48THEN1070
1060 POKE00+4,AC
1070 AC=CT-10*AC+528
1080 POKE00+5,AC
1090 RETURN
1097 REM
1098 REM
1099 REM  END ROUTINE
1100 Y=17
1110 FORX=1TO18
1120 GOSUB 250
1130 POKEZ,TW(X)
1140 NEXTX
1150 FORZ=00-776TO00
1160 IFPEEK(Z)=187THENPOKEZ,32
1170 NEXTZ
1180 RETURN
3000 DATA 82,79,78,65,76,68
3010 DATA 82,32,82,32,69,69,69,32,65,32,65,32
3020 DATA 32,71,71,32,65,32,65,32,78,32,78
3030 DATA 82,32,82,32,69,32,32,32,65,65,65,32
3040 DATA 71,32,71,32,65,65,65,32,78,78,78
3050 DATA 82,82,32,32,69,69,32,32,65,32,65,32
3060 DATA 71,32,32,32,65,32,65,32,78,78,78
3070 DATA 82,32,82,32,69,32,32,32,65,32,65,32
3080 DATA 71,32,71,32,65,32,65,32,78,78,78
3090 DATA 82,82,32,32,69,69,69,32,32,65,32,32
3100 DATA 32,71,32,32,32,65,32,32,78,32,78
OK
```

©



Saving Machine Language Programs On PET Tape Headers

Louis F. Sander
Pittsburgh, PA

This article describes a simple method of using your old ROM PET to write a brief machine language program (MLP) onto the *header* of any PET program tape, where it will automatically LOAD right along with the other program on the tape. The method described is the only way we know to use the first cassette buffer with a program loaded from TAPE #1, short of putting a program in there *after* the LOAD. The only restriction on the MLP is that it cannot exceed 171 bytes in length, but there's a way to expand this limit to 187.

After loading, the MLP will reside in the 1st cassette buffer, where it will not restrict the use of the other buffer or the user program area. Of course the MLP will be removed from memory, but not from the tape header, if the 1st cassette is subsequently used to LOAD, SAVE, or VERIFY, since those actions write into the 1st cassette buffer. For the same reasons, the MLP cannot be SAVEd again itself using normal procedures.

I have used this method to couple a joystick handler to a BASIC game program, and to load a tiny machine language monitor along with a program under development. It could easily be used in a program protection system, and the creative programmer can no doubt find many other uses for it. To see this wonder in action, first write a machine language program that is no longer than 171 bytes, and which will ultimately reside in locations 028F-0339 hex, (655-825 decimal), of your old ROM PET. Be careful, because you are writing a program in an area of memory that is wiped out whenever TAPE #1 is used. When you are ready to put your MLP onto the header of a program tape, use a machine language monitor, BASIC POKEs, or any other method to perform the steps below:

1. Put your MLP temporarily into locations starting at 034F, (847). If you run past location 03F9, (1017 decimal), your MLP is too long.

2. Put ASCII spaces into all memory locations between the end of your MLP and loaction 03F9, (1017), inclusive. The space character is a 20 hex, or a 32 decimal.

3. Using the appropriate ASCII codes for the name of the main program to be SAVEd, 16 characters maximum, put the program name into locations 033F-034E, (831-846 decimal). Fill any unused locations in this range with ASCII spaces, just as in step 2. Note that the name here is *not* the name of your new little MLP, but the name of the main program you'll be saving.

4. Using normal procedures, LOAD or key in the main program to be saved. You can change the program once it has loaded, for instance if you want to add some SYS calls to access your little MLP.

5. If your main program is in machine language, this step is required *unless* the main program was loaded from tape and still has the same starting and ending addresses: Put the main program's starting address, in lo byte, hi byte order, into 00F7-00F8 hex, and put its ending address plus one, in the same order, into 00E5-00E6.

6. In direct mode, POKE 249,63 : POKE 250,3 : POKE 238,187

7. PEEK to see that your POKES were successful and accurate.

8. Get the cassette you are going to SAVE onto, and put it into TAPE #1.

9. In direct mode, type SYS 63135 if your main program is in BASIC, or SYS 63153 if your main program is in machine language. When you hit RETURN, PET should initiate a SAVE, with the normal messages, but with some additional garbage after the program name. The garbage is your little MLP, and this is the last time you'll ever see it in this form.

10. VERIFY what you saved.

11. You can now LOAD and use this tape in the normal way. (Momentarily power down if you

NEW - LOW COST FLOPPY DISK FOR PET*!

PEDISK II

WHY PEDISK?

FAST — The *Pedisk* system loads directly to main memory from the disk and loads — saves information at least 3 times faster than any other disk available.

SIMPLE — A simple command syntax makes its BASIC commands easy to use. At the same time, its sophisticated indexed, sequential, or relative file handling makes it powerful.

RELIABLE — The ultra-simple circuit design with its LSI disk controller chip provides maximum timing margins for error free operation.

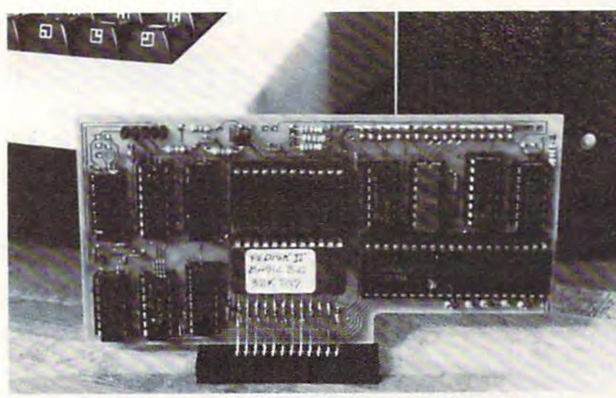
5¼" system — 1 drive, double density (143K) $595.00
5¼" system — 1 drive, quad density (286K) $795.00
8" system — 1 drive, IBM 3740 business (295K) ... $1,295.00

2 drives, double density (286K) $945.00
2 drives, quad density (572K) $1,195.00
2 drives, IBM 3740 business (590K) ... $1,895.00

Introducing **PEDISK II**, a low cost high performance floppy disk system consisting of: 1) *PEDISK II* controller, 2) flat disk cable, 3) Disk Drive Assembly, and 4) PDOS II software. The *PEDISK II* is a full function peripheral that provides high speed program storage and a sophisticated file handling package.

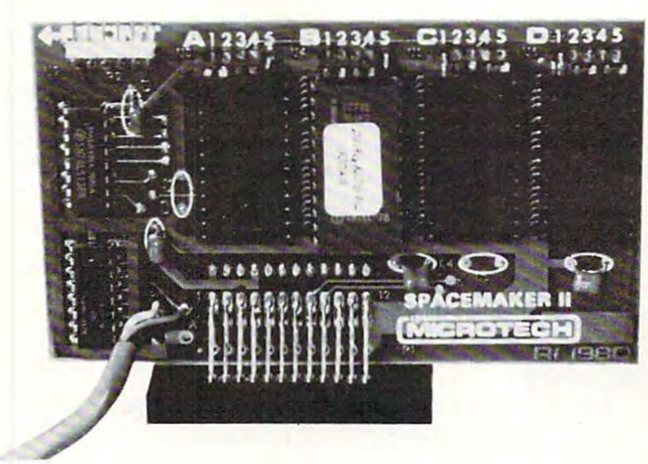
The small *PEDISK II* controller board mounts inside any 2000, 4000, or 8000 series machine. Standard systems are available with one or two disk drives. An optional third drive can also be added. The *PEDISK II* System can be operated simultaneously with any Commodore disk system for data exchange purposes.

The PDOS II software links BASIC by adding a new repertoire of disk commands. !OPEN, !CLOSE, !INPUT, and !PRINT provide the basis of the powerful file handling package. !LOAD, !SAVE and !RUN allow complete disk control. PDOS II also offers a full DOS-mode of operation for all disk diagnostic and utility functions. Diskette format, backup, diagnostic, and reorganization capabilities are provided.



PEDISK II CONTROLLER BOARD

NEW - SPACEMAKER II switch between one of FOUR ROMS



SPACEMAKER II is the new ROM switch from Microtech. It allows either manual or software controlled switching of up to four ROMs in a single ROM expansion socket. The switching is accomplished with a side-mounted slide switch or via ROMDRIVER, an accessory board which allows software controlled switching and keyboard controlled switching. ROM I/O is a special software package available on disk to implement full keyboard control of the ROMs. In addition, it adjusts for memory differences found in various utility ROMs.

SPACEMAKER II $39.00
ROMDRIVER $39.00
ROM I/O Commodore or Pedisk .... $ 9.95

SEE YOUR LOCAL DEALER FOR ALL cgrs MICROTECH PRODUCTS

cgrs MICROTECH P.O. BOX 102, LANGHORNE, PA. 19047

*PET IS A TRADEMARK OF COMMODORE

want to try it now). Every time you LOAD it, your MLP will be sitting at 028F hex, where it can be called from BASIC by a simple SYS 655.

Neat, isn't it? Here's how it works. When a program SAVE is initiated, PET makes up a 192-byte header to record ahead of the program. The header contains the following data, from start to finish:

- One byte, a 01, which identifies the tape as a program, not a data file.
- Two bytes showing the first memory location the program is to be stored in.
- Two bytes showing the last memory location the program will take up, plus one.
- Sixteen bytes of program name that will print onto the screen during searches.
- Another 171 bytes of program name that PET will look at and compare to while searching, but which won't usually print onto the screen.

If the program name is less than 16 + 171, or 187 characters, which most of them are, PET fills out the 187 bytes with spaces.

To make up the header, PET has to locate the program name, which is stored somewhere in memory. Locations 249 and 250 decimal specify the location of the first character of the program name, and location 238 specifies its length. Normally, the contents of these locations are set during the SAVE dialogue between you and the screen, but our three POKES circumvent this. They tell PET that a 187 character "program name" will be found beginning at 033F, (831 decimal). Our previous steps have set this bogus "program name" to a sixteen-character *real* program name including trailing spaces, plus our MLP, plus the expected trailing spaces in unused bytes. We had to put our 187 bytes outside the 1st cassette buffer, because the SAVE in Step 9 clears that buffer out.

Our SYS 631xx tells PET to start recording the header, including the bogus program name, followed by the main program. Little does PET know or care that the final 171 characters in its "program name" include our MLP. (The extra steps and different SYS for the machine language main program are required because PET SAVEs BASIC and Machine Language in slightly different ways.)

When our specially prepared tape is LOADed, all the bytes from the header are deposited in the 1st cassette buffer. PET uses the first five bytes for directions on where to store the main program. It routes the next 16 bytes to the screen to tell us what program it FOUND, and it all but ignores the rest of the bytes that were read in from the header. But when we later call on them as a MLP, PET can execute them just like any other machine language code.

That's all there is to it. But what if we need a longer MLP? Where there's a will, there's a way. In the earlier detailed instructions, we limited ourselves to a 171-byte MLP. By using the method described, we retained the nicety of a 16 character *real* program name. If we are willing to give up

Little does PET know or care that the final 171 characters in its "program name" include our machine language program.

some or all of these characters, we can start our temporary storage area a little lower in the second cassette buffer, change the value of our POKEs and the SYS we use to call our MLP, and put up to 187 bytes of MLP on that header. The first 16 characters will still print on the screen, although since they make up part of a MLP rather than a "name", they'll print some strange characters. The basic idea is given here; the calculations are left up to you, as is finding the way to use this whole method with the *second* cassette buffer. ©

Commodore ROM Systems: Terminology

Jim Butterfield
Toronto, Canada

The first PET ROMs didn't seem to have a name. They were just the PET ROMs. Users perceived different types of ROM sets: some had 28 pins each and were manufactured by MOS Technology, a Commodore subsidiary; others had 24 pins and were made by outside suppliers. Then there was a bug replacement: early ROMs numbered either 6540-011 or 901447-01 were changed for corrected ROM systems.

This first ROM system was internally called Basic 1.0 by Commodore. Unfortunately, the Basic language itself is called Basic Level 2.

A year later, another ROM set arrived which once again had no name. Dealers and users just called them "new ROMs" since that's what they were at the time. Unfortunately, new ROMs aren't new any more, and calling them by that name confuses newcomers.

This second system was called Basic 2.0 by Commodore. And naturally, the Basic was called Basic Level 3. The numbers were generally used only within Commodore so that users weren't exposed to the confusion. How the Commodore people sorted it out, I can't guess.

Finally, Basic 4.0 was released, and at this point the two number systems caught up to each other. The machine prints COMMODORE BASIC 4.0 upon power-up, so that everyone knows what number belongs to this ROM set. Finally.

What happened to the missing number? Nobody seems to know. My theory is this: that Commodore internally called their first Basic "1.0 ROMs" for the obvious reason. In the meantime, Commodore sales may have become concerned by comparisons to Radio Shack Level I Basic (a fairly primitive version of Basic) and decided to start their numbering at Level II. At that point, I surmise, they were stuck, and the production 2.0 ROMs had to be called Level 3. during this period of time, the number you got depended on the department within Commodore that you were talking to. Finally, the production people must have decided that if they skipped a number everybody would be caught up. They were right: we all agree on what Basic 4.0 is, even though we're still cloudy on which one should be called Basic 2...

A proposed standard.

Let us call the first ROM, whatever its number: Original ROM. That's the ROM without a Machine Language Monitor; that can't handle IEEE disk; that has a limit of 256 elements in an array, and that has some minor tape irregularities. Many users will have upgraded their systems, but there are still a lot of the Original ROMs around, and writers for **COMPUTE!** should be specific if their programs will work only on the more recent machines.

The next ROM should be called Upgrade ROM. That's the ROM with a Machine Language Monitor and other improvements, but without the speeded garbage collection routines and without the extra disk commands such as DLOAD or SCRATCH.

Subsequent Basic ROMs may be called by their number: at the moment, Basic 4.0 is the only one out, but there's a 5.0 rumored for the near future.

Within these various styles of Basic, there are a

We all agree on what Basic 4.0 is, even though we're still cloudy on which one should be called Basic 2...

few small variations which may or may not be significant to the reader. Some machines have graphic keyboards and others have full ASCII keyboards; there are minor changes in ROMs to accomodate the particular keyboard used. More significantly, an article which asks a user to press the TAB key may not be too helpful to a user who doesn't have such a key on his computer.

Similarly, there's a visible difference between 40-column and 80-column PET/CBM computers, and there are small ROM differences between the two types of machine. Some programs will run splendidly on both types of machine: but if your program doesn't, it's a good idea to specify the machine for which the program is written.

Hardware Differences

The ROM set doesn't always correspond exactly with the physical configuration of the machine. Original small-keyboard PETs can be fitted with upgrade ROM ... and can even be fitted with large keyboards, which makes them hard to recognize. Similarly, a green screen isn't always a guarantee that the machine is of more recent vintage; and subsequent PETs have gone through more than one board redesign.

The most significant hardware change seems

to be from the original PET — which can be spotted by its memory expansion edge connector, and by its screen noise when a screen POKE is performed. Subsequent models have expansion pins (not an edge connector) and a hash-free screen.

There are other architectural differences, the wide 80-column screen being the most obvious. Some of the changes are important in a negative way: for example, the "speedup" achieved on some computers by poking address 59458 is to be avoided since it can cause chip damage on certain models.

Generally, programs which behave differently in different hardware (as opposed to different ROM sets) are unusual. If there's a good reason for such behaviour, it should be documented ... the more explanations, the better.

...if (your program's) going to go elsewhere, try to spot any signs of travel sickness in advance.

Disk ROMs

We seem to have more consistency in disk ROM systems. The first 2040 ROM is usually called DOS 1.0; units fitted with it require initialization before a diskette can be used. The subsequent ROMs for the 2040 are DOS 2.0; they can be quickly spotted by the fact that the unit bumps the heads at time of power-up. The ROMs currently fitted to the Model 8050 disk are called DOS 2.5. At the present time, the 8050 has only one ROM set, but others may emerge. When this happens, it may be that most programs will work well on either ROM; but of course, any variant behaviour should be noted.

Programs intended for DOS 1.0 systems must initialize a diskette before starting to work on files; the assumption is that the data files are likely to be on a different disk from the program itself. If you are writing a program for your own DOS 2.0 or 8050 system, it's a good idea to remember to include such an initialization even though your system doesn't need it. Others may find it useful.

If you are working with REL type files, it's fairly obvious that DOS 1.0 systems are not compatible with your program. Note it anyway in your text; what's obvious to you may not be to newcomers.

It seems to be safe to assume that a 2.0 DOS, which has Append and Relative file features, will always be used with a 4.0 or later Basic. Don't be too sure. There are plenty of users who don't have the option of going to Basic 4.0. If you write your program for Basic 4.0 — and it's usually easiest for you to do it this way — note the fact. The Basic 2.0 user may want to take a shot at converting your program, but he needs to be warned that conversion is necessary.

Machine Language

It's possible to write machine language programs that will run on any PET/CBM machine. It can't be done every time, but if you can ... do it. I find it a great nuisance to have to keep copies of programs suitable for Original, Upgrade and 4.0 Basic ROMs. The trick is to use the Jump Table (Hexadecimal FFC0 to FFEA) for all input and output; it's identical in all PET ROM systems so that one program fits all machines. To repeat: you can't make your machine language program ROM-independent in every possible case; but it can be done surprisingly often.

If your program runs only on a particular machine type, document it. The same ground rules apply: if the user is cautioned, he may well take a shot at doing the conversion himself. If he's not warned, he may spend a lot of time typing in the program only to find that it doesn't work. Then he may spend hours looking for a transcription error that isn't there.

Summary

If you're writing a program that you think may be used on somebody else's machine, look it over carefully for compatibility problems. It doesn't matter whether you plan to sell the program, give it to a friend, or publish it in **COMPUTE!** — if it's going to go elsewhere, try to spot any signs of travel sickness in advance.

Of course, you don't have every model of PET and every printer and disk unit in your home. Obviously, you can't test everything yourself.

But learn a little about compatibility between machines, and you'll know where to look for potential trouble. If you're not sure, try the program on a friend's machine. If you're still not sure, add some cautionary sentences to your documentation.

PET/CBM machines are really highly compatible. Learn how to look for the few rough spots and your programs will become much more widely useable.

And if you're really not sure, appeal for help. Drop a couple of REM statements in your program asking for feedback. Many users will be glad to help ... to tell you what they needed to do to fix your program for their machine. Or better yet ... to tell you that your program worked fine on the first try. ©


HAL
PCG 6500

Screener

Four Screen Utility Routines For The PET

Ralph Owens

The PET's cursor controls, TAB, SPC, and clear screen functions are good, but sometimes they are not good enough. Several times I have wished for a clear, but reversed, screen. At times I have longed for a scroll-right feature for some neat, action-packed cartoons.

Finally I decided to quit wishing and to write some screen routines. Listing 1 shows the results of my frustrations. These are four short, independent, relocatable, re-entrant, ROM-independent screen routines which use the 1K block of screen memory as data for a block action of some type. Each routine may be implemented alone, or all four may be placed in the PET at one time. Each routine uses one base address for indirect addressing. This is placed into addresses 0-2 which have (in all released ROM's) been reserved for the USR function vector, and are infrequently-used addresses.

The first routine moves everything on the screen one character to the right. It starts in the upper left corner and moves the entire row right. It repeats itself on the next line for 25 times. By changing the starting address, the number of columns to move (in register X) and the number of lines to move (in register Y), one can make any portion of the screen roll right.

The second routine reverses everything on the screen. The trick here is that it toggles the 7th bit of the screen character, which causes the entire screen to be the reverse of what it was. The method used does not allow easy modification to work on only a certain portion of the screen.

The third routine clears a window on the screen. The base address as shown in the listing is the memory address of the upper left corner of the window. The X register contains the height of the window. The Y register contains the width of the window. In listing 1, lines 460 to 490 show how to dynamically change the position of the window. If you change the blank to a reversed blank (160 instead of 32) then a reversed window will appear. If you also set the window to include the entire screen, then a clear, but reversed, screen will result.

The fourth routine is a scroll down routine. It moves everything on the screen down one line. By changing the value in the X register, and the value of the counter at location zero, one can get only a portion of the screen to scroll down.

Putting the two scrolling functions inside of FOR-NEXT loops will provide quite good animation. If you put the reverse screen routine in a FOR-NEXT loop, then a flashing, attention-getting display can be obtained.

These four routines are just some things that I've wanted to do with the screen. I hope that you can use them. Some other ideas that I have had are to make a scroll left function. One could grab the

```
100 PRINT"{CLR}THIS PROGRAM WILL LOAD FOUR SHORT,"
110 PRINT"RELOCATABLE, ROM INDEPENDANT, MACHINE"
120 PRINT"LANGUAGE ROUTINES INTO THE SECOND"
130 PRINT"CASSETTE BUFFER."
140 PRINT"{DOWN}THE FIRST ROUTINE WILL MOVE EVERY"
150 PRINT"CHARACTER ON THE SCREEN ONE SPACE TO"
160 PRINT"THE RIGHT."
170 PRINT"{DOWN}THE SECOND ROUTINE WILL REVERSE EVERY"
180 PRINT"CHARACTER ON THE SCREEN."
190 PRINT"{DOWN}THE THIRD ROUTINE WILL CLEAR ANY"
200 PRINT"SPECIFIED BLOCK OF THE SCREEN."
210 PRINT"{DOWN} THE FOURTH WILL SCROLL THE SCREEN"
220 PRINT"DOWN ONE CHARACHTER"
230 GOSUB2020
240 PRINT"{CLR}DO YOU WANT A DEMONSTRATION? (Y OR N)"
250 GETCH$:IFCH$=""THEN250
260 IFCH$<>"Y"THENPRINT"OH COME ON, HUMOR ME!!"
270 PRINT"{DOWN}{DOWN}{DOWN}FIRST I WILL DEMONSTRATE THE SCROLL RIGHT FUNCTION"
280 PRINT"{DOWN}{DOWN}{DOWN}{DOWN}{DOWN}{DOWN}{DOWN}{RIGHT}BY{DOWN} SHOWING{DOWN} HOW{DOWN} IT{DOWN} MOVES{DOWN} THE {DOWN}WHOLE{DOWN} SCREEN"
290 FORN=826TO965:READA:POKEN,A:NEXT
300
310 REM PUT ROUTINES INTO THE 2ND CASSETTE BUFFER FOR DEMO PURPOSES ONLY
320 REM THEY CAN BE RELOCATED ANYWHERE THAT IS SAFE
330
340 FORN=1TO40:SYS(826):FORM=1TO400-10*N:NEXTM,N
350 PRINT"{DOWN}{DOWN}NOTICE HOW THE SPEED INCREASED AS THE"
```

IRQ vector and view each character before the operating system gets it. Then when the top of the screen is reached, the scroll down function could be called automatically. This could also be done with the left and right scroll routines. If the screen buffer were expanded (under software) then true scrolling into and out of the non-visible portion of the buffer could be implemented. This could help word processing, some search-and-destroy games, and some advertising and attention-getting routines. Additionally, one could implement the cleared reverse by reading the reverse flag in the operating system and shunting to the reverse routine. I am sure that you can think of many other creative ideas which could be accomplished using similar techniques.

For those of you who don't wish to type in the complete listing of the program, I will supply a copy on tape for $2.50, the approximate cost of the tape and postage. Send a self-addressed envelope to:

Ralph Owens
Box 202
Enterprise, KS 67441

```
360 PRINT"TIME CONTROL LOOP COUNTER DECREASES"
370 GOSUB2020
380
390 PRINT"■■■■NOW I WILL DEMONSTRATE THE SCREEN"
400 PRINT"REVERSAL PROGRAM.  I WILL TOGGLE"
410 PRINT"THE ENTIRE SCREEN 15 TIMES"
420 FORN=1TO15:SYS(863)
430 FORM=1TO200:NEXTM,N
440 GOSUB2020
450
460 PRINT"■NOW I WILL DEMONSTRATE THE CLEARING"
470 PRINT"OF A WINDOW ON THE SCREEN.  FIRST I"
480 PRINT"WILL REVERSE THE ENTIRE SCREEN, SO THAT"
490 PRINT"YOU CAN SEE WHAT HAS BEEN CLEARED■■■■"
500 SYS(863):GOSUB2020:SYS(890)
510 GOSUB2020
520
530 PRINT"■NOW I WILL CHANGE THE LOCATION OF THE WINDOW AND ITS DIMENSIONS"
540 POKE891,20:POKE895,130:REM SETS UPPER LEFT CORNER OF WINDOW
550 POKE 899,10:POKE901,20
560 SYS(863):SYS(890)
570 GOSUB2020
580
590 PRINT"■THE NEXT DEMONSTRATION IS OF THE"
600 PRINT"SCROLL DOWN ROUTINE."
610 PRINT"■■I WILL SCROLL THE SCREEN DOWN ONE STEP"
620 PRINT"■■THEN 10 STEPS
630 GOSUB 2020
640
650 SYS(926)
660 GOSUB2020
670 FORN=1TO10:SYS(926):NEXTN
680 GOSUB2020
690
700 PRINT"■THE IMPORTANT ADDRESSES IN THESE"
710 PRINT"ROUTINES ARE:"
720 PRINTTAB(10)"THEIR STARTING ADDRESSES"
730 PRINTTAB(10)"THEIR CONTROL ADDRESSES"
740 PRINT"■■THE STARTING ADDRESSES ARE"
750 PRINTTAB(10)"■SCROLL RIGHT     826"
760 PRINTTAB(10)"REVERSE SCREEN   863"
770 PRINTTAB(10)"CLEAR WINDOW     890"
780 PRINTTAB(10)"SCROLL DOWN      926"
790 GOSUB2020
800
810 PRINT"■THE CONTROL POINTS ARE IDENTIFIED"
820 PRINT"IN THE COMMENTS IN THE LISTING."
830 PRINT"■THEY CONSIST BASICALLY OF THE BASE"
840 PRINT"ADDRESS, AND THE INITIALIZING OF THE"
```

```
850 PRINT"POINTER REGISTERS X AND Y.  BY CHANGING"
860 PRINT"THESE, YOU CAN MAKE THE ROUTINES WORK"
870 PRINT"FOR ANY PORTION OF THE SCREEN."
880 PRINT"▮TO SAVE THE ROUTINES ON TAPE 1,"
890 PRINT"ENTER THE MONITOR (BY SYS(1024))"
895 PRINT"AND TYPE:"
900 PRINT"▮S "CHR$(34)"SCREEN ROUTINE"CHR$(34)",01,033A,03C7"
910 PRINT"▮TO SAVE ON DISK DRIVE 0 TYPE:"
920 PRINT"▮S "CHR$(34)"0:SCREEN ROUTINE"CHR$(34)",08,033A,03C7"
930 PRINT"▮TO SAVE ON DISK DRIVE 1 TYPE:"
940 PRINT"▮S "CHR$(34)"1:SCREEN ROUTINE"CHR$(34)",08,033A,03C7"
950
960 END
970
980
990 REM DATA FOR ROUTINES WITH ASSEMBLY MNEMONICS
1000
1010 REM SCROLL RIGHT ROUTINE
1020
1030 REM DECIMAL     ASSEMBLY           COMMENTS
1040 REM LISTING     NMEMONICS
1050
1060 DATA 169,0     :REM LDA #$00      :THESE FOUR LINES TELL WHERE
1070 DATA 133,1     :REM STA $01       :TO START MOVING THE SCREEN.
1080 DATA 169,128   :REM LDA #$80      :THE INDIRECT ADDRESS IS
1090 DATA 133,2     :REM STA $02       :AT LOCATIONS 1,2
1100 DATA 162,25    :REM LDX #$19      :SET X TO #LINES TO MOVE
1110 DATA 160,38    :REM LDY #$26      :SET Y TO #COLUMNS TO MOVE
1120 DATA 177,1     :REM LDA ($01),Y   :LOAD CHAR IN NEXT COLUMN
1130 DATA 200       :REM INY           :INCREMENT Y AND
1140 DATA 145,1     :REM STA ($01),Y   :STORE IN THIS COLLUMN
1150 DATA 136       :REM DEY           :DEC Y TWICE TO GO
1160 DATA 136       :REM DEY           :BACK TO NEXT COLUMN
1170 DATA 16,247    :REM BPL $F7       :BUT CHECK IF DONE FIRST
1180 DATA 165,1     :REM LDA $01       :IF SO THEN ADD 40
1190 DATA 105,40    :REM ADC #$28      :TO BASE ADDRESS
1200 DATA 133,1     :REM STA $01       :TO GO TO NEXT LINE
1210 DATA 165,2     :REM LDA $02       :ON SCREEN.
1220 DATA 105,0     :REM ADC #$00      :ADD IN CARRY TO
1230 DATA 133,2     :REM STA $02       :HI ADDRESS AND
1240 DATA 202       :REM DEX           :DEC X TO COUNT # LINES
1250 DATA 208,230   :REM BNE $E6       :IF NOT DONE THEN KEEP GOING
1260 DATA 96        :REM RTS           :ELSE RETURN TO BASIC
1270
1280 REM REVERSE SCREEN ROUTINE
1290
1300 REM DECIMAL     ASSEMBLY           COMMENTS
1310 REM LISTING     NMEMONICS
1320
1330 DATA 169,0     :REM LDA #$00      :LOAD BASE ADDRESS
1340 DATA 133,1     :REM STA $01       :INTO ADDRESS 1,2
1350 DATA 169,128   :REM LDA #$80
1360 DATA 133,2     :REM STA $02
1370 DATA 162,4     :REM LDX #$04
1380 DATA 160,0     :REM LDY #$00
1390 DATA 177,1     :REM LDA ($01),Y   :GET CURRENT CHAR
1400 DATA 73,128    :REM EOR #$80      :TOGGLE 7TH BIT
1410 DATA 145,1     :REM STA ($01),Y   :STORE NEW CHAR
1420 DATA 136       :REM DEY           :GOTO NEXT CHAR
1430 DATA 208,247   :REM BNE $F7       :IF NOT DONE
1440 DATA 230,2     :REM INC $02       :INC BASE ADDRESS
1450 DATA 202       :REM DEX           :DEC COUNTER
```

```
1460 DATA 208,240   :REM BNE $F0       :CONTINUE IF NOT DONE
1470 DATA 96        :REM RTS           :ELSE RETURN TO BASIC
1480
1490 REM CLEAR WINDOW ROUTINE
1500
1510 REM DECIMAL     ASSEMBLY           COMMENTS
1520 REM LISTING     NMEMONICS
1530
1540 DATA 169,0     :REM LDA #$00      :LOAD BASE ADDRESS
1550 DATA 133,1     :REM STA $01       :IN LOCATIONS 1,2
1560 DATA 169,128   :REM LDA #$80
1570 DATA 133,2     :REM STA $02
1580 DATA 162,5     :REM LDX #$05      :LOAD HEIGHT OF WINDOW
1590 DATA 160,16    :REM LDY #$10      :LOAD WIDTH OF WINDOW
1600 DATA 169,32    :REM LDA #$20      :LOAD A BLANK
1610 DATA 145,1     :REM STA ($01),Y   :STORE BLANK IN LOCATION
1620 DATA 136       :REM DEY           :GOTO NEXT POS
1630 DATA 16,251    :REM BPL $FB       :CHECK IF DONE
1640 DATA 165,1     :REM LDA $01       :GET LO BASE ADDRESS
1650 DATA 24        :REM CLC           :CLEAR CARRY JUST IN CASE
1660 DATA 105,40    :REM ADC #$28      :ADD 40 FOR NEXT ROW
1670 DATA 133,1     :REM STA $01       :AND REPLACE
1680 DATA 165,2     :REM LDA $02       :NOW ADD CARRY
1690 DATA 105,0     :REM ADC #$00      :TO HI BASE ADDRESS
1700 DATA 133,2     :REM STA $02       :AND REPLACE
1710 DATA 202       :REM DEX           :NOW COUNT NUMBER OF ROWS
1720 DATA 16,231    :REM BPL $E7       :IF NOT DONE, CONTINUE
1730 DATA 96        :REM RTS           :ELSE RETURN TO BASIC
1740
1750
1760 REM SCROLL DOWN ROUTINE
1770
1780 REM DECIMAL     ASSEMBLY           COMMENTS
1790 REM LISTING     NMEMONICS
1800
1810 DATA 169,191 :REM LDA #$BF      :LOAD STARTING ADDRESS LO
1820 DATA 133,1   :REM STA $01       :AND STORE IN USR VECTOR
1830 DATA 169,131 :REM LDA #$83      :LOAD STARTING ADDRESS HI
1840 DATA 133,2   :REM STA $02       :AND STORE IN USR VECTOR
1850 DATA 169,4   :REM LDA #$04      :LOAD PAGE COUNTER
1860 DATA 133,0   :REM STA $00       :AND STORE IN USR VECTOR
1870 DATA 162,240 :REM LDX #$F0      :LOAD X WITH 240 (1/4 OF 1000-40)
1880 DATA 160,0   :REM LDY #$00      :INDEX BY ZERO
1890 DATA 177,1   :REM LDA ($01),Y   :TO GET NEXT CHARACHTER
1900 DATA 160,40  :REM LDY #$28      :THEN RESET POINTER TO ONE LINE LOWER
1910 DATA 145,1   :REM STA ($01),Y   :AND STORE CHAR ONE LINE LOWER
1920 DATA 198,1   :REM DEC $01       :MOVE TO NEXT BASE ADDRESS
1930 DATA 169,255 :REM LDA #$FF      :CHECK TO SEE IF
1940 DATA 197,1   :REM CMP $01       :A PAGE BOUNDARY HAS BEEN CROSSED
1950 DATA 208,2   :REM BNE $02       :IF NOT THEN SKIP
1960 DATA 198,2   :REM DEC $02       :ELSE DEC HI ADDRESS
1970 DATA 202     :REM DEX           :NOW DEC COUNTER
1980 DATA 208,235 :REM BNE $EB       :CHECK TO SEE IF 1/4 FINISHED
1990 DATA 198,0   :REM DEC $00       :IF SO THEN DEC PAGE COUNTER
2000 DATA 208,229 :REM BNE $E5       :CHECK IF FINISHED
2010 DATA 96      :REM RTS           :IF SO THEN RETURN TO BASIC
2020 PRINT"▒▒PRESS ANY KEY TO CONTINUE"
2030 GETCHAR$:IFCHAR$=""THEN2030
2040 RETURN
2050 END
READY.
```

©

Machine Language: Comparison Shopping

Jim Butterfield
Toronto, Canada

One of the early things we learn in 6502 coding is how to compare numbers. Unfortunately, we are often taught wrongly. It's not hard: but some of the intuitive things we do at the start can backfire on us later.

We can compare any of our three data registers — A, X, or Y — with memory or with a fixed (immediate) value. We will usually follow this comparison with a branch: BEQ if equal, BNE if not equal, BCS (Branch Carry Set) if the register is equal or higher, BCC (Branch Carry Clear) if lower. Unless you're a very subtle programmer, don't follow a comparison with BPL (Branch Plus) or BMI (Branch Minus).

If you want to test A for less than 8, for example, it would seem natural to say CMP #$08:BMI LESS — which is wrong! Correct coding is BCC LESS. Don't feel bad if you have done this: even the Microsoft interpreter gets this one wrong occasionally.

Where does the problem arise? We're told, correctly, that a Compare instruction does a subtraction and throws away the result, leaving the flags behind to tell us how the comparison has gone. The flags affected are the same ones as for a subtract: Z, C, and N (Zero, Carry and Negative). Reading this we tend to think — wrongly! — that if we subtract a smaller number from a larger number we must get a positive result. Wrong! If we subtract a very small number, say 01, from a very large number, say hexadecimal FF, the result is of course FE and that is a negative number. If we then tested the N flag with BPL or BMI, our program would seem to tell us that 01 is not smaller than FF — and this is obviously nonsense. Don't get the impression that BPL and BMI might work if they were used on signed numbers. They don't work at all for almost any application.

The Carry flag, on the other hand, never plays us false. Carry is set if the value in the register is equal or higher.

So break yourself of the habit of using BPL or BMI after comparisons. Switch to BCS and BCC — they won't trip you up.

Address Comparisons

A common job in medium-to-large size machine language programs is to compare one address against another. You'll often be walking an indirect address through a table, and you want to know when you have reached the end.

There is no single instruction which will compare two bytes at a time for you. You'll have to make up a series of instructions to do it. There are many ways, but one of my favorites is two-byte subtraction. If you subtract one number from another, you'll notice right away which one is higher.

When using subtraction, remember not to fall into the same BPL trap we have already mentioned. We must once again make a point of using BCS and BCC.

First, a less elegant way which illustrates the methodology. Suppose we have an address at ADDR and ADDR + 1, low order first as usual; and suppose our second address is stored at TOP and TOP + 1. We can spot whether ADDR has reached or passed TOP by subtracting (low order first, of course):

```
SEC
LDA ADDR        low order
SBC TOP
LDA ADDR + 1    high order
SBC TOP + 1
BCS REACHED     branch if there or beyond
```

Notice that we don't care about the results of the subtraction: the flags tell us all we need. Now ... since we don't need those results, we could change the first subtraction to a comparison. This would save time and space, since the compare instructions don't need SEC:

```
LDA ADDR        low order
CMP TOP
LDA ADDR + 1    high order
SBC TOP + 1
BCS REACHED
```

This works in the same way as the previous program, but faster and in one less byte. It's quite common in monitors and other large programs.

It often happens that you have your working address loaded into your registers; you may have been doing arithmetic on the address. If your high-order address happens to end up in A, and the low-order, say, in X, you can then code quite elegantly:

```
CPX TOP         low order
SBC TOP + 1     high order
BCS reached
```

Summary

Comparisons are quite easy to handle, once you get rid of your bad habits. The same techniques can be readily extended to compare values of greater than one byte.

After a while, the coding methods become quite automatic, and comparing methodology will be just one more tool in your bag of tricks. At that time, you can start writing programs that are beyond compare ...

Using TAB, SPC And LEN

Ronald L. Straley
Ft. Myers, FL

Back to the basics of programming on the PET. Let us look at two of the functions used to print with: (TAB and SPC). Also we will look at the LEN function while we demonstrate the other two.

According to COMMODORE's write-up, the TAB function will print strings in the position you specify + 1 which is great for printing to the screen in an unformatted manner. They say the SPC function prints the number you specify in blanks or spaces to move your print positions. But you still have an unformatted printout. What we will do is work up a program to demonstrate formatting on the screen and, also, when we want to, we can use the same routine to format the printer.

When using the TAB function, the PET always starts counting from the left side of the screen whenever TAB is encountered. As far as SPC it always starts counting from the last printed position and counts from there. **Example 1** is using TAB to print to the screen. **Example 2** is using SPC to print to the screen.

The problems start when we want to use the same routine for the printer. The printer looks at TAB and SPC in the same manner, always counting from the last printed position and is also unformatted.

Example 2

We can see that by using TAB or SPC by themselves, we can't use the same routine for the printer that we use for the screen. This is where the LEN function comes into play for us. Accordingly, the LEN function will count how many spaces there are in the string we want to print. So, with a combination of TAB or SPC and LEN, we should be able to format our output and use it either on the screen or on our printer.

What we want to do is space our print over so that, whatever the length of the string we want to print is, it will always line up in a formatted manner.

First we will try TAB and set up our format to space over X spaces and then format our output right justified. We will use a statement like: PRINT-TAB(X-LEN(B1$));B1$. What this will do is tab over X number of spaces and then subtract the number of spaces in our string and then start printing, except TAB starts from the right of the screen and we end up with no format again.

Example 3

But we now have our printer formatted.

Example 4

Now we are down to our last option, but the best one of all. SPC and LEN used in combination are the commands we have been looking for. On both the screen and the printer, the SPC function is used in the same manner: it starts counting from the last printed position. Let us now use a statement like: PRINTSPC(X-LEN(B1$));B1$ This will let us space over X number of spaces, but will then subtract the number of spaces in our string and then start printing from there. Since the printer and the screen treat SPC in the same manner, both printouts will be the same.

Example 4

This will work on all ROM machines, and 40 or 80 column, but if you want to use the 80 column PET, it works great the way it is since both the screen and the printer are 80. If you are using the 40 column PET and printer you may want to add 68 IFA$="P" THENX=15:Y=20 This will cause the printer to spread the lines out and you can have more room between the columns.

You can have fun and do some experimenting with the 3 commands TAB, SPC and LEN. You should now be able to fix up those troublesome print routines and only use one routine to do all your printing, whether on the screen or the printer.

```
10 PRINT"{CLR}":X=5:Y=8
20 PRINT"{RVS}S{OFF}CREEN OR {RVS}P{OFF}RINTER"
30 GETA$:IFA$=""GOTO30
40 IFA$="P"THENOPEN1,4:CMD1
50 B$="RON"
60 B1$=".56"
65 B2$="1.25"
66 B3$="23.67"
70 PRINTB$;TAB(X);B1$;TAB(Y);B2$;TAB(Y);B3$
71 :
72 :
75 PRINTB$;TAB(X);B2$;TAB(Y);B3$;TAB(Y);B1$
78 PRINT:PRINT
```

```
79 :
80 PRINTB$;SPC(X);B1$;SPC(Y);B2$;SPC(Y);B3$
81 :
82 :
85 PRINTB$;SPC(X);B2$;SPC(Y);B3$;SPC(Y);B1$
88 PRINT:PRINT
89 :
90 PRINTB$;TAB(X-LEN(B1$));B1$;TAB(Y-LEN(B2$));B2$;TAB(Y-LEN(B3$));B3$
91 :
92 :
95 PRINTB$;TAB(X-LEN(B2$));B2$;TAB(Y-LEN(B3$));B3$;TAB(Y-LEN(B1$));B1$
98 PRINT:PRINT
99 :
100 PRINTB$;SPC(X-LEN(B1$));B1$;SPC(Y-LEN(B2$));B2$;SPC(Y-LEN(B3$));B3$
101 :
102 :
105 PRINTB$;SPC(X-LEN(B2$));B2$;SPC(Y-LEN(B3$));B3$;SPC(Y-LEN(B1$));B1$
110 PRINT
111 :
200 IFA$="P"THENPRINT#1:CLOSE1
```

```
EXAMPLE 1   RON   .561.2523.67            EXAMPLE 3   RON.561.2523.67
            RON   1.2523.67.56                        RON1.2523.67.56

EXAMPLE 2   RON     .56      1.25   23.67 EXAMPLE 4   RON  .56    1.25  23.67
            RON    1.25     23.67     .56             RON 1.23   23.67    .56
```

©

THE SBC GAZETTE

Nuts And Volts #6

Gene Zumchak
Buffalo, NY

In my last column, I reviewed the hardware aspects of handshaking and also described the workings of a programmable input/output port. I suggested that I would detail the transmit and receive software for using handshaking to pass a block of data between two systems. While I still plan to do that (some day), I think a more meaningful exercise would be to consider the hardware and software necessary to connect a common peripheral to a computer system. Namely, I would like to consider the connection of a parallel style, or so-called "Centronics" style printer.

As I mentioned last time, (actually a whole line was left out during the typesetting), Centronics style handshake timing has three possible flags. The one asserted by the sender is a pulse called DATA STROBE and is usually low-true. The receiver responds with a BUSY indication, usually high-true. When BUSY falls false (unbusy), the ACK pulse (low-true) is asserted. Again as mentioned, only one of the return flags, either BUSY or ACK, need be used.

I just recently took delivery of a NEC Spinwriter with a parallel interface. While it didn't quite take the day I allowed to get it running with my SYM, I did encounter a few surprises. While the connector and the pin assignments are definitely Centronics style, the flags are not true Centronics.

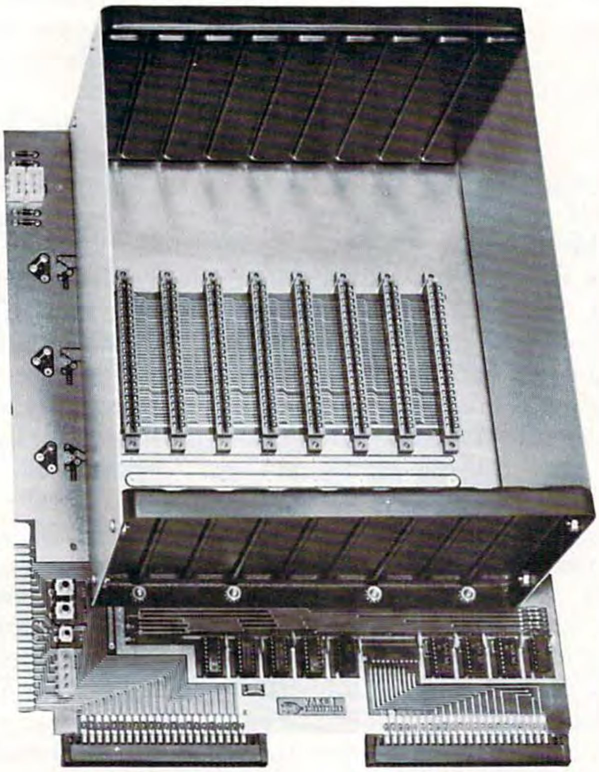
The product description manual gives one short paragraph on the timing. Fortunately, the accompanying timing diagram accurately describes the timing. While the printer has a flag called BUSY, it is not the Centronics BUSY, and does not take part in the handshake sequence. To avoid confusion, it might better have been called READY. When this BUSY goes low, it indicates that the printer is ready to receive data. Only ACK takes place in handshaking. An initial ACK is sent to indicate that it is ready for the first character, thus, ACK is used to prompt for characters rather than to indicate that the printer is through processing the last character. Actually, for characters beyond the first, the two descriptions mean the same thing. For my Receive Only model (5530), characters are accepted with the handshaking, but no characters are printed until the buffer is full, or a carriage return is received. Busy goes true when the buffer is full, and remains high while the line is being printed. This timing is shown in Fig. 1.

The easiest way to handle Centronics timing is to poll BUSY and forget about ACK. However, this is not true Centronics timing, and BUSY takes no part in the handshaking. Since ACK is a narrow pulse (2.2 microseconds), it cannot be polled, but must be used to set a flip-flop. This precludes the use of a simple input port bit to handle the flag from the printer. There are two choices. We can either use a family port chip which has edge-sensitive inputs (like the 6522), or we must provide the flip-flop, which is reset by ACK and polled as a conventional BUSY flag with an input bit. DATA STROBE can be used to set the BUSY flip-flop when a character is sent. This alternative is shown in Fig. 2.

Since my SYM has no fewer than three 6522s which are available for I/O, I did not have to provide the flip-flop, but could use one of the edge-sensitive input control bits. The connection I used is shown in Fig. 3. I used the VIA chip that responds to the base address $A800. I used the low seven bits of Port A for the printer data, and bit PA7 as an input bit to poll BUSY. The CA2 output bit was used for the DATA STROBE and input control bit CB1 was used to detect the ACK edge. As can be seen from the figure, the data lines and strobe were buffered. This was necessary since the Spinwriter inputs have a 470 ohm pullup resistor. When the input is zero, there are five volts across 470 ohms and about 10 milliamps are drawn in addition to the 1.6 mA TTL input. A family port chip can usually drive only one TTL load, but certainly not 11.6 mA. I used an octal three-state gate chip for the buffer (81LS97). Any non-inverting gate, like the 74LS241 would be suitable.

A suitable program for sending a character to the printer is shown. For those not familiar with the 6522, certain defined events cause a flag in the Interrupt Flag Register (IFR) to be set. A corresponding bit in the Interrupt Enable Register (IER) can be set with software to enable the occurrence of any particular event to generate an interrupt as well. In this case, interrupts are not desirable and we will poll the IFR to see if our flag has been set. Four control bits, CA1, CA2, CB1, and CB2 can be used as input/output flags for handshaking.

VAK-1 MOTHERBOARD



We also carry:

SYM-1	**$229⁰⁰**
AIM-65 w/1K	**389⁰⁰**
AIM-65 w/4K	**439⁰⁰**

We also do custom hardware and software for the 6502 microprocessor

PRICE: **$139⁰⁰**
Call or write for shipping charges and our complete catalog.

The VAK-1 was specifically designed for use with the KIM-1, SYM-1 and the AIM 65 Microcomputer Systems. The VAK-1 uses the KIM-4* Bus Structure, because it is the only popular Multi-Sourced bus whose expansion boards were designed specifically for the 6502 Microprocessor.

SPECIFICATIONS:

- Complete with rigid CARD-CAGE
- Assembled (except for card-cage). Burned in and tested.
- All IC's are in sockets
- Fully buffered address and data bus
- Uses the KIM-4* Bus (both electrical Pin-out and card size) for expansion board slots
- Provides 8 slots for expansion boards on 1″ centers to allow for wire-wrap boards
- Designed for use with a Regulated Power Supply (such as our VAK-EPS) but has provisions for adding regulators for use with an unregulated power supply.
- Provides separate jacks for one audio-cassette, TTY and Power Supply.
- Board size: 14.5 in. Long x 11.5 in. Wide x 8 in. High
- Power requirements; 5V.DC @ 0.2 Amps.

**KIM-4 is a product of MOS Technology/C.B.M.*

RNB ENTERPRISES INCORPORATED

4030 N. 27th Avenue
Suite D.
Phoenix, AZ 85017

master charge VISA

CA2 and CB2 can be outputs which are automatically asserted when data is read or written to the corresponding port. For write handshaking, for example, CA2 can be made to go low automatically when writing to Port A. CA2 is returned high by a transition on the CA1 input. Alternatively, CA2 can be pulsed low on a write to Port A, returning high without feedback after 500 ns. Finally, CA2 can be manually made low or high with software. Both CA2 and CB2 can serve as edge-sensitive inputs.

It is assumed that the character to be printed is in the accumulator when the program is called. It is good practice for an output routine to leave the registers unchanged. Since X and Y are not used in the program, they will not be affected and need not be saved. The accumulator is pushed on the stack twice; once for later use in the program, and a second time so that the program can terminate with A unchanged. This permits one output routine to follow another.

Lines 260 and 270 cause the low seven bits of Port A to be made outputs, keeping PA7 as an input bit. The PCR register, which defines the use of the CA and CB control pins, is initialized with data "$0E", called STRBOFF, which manually forces CA2 high.

The BUSY output is polled in the first loop, WAIT1. When BUSY goes false, the program falls into the second loop, WAIT2, where the IFR is read and the bit corresponding to the CB1 input flag is tested. This flag is set by a transition on the ACK line. When that condition is found, the flag is cleared by reading port B or Input Register B (IRB). Now the character to be printed is pulled from the stack and sent to port A. The DATA STROBE is exercised, by manually forcing CA2 low, then high again. This destroys the character in A, which is restored prior to the return with the second pull from the stack.

This program is by no means the only solution. There would appear to be a large number of possible connections of the port bits to accomplish the same thing, and perhaps different programs for a particular connection. However, many combinations and programs will not work. For example, why did I manually lower and raise the CA2 strobe? Why not program CA2 for the handshake pulse mode and let it pulse automatically when data is written to the A port? I confess that I tried it. Since auto pulsing will also occur for a "read" of port A as well, when an attempt was made to poll BUSY at WAIT1, data was unintentionally strobed. This caused the same character to be printed more than once. Another possible change would be to detect ACK with the CA1 input instead of CB1, and keep all functions in Port A. Again, reading Port A at WAIT1 would cause the CA1 flag to be cleared before it was recognized. The program would then fall into the loop at WAIT2 and wait for an $\overline{\text{ACK}}$ signal that would never occur since the printer would be waiting for a DATA STROBE that will never occur.

An experienced programmer will not panic when something strange happens, or worse yet, nothing at all happens. In this case, most of the unexpected results can be predicted with a careful reading of the 6522 spec' sheet.

I was initially annoyed because a legitimate BUSY flag wasn't available and also because two return flags seemed to be required. Actually, the printer has two additional output flags that I did not choose to use, one called FAULT and another for PAPER OUT. Both of these flags, however, are reflected in the BUSY flag. That is, if a fault occurs (cover not closed) or if the PAPER OUT switch is engaged, BUSY will not return false. A PAPER indicator appears on the printer panel, as well as a READY light. Thus, nothing is lost by not using these additional flags.

Is, in fact, BUSY necessary? After I had a program running successfully, I NOPed the WAIT1 loop. I then created "paper out" and "fault" conditions. The printer stopped and the READY light went out. Printing resumed as soon as the condition cleared. Thus, it appears that the printer always affects BUSY before $\overline{\text{ACK}}$, and $\overline{\text{ACK}}$ will not be asserted if an unready state exists. It would then appear that the information contained in BUSY is in fact redundant, and that only one flag, $\overline{\text{ACK}}$, and one wait loop need be used to obtain normal print operation.

The actual incorporation of the Print Character program just considered will depend upon the particular computer system which you are using, and how it handles input and output. This is, in fact, worthy of an article by itself, and I am preparing an article on console input/output, if not for this issue, then the next.

If one is attempting to interface a Spinwriter to a PET, the above program is not necessary, if the printer is interfaced by the GPIB bus. In this case, the polling and sending of flags is performed routinely internally. However, the printer will need a hardware interface that makes it look like a legitimate GPIB instrument. While building this

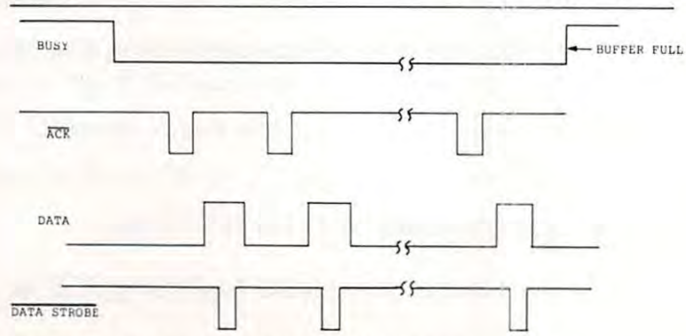


Figure 1. Timing Signals for NEC Spinwriter 6522

A BRILLIANT FUTURE FOR YOUR AIM-65 WITH THE BANKER MEMORY ™

Your 36K of free address space is the AIM's most valuable and limited resource. With today's large capacity RAM boards, ROM boards, disk systems, video boards, and other expansion accessories it is easy to deplete this resource before the application requirement is satisfied. MTU has solved this problem.

THE BANKER MEMORY contains 32K of RAM, 4 PROM sockets for 2716/2732/2332, a PROM programmer, 40 bits of parallel I/O, and 4 timers from two 6522 I/O chips. Addressing is extremely flexible with the RAM independently addressable in 4K blocks, PROM's independently addressable, and I/O addressable anywhere on a 64 byte boundary (even in AIM's I/O area at AXXX by adding a single jumper to the AIM).

This may sound familiar, but read on! Unlike other AIM compatible memory boards, THE BANKER MEMORY has on-board bank-switching logic! The four 8K blocks of RAM plus the 4 PROM sockets make up 8 **resources**, each associated with a bit in an Enable Register. Through this Enable Register resources may be turned on and off under software control. When a resource is off, its address space is freed for other uses. You can even put BANKER resources at the same address and switch among them for virtually unlimited RAM and PROM expansion! You can even have multiple page zero's and stacks! Do you need 160K byte of memory? It only takes 5 of THE BANKER MEMORY boards and you end up with 5 page zeros and stacks to boot!

There's more! The BANKER MEMORY also incorporates 18 bit addressing which allows for the 256K address spaces of the future. RAM, PROM, and I/O each has its own full 18 bit address decoder which allows these resources to be in different 64K banks. This board and other MTU products, such as our 320 by 200 dot VISIBLE MEMORY and Floppy Disk Controller with 16K DMA RAM, can turn your AIM into a truly powerful 6502 computer that far surpasses the packaged systems in functional performance.

INTRODUCTORY SPECIAL K-1032-1 32K BANKER MEMORY FULLY ASSEMBLED AND TESTED $395.00 ($450.00 as of March 1, 1980) or the K-1032-2 16K RAM only with bank switching and 18 bit address bus only $295.00

Isn't it time you took a closer look at MTU — we offer you power now with an eye to the future.

WRITE OR CALL TODAY FOR OUR 48 PAGE FALL 1980 6502 CATALOG
International requests include $1.00

VISA and MASTERCARD accepted

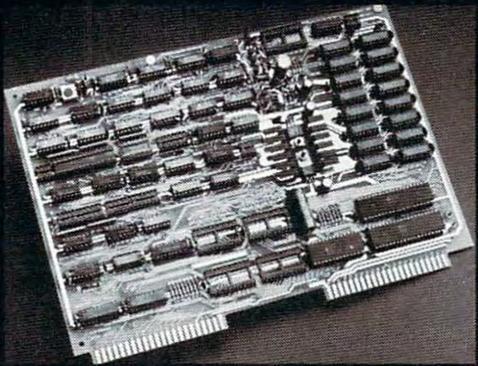


Photo credit:
SUPERNOVA CRAB NEBULA:
Palomar Observatory,
California Institute of Technology

MTU
Micro Technology Unlimited
2806 Hillsborough Street
P.O. Box 12106
Raleigh, NC 27605, U.S.A.
(919) 833-1458

interface is not trivial, it would not appear to require more than a handful of gates and a flip-flop or two, perhaps $10 or less in parts (sans connectors). Since I do not have a PET, I cannot verify my gut feeling. However, I do hope to get my hands on one in the near future, and it should result in an article on building a GPIB printer interface, or perhaps building an interface for any non-standard peripheral (without using another microcomputer on the inside).

I welcome your feedback on my articles. I know that at least two people read my column.

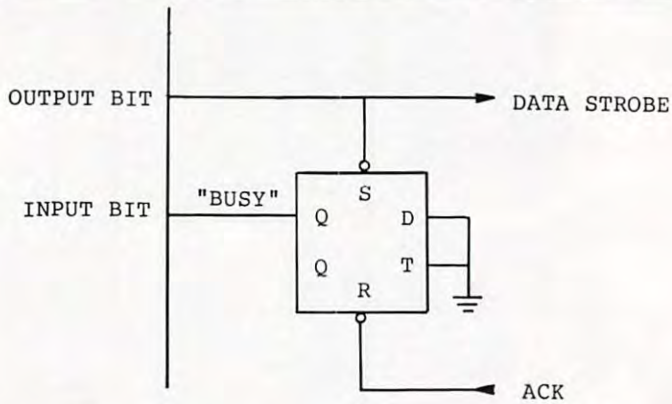


Figure 2. Flip-flop to generate "Centronics" type BUSY from ACK and DATA STROBE.

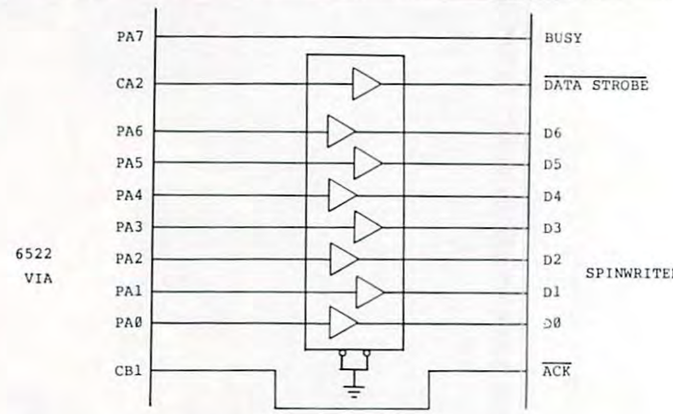


Figure 3. Connection from VIA to Spinwriter.

```
                0010 ;BASIC HANDSHAKING WRITE ROUTINE FOR SPINWRITER
                0020                ;
                0030                ;2/25/80
                0040                ;
                0050                ; $A800 IS THE BASE ADDRESS OF 6522
                0060                ; ON SYM USED FOR PRINTER PORT
                0070                ;
                0080 IRB            .DE $A800
                0090 ORA            .DE $A801     ; LOW SEVEN BITS PORT A
                0100                ;                IS PRINTER DATA
                0110 IRA            .DE $A801     ; PA7 IS USED TO POLL BUSY
                0120 DDRA           .DE $A803
                0130 PCR            .DE $A80C     ; DETERMINES CONTROL BIT USE
                0140                ;               CA2 USED AS OUTPUT FOR
                0150                ;               PRINTER DATA STROBE
                0160 IFR            .DE $A80D     ; BIT 4 (CB1) USED FOR ACK
                0170 STRBOFF        .DE $0E       ; MAKES CA2 HIGH
                0180 STRBON         .DE $0C       ; MAKES CA2 LOW
                0190                ;
                0200                .BA $300
                0210                .MC $300
                0220                .OS
                0230                ;
0300- 48        0240 PRINTCHAR      PHA  ; SAVE FOR POSTERITY
0301- 48        0250                PHA
0302- A9 7F     0260                LDA #$7F      ; MAKE LO-7 BITS PORTA OUTPUTS
0304- 8D 03 A8  0270                STA DDRA
0307- A9 0E     0280                LDA #STRBOFF         ; MAKE SURE CA2 HIGH
0309- 8D 0C A8  0290                STA PCR
030C- AD 01 A8  0300 WAIT1          LDA IRA       ; WAIT FOR UNBUSY
030F- 30 FB     0310                BMI WAIT1
0311- AD 0D A8  0320 WAIT2          LDA IFR       ; WAIT FOR ACK PULSE
0314- 29 10     0330                AND #$10
0316- F0 F9     0340                BEQ WAIT2
0318- AD 00 A8  0350                LDA IRB       ; CLEAR CB1 FLAG
031B- 68        0360                PLA '         ; GET BACK PRINT DATA
031C- 8D 01 A8  0370                STA ORA       ; SEND TO PRINTER
031F- A9 0C     0380                LDA #STRBON   ; PULSE DATA STROBE
0321- 8D 0C A8  0390                STA PCR
0324- A9 0E     0400                LDA #STRBOFF
0326- 8D 0C A8  0410                STA PCR
0329- 68        0420                PLA '         ; GET BACK PRINT DATA
032A- 60        0430                RTS
                0440                .EN
```

DLOAD : AIM Memory Loader

Joel Swank
Rockaway, OR

The AIM 65 monitor gives you the ability to save and load non-contiguous blocks of memory on cassette tape, paper tape, or a user device. This handy feature allows you to save a program, along with any vectors or data values it needs to execute, and then to load it all with just one command. The AIM assembler uses the same format for its object files. You can assemble several programs at different locations and load them all with one load command.

But the AIM load command is missing a couple of useful features. When loading a file with the AIM load command there is no way to tell which memory locations are being loaded. After assembling a program there is no way, without a listing, to tell where the program ends. You might also want to assemble a program at one address and load it into memory at a different address, as in the case of a program that is to reside in ROM. It would also be convenient to be able to save data from one area of memory and load it back to a different area. The AIM load command cannot do this.

DLOAD is a modified version of the AIM load command that adds these two missing features. DLOAD works like the AIM load command except that it first requests an offset with the 'OFFSET=' prompt. This hexidecimal number is input with the AIM subroutine ADDIN. ADDIN is the same routine that inputs your reply to the 'FROM=' and 'TO='

```
0000
0000            ;  DLOAD : LOAD AIM OBJECT FILE AND DISPLAY THE
0000            ;          ADDRESSES OF THE DATA LOADED. OPTIONAL
0000            ;          OFFSET LOAD.

0000            ;   AIM SUBROUTINES

0000            WHEREI =$E848            ;OPEN INPUT
0000            INALL  =$E993            ;INPUT A CHAR FROM AOD
0000            CLRCK  =$EB4D            ;CLEAR CHECKSUM
0000            CHEKAR =$E54B            ;INPUT HEX BYTE
0000            CKERR  =$E385            ;ERROR RETURN
0000            RBYTE  =$E3FD            ;READ OBJECT BYTE
0000            STBYTE =$E413            ;STORE OBJECT BYTE (ADDR)
0000            DU13   =$E520            ;CLOSE TAPE INPUT FILE
0000            ADDIN  =$EAAE            ;INPUT ADDRESS FROM KBD
0000            RCHEK  =$E907            ;CHECK FOR INTERRUPT
0000            WRITAZ =$E2DB            ;DISPLAY CONTENTS OF ADDR
0000            COMIN  =$E1A1            ;NORMAL RETURN TO AIM
0000            NUMA   =$EA46            ;DISPLAY BYTE IN HEX
0000            OUTPUT =$E97A            ;DISPLAY ACCUM
0000            CRLF   =$E9F0            ;SEND CR AND LF

0000            ;   AIM RAM

0000            ADDR   =$A41C            ;OBJECT LOAD POINTER
0000            CKSUM  =$A41E            ;CHECKSUM STORAGE
0000            CURAD  =$A41C            ;ADDRESS INPUT BUFFER

0000            ;   ZERO PAGE

0000            OFFL   =0                ;OFFSET SAVE AREA
0000            OFFH   =1
0000            POINTL =2                ;DUPLICATE LOAD POINTER
0000            POINTH =3
0000            RECLEN =4                ;RECORD LENGTH SAVEAREA

0000                   *=$200

0200 A000       DLOAD  LDY #OFFMSG-LITS
0202 20D502            JSR KEPX          ;DISPLAY 'OFFSET='
0205 20AEEA            JSR ADDIN         ;INPUT ADDRESS
0208 B0F6              BCS DLOAD         ;ERROR - TRY AGAIN
020A AD1EA4            LDA CKSUM         ;ANY ENTERED?
020D F005              BEQ SAVOFF        ;YES, SAVE IT
020F A900              LDA #0
0211 8D1DA4            STA CURAD+1       ;NO, USE ZERO
0214 AD1CA4     SAVOFF LDA CURAD         ;COPY CURAD TO OFFSET
0217 8500              STA OFFL
0219 AD1DA4            LDA CURAD+1
021C 8501              STA OFFH

021E 2048E8     OPFIL  JSR WHEREI        ;OPEN INPUT DEVICE
0221 207C02            JSR STREC         ;START RECORD
0224 20A602            JSR PSTART        ;DISPLAY START ADDRESS
0227 4C4502            JMP BYTLUP

022A 207C02     RECLUP JSR STREC         ;START RECORD
022D A604              LDX RECLEN        ;ZERO LENGTH RECORD?
022F F037              BEQ FINISH        ;YES, END
0231 AD1CA4            LDA ADDR          ;IS NEW ADDRESS EQUAL
0234 C502              CMP POINTL        ;TO OLD ADDRESS?
0236 D007              BNE NEWLOC        ;NO, NEW BLOCK OF MEMORY
0238 AD1DA4            LDA ADDR+1
023B C503              CMP POINTH
023D F006              BEQ BYTLUP

023F 20BC02     NEWLOC JSR PEND          ;DISPLAY END OF LAST RECORD
0242 20A602            JSR PSTART        ;AND START OF THIS ONE

0245 20FDE3     BYTLUP JSR RBYTE         ;INPUT AN OBJECT BYTE
0248 2013E4            JSR STBYTE        ;STORE IT
024B E602              INC POINTL        ;BUMP POINTER
024D D002              BNE NOCY
024F E603              INC POINTH
0251 C604       NOCY   DEC RECLEN        ;COUNT BYTE
0253 D0F0              BNE BYTLUP        ;DO NEXT BYTE

0255            ; END OF RECORD
0255 20FDE3            JSR RBYTE         ;GET CHECKSUM
0258 CD1FA4            CMP CKSUM+1       ;AND COMPARE
025B D008              BNE ERROUT        ;ERROR IF NOT EQUAL
025D 20FDE3            JSR RBYTE
0260 CD1EA4            CMP CKSUM

0263 F0C5              BEQ RECLUP        ;CHECKSUM OK - NEXT RECORD

0265 4C85E3     ERROUT JMP CKERR         ;ERROR EXIT

0268 20BC02     FINISH JSR PEND          ;PRINT ADDRESS OF LAST RECORD
026B A205              LDX #5
```

prompts, so the syntax is the same. The hexadecimal number you enter is added to the starting address of each block of memory in the file. For example, a block that was saved from location $200 can be loaded back at location $1000 by replying 'E00' to the 'OFFSET=' prompt. You can calculate the proper offset by: $1000-$200=$E00. You can also load a file to a location lower in memory by adding $10000 to the desired load address before performing the calculation. A file dumped from location $B000 can be loaded back at $200 as follows: $10200-$B000=$5200. Enter '5200' in response to the 'OFFSET=' prompt. If the file contains multiple blocks, then the offset is added to the starting address of all blocks. This means you must take care when loading a file containing vectors or zero page data. These blocks will also be displaced by the offset you entered. You may load a file to its original address by entering a space or return in response to the 'OFFSET=' prompt.

DLOAD next issues the standard AIM 'IN=' prompt to open the input device. You respond as you normally would when using the AIM load command. DLOAD then displays the start and end addresses of each contiguous block of memory as it is loaded. If you are using an offset, the addresses displayed are those at which the data is being stored and not the addresses in the file. DLOAD calls the AIM RCHEK subroutine at the start of each data block so that you can stop or cancel the program. DLOAD used zero page memory locations 0-4, so be sure not to try to load anything there. Included is a listing of DLOAD assembled at location $200. DLOAD can be executed from ROM.

```
026D 20FDE3  FLUP   JSR RBYTE        ;READ END OF LAST RECORD
0270 CA             DEX
0271 D0FA           BNE FLUP
0273 2093E9         JSR INALL
0276 2020E5         JSR DU13         ;CLOSE TAPE
0279 4CA1E1         JMP COMIN        ;RETURN TO MONITOR

027C                ;   END OF MAINLINE

027C                ;   SUBROUTINES FOLLOW

027C                ;   STREC : INPUT BEGINNING OF RECORD

027C 2007E9  STREC  JSR RCHEK        ;CHECK FOR INTERRUPT
027F 2093E9         JSR INALL        ;SEARCH FOR ';'
0282 C93B           CMP #';'
0284 D0F6           BNE STREC
0286 204DEB         JSR CLRCK        ;CLEAR CHECKSUM
0289 204BE5         JSR CHEKAR       ;GET RECORD LENGTH
028C 8504           STA RECLEN       ;SAVE IT
028E 204BE5         JSR CHEKAR       ;GET RECORD ADDRESS
0291 8D1DA4         STA ADDR+1       ;AND SAVE
0294 204BE5         JSR CHEKAR
0297 18             CLC
0298 6500           ADC OFFL         ;ADD OFFSET
029A 8D1CA4         STA ADDR
029D AD1DA4         LDA ADDR+1
02A0 6501           ADC OFFH
02A2 8D1DA4         STA ADDR+1
02A5 60             RTS

02A6                ;   PSTART : DISPLAY STARTING ADDRESS OF MEMORY BLOCK

02A6 20F0E9  PSTART JSR CRLF         ;NEW LINE
02A9 A007           LDY #STMSG-LITS
02AB 20D502         JSR KEPX         ;DISPLAY 'START='
02AE 20DBE2         JSR WRITAZ       ;DISPLAY ADDRESS
02B1 AD1CA4         LDA ADDR         ;COPY ADDR TO POINT
02B4 8502           STA POINTL
02B6 AD1DA4         LDA ADDR+1
02B9 8503           STA POINTH
02BB 60             RTS

02BC                ;   PEND : DISPLAY ENDING ADDRESS OF MEMORY BLOCK

02BC A00E    PEND   LDY #ENDMSG-LITS
02BE 20D502         JSR KEPX         ;DISPLAY ' END='
02C1 38             SEC
02C2 A502           LDA POINTL       ;DECREMENT LAST ADDRESS
02C4 E901           SBC #1
02C6 8502           STA POINTL
02C8 A503           LDA POINTH
02CA E900           SBC #0
02CC 2046EA         JSR NUMA         ;DISPLAY ADDRESS
02CF A502           LDA POINTL
02D1 2046EA         JSR NUMA
02D4 60             RTS

02D5                ;   KEPX : DISPLAY MESSAGE FROM LITERAL TABLE

02D5 B9E102  KEPX   LDA LITS,Y       ;GET A BYTE
02D8 F006           BEQ RETURN       ;QUIT ON NULL
02DA 207AE9         JSR OUTPUT       ;DISPLAY IT
02DD C8             INY              ;NEXT CHARACTER
02DE D0F5           BNE KEPX
02E0 60      RETURN RTS

02E1                ;   LITERAL TABLE

02E1         LITS   =*

02E1 4F46    OFFMSG .BYTE 'OFFSET',0
02E3 00


02E8 5354    STMSG  .BYTE 'START=',0
02EA 00
02EF 2045    ENDMSG .BYTE ' END=',0
02F1 00

02F2                .END
02F2          ERRORS= 0000
```

©

New Products

Educational Software Exchange Announces Mail-Order Service...

SOFTSWAP is a joint project of the San Mateo County Office of Education in Redwood City, CA, and Computer-Using Educators (CUE). SOFTSWAP offers approximately 240 public domain instructional programs collected for the TRS-80, Commodore PET, Apple and Compucolor. A disk of programs for the Atari is in preparation.

Most of the SOFTSWAP programs are short, stand-alone instructional units. Many are drill & practice exercises for the elementary school level or for remedial work at the secondary level. About one-fourth are math oriented. All of the programs have been evaluated by educators and edited for factual and spelling errors, inaccurate or incomplete instructions, programming errors and other problems. Each disk contains from 5 to 28 programs for various subjects and grade levels, all for one microcomputer system.

Programs and disks may be copied without charge by visitors to the Center. Any of the 13 disks may also be purchased by mail at a cost of $10 per disk, or one SOFTSWAP disk will be sent free in exchange to any educator who contributes a disk with at least one original program for the SOFTSWAP.

Our goal is to distribute the SOFTSWAP programs as widely as possible and they may all be freely duplicated onto either disks or tapes (but may not be sold). All programs have been donated to the SOFTSWAP and we encourage educators everywhere to send their own contributions to this growing collection. New disks are listed in the CUE NEWSLETTER, or send $1 for ordering/exchange information to:

Ann Lathrop,
Library Coordinator
SOFTSWAP, San Mateo
County Office of Education
333 Main Street
Redwood City, CA 94063

Be sure to include your name, address and information on the microcomputer system(s) you have.

Copyright Notice: Every effort is made to scrupulously avoid copyright infringement and programs identified as being in violation of copyright are removed from the exchange.

The Complete Guide To Hassle-Free Pet Programming

Reston Publishing Company has announced the publication of a new computer programming book, PET BASIC by Ramon Zamora, William Scarvie and Bob Albrecht.

Perfect for the beginning PET user, this book is filled with lively examples, do-it-yourself exercises and creative explorations. You'll be confident as you experiment with your machine's many capabilities and features, and you'll create graphical images without confusion.

Experienced PET users will find this book a handy reference guide. You'll discover a variety of things your PET can do. Each new piece of information is presented logically, step-by-step with open page formats and lots of humor.

PET BASIC I can also be used to teach children to use PET (age 9 and up). You'll find games and exercises and current symbols in children's world characters.

The authors have over a dozen years of experience teaching and writing books for beginning computer users.

May, 1981, (R-5524-5) paper $12.95, 222 pp. (R-5525-2) cloth $17.95, 272 pp. To Order from Reston: call 800-336-0338.

Smart Terminal Program For The Atari® Features Autodialing

Redmond, Wa. — The MicroPeripheral Corporation has announced TSMART, a smart terminal program written for the ATARI 800®, with provision for autodialing other computers. The program is available on cassette with instructions for transferring to disk. TSMART permits transfer of BASIC programs between a remote host computer and an ATARI cassette or disk storage device. The autodial feature works in conjunction with the AUTO-MICROCONNECTION, a direct connection modem ($199.50), manufactured by the MicroPeripheral Corporation.

The program permits off-line text preparation (messages, manuscripts, letters, etc.) with a text editing or word processing program for on-line transmission. A built-in feature permits creation and storage of text, then transmission by TSMART for those who do not have a text editor.

TSMART also permits transfer of source code assembler files. The recipient can create the object code using an editor/ assembler program. A separate command is available for transferring object (hexadecimal) code files, such as ATARI Music Composer Files.

An AUTOBUF feature will open and close the memory storage buffer automatically when uploading or downloading. TSMART recognizes the automatic buffer open/close (X-on/X-off) codes transmitted by TSMART or other compatible programs. Downloading from FORUM 80 bulletin boards is also accomplished automatically. The buffer can be "toggled" on and off as many times as desired while data is being downloaded. Another feature is software selectable half or full duplex operation.

The program will also automatically send messages to bulletin boards using the standardized block mode or 16 line prompt recognition message entry.

The program was written for the ATARI 800® by Dr. James W. Clark. It can be used with any RS-232 compatible modem, although the dialer feature cannot be used with acoustic modems. TSMART is supplied in a protective binder with extensive easy-to-use operating instructions and is priced at $79.95. For additional information on TSMART or the MICRO-CONNECTION™, contact the MicroPeripheral Corporation, 2643 151st Place N.E., Redmond, WA. 98052, Telephone (206) 881-7544.

Computer Courses For Deaf Adults

Rochester Institute of Technology (RIT) will offer two computer courses for deaf adults this summer through the National Technical Institute for the Deaf (NTID). The first course, Introduction to Data Processing, will provide deaf adults with introductory technical skills applicable to job situations involving computers. Topics include: the relationship of data processing to other parts of a business and an introduction to the BASIC programming language. It will be offered from August 3–7.

The second course, Advanced Data Processing, will give experienced computer users knowledge of software applications on small computer systems. Topics include: data base, interactive programming packages, and color graphics. This course will be offered from August 10-14.

Both courses involve intensive full-day sessions and feature hands-on practice in a microcomputer center. For more information or to register, contact Donald Beil, NTID Data Processing Dept., Rochester Institute of Technology, One Lomb Memorial Drive, Rochester, NY 14623, or call (716)475-6373 (voice or TTY).

Game I/O Extender For Apple II From Vera Computing

Newbury Park, CA, June 15, 1981 — Versa Computing, Inc. has announced a new peripheral device for Apple II computers. E Z Port extends the I/O port to the outside of the computer for quick and easy changeover from game paddles to joystick to VersaWriter, etc.

E Z Port is a board which adheres to the side of the computer with a special foam adhesive strip. A 24" cable connects to the game I/O inside the Apple.

E Z Port incorporates a ZIP socket (Zero Insertion Pressure) in its design. Ordinary DIP plugs and sockets are not designed to be used over and over — eventually the sockets will not make contact and the pins will snap off the DIP plugs. ZIP sockets will help 16 pin connectors last much longer, because no pressure is exerted within the socket until the ZIP's cam lever is switched to the engage position.

At only $24.95, E Z Port is one of the most effective improvements an Apple owner can make to the computer.

E Z Port is available from your local computer retailer or from: Versa Computing, 3541 Old Conejo Road — Suite 104, Newbury Park, CA 91320. Telephone: (805) 498-1956 or 499-4800. Dealer Inquiries Welcome.

AIM 65 Enclosure

This enclosure is specially designed for the Aim 65 microcomputer. Four models are offered to hold any of the major systems on

the market for the Aim 65 including power supply. Formed out of heavy ABS plastic it features a metal card cage and sturdy metal base. The color is white with black trim. The price is $149.95 each. Contact Don-El Enterprises, 3261 Michigan Ave., Costa Mesa, CA 92626. Phone (714)546-7481.

Free PET*/CBM COMAL Compiler

The COMAL USERS GROUP has announced that it will distribute, free of charge, a COMAL Information Package that will include instructions on how Commodore PET/CBM users may obtain a FREE COMAL compiler for their computer.

COMAL is a powerful, struc-

consumer computers

formerly Computers 'R' Us

We accept these major credit cards:

master charge VISA Carte Blanche DINERS CLUB INTERNATIONAL AMERICAN EXPRESS

mail order

OPEN EVERY DAY 9 to 6 PST

California, Alaska & Foreign orders *(714) 698-8088
Shipping Information or Backorders call (714) 698-0260
Service Center and for Technical Information (714) 460-6502

TELEX 695-000 ANS:BETA REF:CCMO

ORDER TOLL FREE
800-854-6654

apple computer
Authorized Dealer

ALL EQUIPMENT IS FCC APPROVED.

APPLE II PLUS 16K......... 1049
APPLE II PLUS 48K
(APPLE Memory)........... 1189
APPLE II Standard Models... CALL
DISK II DRIVE & CONTROLLER. 529
This model includes DOS 3.3 16 sector

TOP FIVE SELLERS

Language System W/Pascal	425
Silentype Printer W/Interface	549
Hayes Micromodem II	319
Videx Videoterm 80 w/graphics	335
Z-80 Microsoft Card	299

APPLE COMPUTER INC.

Disk II Drive Only	445
Integer or Applesoft II Firmware Card	155
Graphics Tablet	649
Parallel Printer Interface Card	155
Hi-Speed Serial Interface Card	155
Smarterm 80 Column Video Card	335

MOUNTAIN COMPUTER INC.

Music System (16 Voices)	479
A/D + D/A Interface	319
Expansion Chassis	555
Introl/X-10 System	249
Clock/Calendar Card	239
Supertalker SD-200	249
Romplus+ Card	135
Romwriter Card	155

CALIFORNIA COMPUTER SYSTEMS

Clock/Calendar Module	109
GPIB IEEE-488 Card	259
Asynchronous Serial Interface Card	129
Centronics Parallel Interface Card	99
We carry all CCS hardware.	Please call

MISC. APPLE HARDWARE

16K Ram Card Microsoft	189
ABT Numberic Keypad(old or new kybrd)	115
ALF 3 Voice Music Card	229
Alpha Syntauri Keyboard System	1399
Corvus 10MB Hard Disk	CALL
Lazer Lower Case Plus	50
Micro-Sci Disk Drives	CALL
SSM AIO Serial/Parallel Card A&T	189
Sup-R-Terminal 80 Col. Card	339
SVA 8 inch Floppy Disk Controller	345
Versawriter Digitizer Pad	229

WE HAVE MANY MORE ACCESORIES FOR THE APPLE II IN STOCK— PLEASE CALL OR WRITE FOR A PRICE LIST.

ATARI

MODEL
800 16K
$799

Atari 400 16K	499
810 Disk Drive	499
410 Program Recorder	69
850 Interface Module	175
822 Thermal Printer (40 col)	369
825 Printer (80 col)	795
Atari 16K Ram Module	155
Atari Light Pen	65

We stock all Atari accessories & software, please call for more info.

PRINTERS

Anadex DP-9500 W/2k Buffer	1375
Anadex DP-9501 W/2K Buffer	1450
C. Itoh Starwriter 25 CPS	1750
C. Itoh Starwriter 45 CPS	2450
Centronics 737	825
Epson MX-70 W/Graphics	449
Epson MX-80 132 Col	620
Paper Tiger IDS-445 W/Dot Plot	749
Paper Tiger IDS-460 W/Dot Plot	1195
Paper Tiger IDS-560 W/Dot Plot	1495
Qume Sprint 5/45 Daisywheel	2550
Silentype w/Interface for Apple II	549
Watanabe Digiplot	1295

VIDEO MONITORS

Amdex/Leedex Video-100 12" BGW	139
Hitachi 13" Color	389
NEC 12" P31 Green Phospher	CALL
Panacolor 10" Color	375
Sanyo 9" BGW	179
Sanyo 12" BGW	255
Sanyo 12" P31 Green Phospher	295
Sanyo 13" Color	445

OHIO SCIENTIFIC

Challenger 4P	699
C4PMF (Mini Floppy System)	1599
CIP Model II	449
Sargon II (Disk or Cassette)	35
Fig Forth (Disk Only)	69

APPLE SOFTWARE

DOS Toolkit	65
Appleplot	60
Tax Planner	99
Apple Writer	65
Apple Post	45
D.J. Portfolio Evaluator	45
D.J. News & Quotes Reporter	85
Apple Fortran	165
Apple Pilot	129
DOS 3.3 Upgrade	49
Music Theory	45
The Controller Bus. Sys	519

MISC. APPLICATIONS PACKAGES

Visicalc	125
Desktop Plan II	169
CCA Data Management DMS	85
Easywriter Word Processor	225
ASCII Express	65
Super Text II	139
Programma Apple Pie	119
The Landlord Apt. Mgmt. Pkg	649
Peachtree Business Software	CALL
Tax Preparer by HowardSoft	89
Applebug Assem/Disassm/Editor	75
3-D Graphics By Bill Budge	53

GAMES

Flight Simulator	34
The Wizard and The Princess	32
Cosmos Mission (Space Invaders)	24
Sargon II Chess	32
Hi-Res Football	39
Adventure by Microsoft	27
Phantoms Five	39
Reversal (Othello)	34

PLEASE CALL OR WRITE FOR A COMPLETE SOFTWARE LIST.

ORDERING INFORMATION: Phone Orders invited using VISA, MASTERCARD, AMERICAN EXPRESS, DINERS CLUB, CARTE BLANCHE, or bank wire transfer. Credit cards subject to service charge: 2% for VISA & MC, 5% for AE, DC & CB. Mail Orders may send credit card account number (include expiration date), cashiers or certified check, money order, or personal check (allow 10 days to clear). Please include a telephone number with all orders. Foreign orders (excluding Military PO's) add 10% for shipping all funds must be in U.S. dollars. Shipping, handling and insurance in U.S. add 3% (minimum $4.00). California residents add 6% sales tax. We accept COD's under $500. OEM's, Institutions & Corporations please send for written quotation. All equipment is subject to price change and availability without notice. All equipment is new and complete with manufacturer warranty (usually 90 days). We cannot guarantee merchantibility of any products. We ship most orders within 2 days.

WE ARE A MEMBER OF THE BETTER BUSINESS BUREAU AND THE CHAMBER OF COMMERCE
SHOWROOM PRICES MAY DIFFER FROM MAIL ORDER PRICES.
PLEASE SEND ORDERS TO:
CONSUMER COMPUTERS MAIL ORDER 8314 PARKWAY DRIVE, GROSSMONT SHOPPING CENTER NORTH LA MESA CALIF. 92041

tured language like PASCAL, yet is easy to learn like BASIC. It is destined to become quite popular in the years to come. Already it is reported to be the official programming language in DENMARK.

The COMAL Information Package contains information about User Groups, Books, Articles, and Software that both current and prospective COMAL users should know about. And it contains instructions for Commodore PET*/CBM computer users on how to get a free COMAL compiler for their computer.

To receive the COMAL Information Package, simply send a Self Addressed Stamped Envelope to: The COMAL USERS GROUP, 5501 Groveland Terrace, Madison, WI 53716. Outside the United States, please include $2.00 for Air Mail Handling, $.50 to Canada. *PET is a trademark of Commodore.

SHORTAX Program Updated For 1981 Taxes

SHORTAX, a year around tax planning program, will now compute 1981 income and social security taxes as well as 1980 and 1979. The update is based on tax laws in effect as of January 1, 1981. The program will be revised whenever the tax laws are changed.

The SHORTAX program has also been modified to run on most types of CP/M systems that use Microsoft's BASIC-80 (MBASIC) release 5.0 or later, and will run on Apple computers with the appropriate CP/M modification. SHORTAX will also operate on the Radio Shack TRSDOS systems (Models I, II or III) and on Micropolis disk operating systems using the Micropolis disk extended BASIC. Use of the program requires a cpu with at least 48,000 bytes and at least one disk drive. Under CP/M a few systems may require 56,000 bytes of cpu memory. According to the company, the program is also being converted to run on a Pertec 2000 system and the Apple disk systems.

The SHORTAX program is designed for fast, interactive calculations of before and after tax simulations. As many as 20 complex tax computations can be simulated in as little as an hour. It can be used to quickly calculate the tax impact of incorporating a business, filing an amended tax return, investing in a tax sheltered investment or transferring income producing property to a college trust fund. The program calculates the federal income tax using the tax rate schedules, the income averaging method and the optional maximum tax formula.

Where applicable, it will compute the add-on minimum tax or the alternative minimum tax. For individuals, it will compute the employee social security tax or the self employment tax. The program will compute the applicable taxes for all individual filing statuses, for corporations and for accumulation trusts.

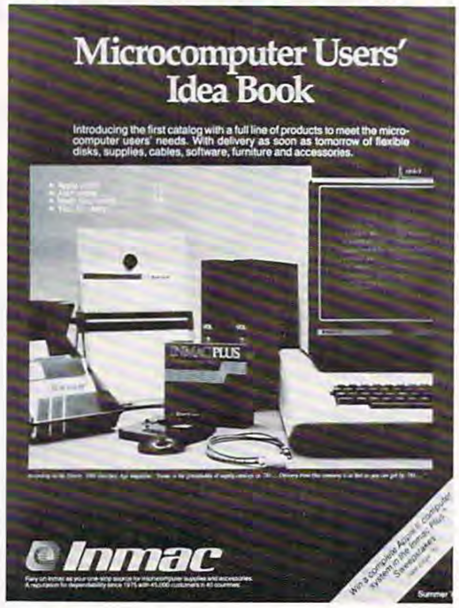
Although designed primarily for professional tax and financial advisors, it is self documented and can be used by business owners or investors who are reasonably familiar with the tax laws. The program was designed by Vernon K. Jacobs, a CPA and CLU who practices as a financial and tax consultant. He is also the editor of **Tax Angles**, a tax strategy newsletter, and is the author of **The New Taxpayer's Counterattack**.

The retail price of the program is $500 and annual updates are $300. The user manual is $15. For further details write to Syntax Corporation, Box 8137-P, Prarie Village, KS, 66208 or call at (913) 362-9967. Dealer and O.E.M. inquiries are welcome.

New Inmac Catalog For Microcomputer Users: A "Micro-Offspring"

Santa Clara, CA, June 18 — The first catalog dedicated to meeting supply, accessory, and cable needs of microcomputer users is now available from Inmac. Called **The Microcomputer User's Idea Book**, the 32-page publication lists over 1000 products — from software packages and CRTs to flexible disks, printer ribbons, and EDP-tailored furniture.

Featured for the first time by Inmac are several peripherals and software packages. These include the recently introduced high-speed Centronics 739 printer, a line of Sanyo data display monitors, and VisiCalc and DB Master



software for Apple users.

The Idea Book has been designed to make product selection quick and simple. Separate sections show complete supplies, accessories, and cables for Apple, Atari, TRS-80, and Northstar. In addition, extensive cross-referencing shows compatibility with many other systems.

Other sections list products by functions — storage, preventative maintenance, safety and security, lightning, and productivity.

For more information write to:

Inmac
Department 12
2465 Augustine Drive
Santa Clara, CA 95051

Utility "Translation" For Apple Owners

Mint Software has announced the release of a utility for users of the Applewriter, Supertext and Superscribe word processing systems. Super Apple Textwriter allows the user to:

1. Convert files generated under any one of these three word producers into files accessible by the other two. For example, users wishing to convert files generated by Applewriter into files accessible by Supertext may do so with ease.

2. Convert standard text files into files accessible by either Supertext or Applewriter or Supertext files into standard text files.

This utility is of particular value to those users who wish to use their word processing system to edit information obtained from one of the communications networks (e.g. The Source), as well as those who wish to use a modem to transmit over the phone lines files created by one of the word processors. It is even possible to develop and edit a BASIC program utilizing the editing features of a word processor, and then use Super Apple Textwriter to convert the resultant file into a text file which may then be EXECed into mmemory.

Super Apple Textwriter retails for $49.95. It may be ordered from Mint Software, 6422 Peggy Drive, Baton Rouge, Louisiana 70806. Phone (504) 766-2318. Dealer inquiries are invited.

The International Microcomputer Software Directory

This directory provides all microcomputer users, professional and amateur, with a primary reference source of microcomputer software for all applications and systems.

It is drawn from a large database that is continually being updated with information collected from all parts of the world through offices in Britain and America. The directory has three main sections:

1. **System Classification** — for the user limited to a particular system. Programs compatible with each respective system are listed in subject and price order.

2. **Subject Classification** — for the user who can use any system or who has not yet purchased a system. Programs are listed under subject areas in price order giving information as to

compatible systems.

3. Software House Classification — for the user wishing to buy from a particular software house. Programs are listed in ISPN (international standard program number) order, cross-referenced with the other two sections and giving full details (where available) as to date of release, warranties, distributors, distribution medium (cassettes, disk, etc.) source code, compatible systems, special configurations needed, special features, limitations and prices.

The complete directory is available for $28.95 (plus $2.95 postage and packing). Also available at $14.95 (plus $1.95 postage and packing) are system specific directories that are extracted and cross-referenced from the main data base. These list those programs written specifically for the Apple, PET, TRS 80, and CP/M. These publications will be available in June, 1981.

An on demand search facility will be available from July 1981, which will provide up to the minute information on new software available for a particular application or system.

For more information contact:

Imprint Software
US-420 South Howes
Ft. Collins, CO 80521
(303) 482-5574
UK-16 Milton Avenue
Highgate, London N6
Tel ol-348-3998

New Autodial-Autoanswer Modem For The Atari® 400/800 Computer

Redmond Wa. — The Microperipheral Corporation has just announced a new peripheral for the ATARI® Model 400/800 Personal Computer System. The MICROCONNECTION™ is a direct connect modem which eliminates the need for acoustic coupled devices. An AUTODIAL/AUTOANSWER option permits dialing or responding to other computers automatically. The option is available for use with either the Model 400 or 800, with or without the ATARI® 850 Expansion Interface. When used with the Model 850, it is directly interchangeable with the ATARI® modem.

Applications for this new product are virtually unlimited. For example, by using the AUTODIAL feature, the unattended 400/800 can send messages. text or other data to a host computer. The ATARI® 400/800 can also be set up to act as an unattended host. The modem will automatically answer the telephone and permit the 400/800 to capture data being sent to it. Typical applications include small business bulletin boards and message centers or automatic downloading of programs and other data.

The MICROCONNECTION™ for the ATARI® is Bell 103 compatible and operates in the originate or answer mode at 300 baud. Models for use without the 850 Interface incorporate a socket (DB-25) for connection to any serial printer capable of operation at 300 baud.

Another significant feature is the provision for on-line data storage. An inexpensive, voice grade cassette recorder can be plugged into the MICROCONNECTION™ and will "transcribe" on-line communications for later playback. A special version, which is compatible with European telecommunications standards, is also available.

The unit measures 7.7 inches wide by 5.5 inches deep by 1.7 inches high and weighs less than one pound. The price, complete with autodialing terminal software, power source and connecting cable (but without options), is $199.50. The AUTODIAL/AUTOANSWER OPTION IS $79.00 extra.

For more information, contact The Microcoperipheral Corporation, 2643 151st Place N.E., Redmond, Wa. 98052, telephone (206) 881-7544.

ATARI, ATARI 400/800 and ATARI 850 are trademarks of Atari, Inc., Sunnyvale, CA, a Warner Communications Co.

Atari I/O Package

The MOSAIC I/O Package can help give the ATARI computer direct ties to the real world. The four ports on the front of the ATARI computer connect directly to a PIA for use as output as well as input ports. Now ATARI owners can build custom program controllers, interface to home control circuits, or use any hardware the imagination can devise.

The I/O package comes with 4 – nine pin connectors, 4 – twelve inch lengths of nine conductor ribbon cable, and complete instructions for their use. The documentation includes examples of home-built program controllers, how to access the ports through BASIC commands, shadow registers, or directly, and how to set up and address the ports for output. Order number H309. Price $18.00.

MOSAIC ELECTRONICS
P.O. Box 748
Oregon City, OR 97045

New Product releases are selected from submissions for reasons of timeliness, available space, and general interest to our readers. We regret that we are unable to select all new product submissions for publication. Readers should be aware that we present here some edited version of material submitted by vendors and are unable to vouch for its accuracy at time of publication.

New Literature Microcomputer Products

A new catalog from ELECTRIC SPECIALISTS presents their line

of MICROCOMPUTER interference control products. Protective devices are also included.

Descriptive sections are included which outline particular problems. Suggested solutions are given. Typical applications and uses are also outlined. Request Catalog 811.

Electronic Specialists, Inc.
171 South Main Street
Natick, MA 01760
Phone: (617) 655-1532

Advertiser's Index

The Editor's Feedback:

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**____________
☐ **Don't yet have one...**

During the next year I expect to buy:
☐ **computer** ☐ **printer**
☐ **disk drive** ☐ **other peripherals**

Content:
Best Article This Issue (page #, title)

Other suggestions:

11 12

The 6502 Resource Magazine
COMPUTE! PET•ATARI•APPLE OSI•KIM•SYM•AIM

For Fastest Service, Call Our **Toll-Free** US Order Line **800-345-8112** In Pennsylvania call 800-662-2444

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**____________
☐ **Don't yet have one...**

☐ Please enter my 1 year (12 issue) subscription to **COMPUTE!**
☐ New subscription ☐ Renewal subscription
☐ $20.00 U.S. subscription
☐ $25.00 Canada and international surface mail: Payment in U.S. funds, drawn on U.S. bank or money order
☐ Bill me (U.S. only). $1.00 billing fee

Name, Address: ________________________

Charge my: ☐ Visa ☐ MC
Number________________ Expires_____/_____/

11 12

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

Place Stamp Here

COMPUTE! Magazine
Post Office Box 5406
Greensboro, NC 27403

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

Author Feedback:

____

____

____

____

____

____

____

____

____

____

COMPUTE!

Place
Stamp
Here

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008
USA

The 6502 Resource Magazine
COMPUTE! PET•ATARI•APPLE OSI•KIM•SYM•AIM

For Fastest Service,
Call Our **Toll-Free**
US Order Line
800-345-8112
In Pennsylvania call 800-662-2444

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**__________
☐ **Don't yet have one...**

☐ Please enter my 1 year (12 issue) subscription to **COMPUTE!**
☐ New subscription ☐ Renewal subscription
☐ $20.00 U.S. subscription
☐ $25.00 Canada and international surface mail. Payment in U.S. funds, drawn on U.S. bank or money order
☐ Bill me (U.S. only) $1.00 billing fee

Name, Address: ______________________________

____

____

____

Charge my: ☐ Visa ☐ MC
Number____________________ Expires_____/_____/

11 12

Place
Stamp
Here

To:

For Fastest Service,
Call Our **Toll-Free**
US Order Line
800-345-8112
In Pennsylvania call 800-662-2444

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**__________
☐ **Don't yet have one...**

☐ Please enter my 1 year (12 issue) subscription to **COMPUTE!**
☐ New subscription ☐ Renewal subscription
☐ $20.00 US subscription
☐ $25.00 Canada and international surface mail. Payment in US funds, drawn on US bank or money order
☐ Bill me (US only) $1.00 billing fee

Name, Address: ____________________

Charge my: ☐ Visa ☐ MC
Number________________ Expires_____/_____/

11 12

YES! I would like to order The LIMITED GOLD EDITION. ☐ ✓

Please fill out the information below. We will then send you a reserved purchase authorization. Your nearest dealer will then be notified to put aside one copy for you. (Please note that the **Limited Gold Edition** is in fact, a limited edition. We can only guarantee delivery to persons filling out this card, or the reserved purchase card at their local dealer. Should the edition be over-subscribed, orders will be fulfilled on a "first come" basis only.)

YOUR ADDRESS:
Name __________
Address __________
City __________
State _____ Zip _____
Telephone # __________

YOUR DEALERS ADDRESS:
Store Name __________
Address __________
City __________
State _____ Zip _____
Telephone # __________

TYPE OF SYSTEM AND MEDIA YOU WOULD LIKE:
☐ TRS-80 Tape ☐ TRS-80 Disk ☐ APPLE Disk ☐ ATARI Tape ☐ ATARI Disk

OVER →

TO ORDER THE LIMITED GOLD EDITION

1 Fill in the information on the back.
2 Detach this card and place it in an envelope.
3 Enclose a stamped self-addressed envelope.
4 Mail to:

ADVENTURE INTERNATIONAL
Limited Gold Edition
Box 3435
Longwood, Florida 32750

******* DO NOT ENCLOSE ANY PAYMENT AT THIS TIME *******

★★★★★★★★★★★★★★★★★★★★★★★★★★★

If you would just like our free catalog of 150 software packages, fill in your name and address on the back and check here ☐ ✓

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

IF YOU CAN WAIT A MINUTE, WE CAN SAVE YOU $1,000.

With the Starwriter™ Daisy Wheel 25 cps printer from C. Itoh.

A business letter, written on a 45 cps word-processing printer, might take about two minutes to print.

With the Starwriter, it might take closer to three.

The typical 45 cps printer retails for about $3,000.

But the Starwriter 25 retails for about $1,895—thus saving you about $1,000.

And therein lies the biggest difference between the Starwriter 25 and the more expensive, daisy wheel printers.

The Starwriter 25 comes complete and ready-to-use, requiring no changes in hardware or software. It uses industry-standard ribbon cartridges, and it's "plug-in" compatible to interface with a wide variety of systems, to help lower system-integration costs.

Using a 96-character wheel, it produces excellent letter-quality printing on three sharp copies with up to 163 columns, and offers the most precise character-placement available, for outstanding print performance.

C. Itoh's warranty;

3 months on parts and labor, supported by one of the best service organizations in the industry.

$1,000 OFF

Leading Edge Products, Inc., CM-5
225 Turnpike Street,
Canton, Massachusetts 02021

Dear Leading Edge:
I'd like to know more about the Starwriter, and how spending a minute can save me a grand.
Please send me the name of my nearest dealer.

Name ____________
Title ____________
Company ____________
Street ____________
City ____________
State ______ Zip ______
Phone: Area Code ____________
Number ____________

LEADING EDGE.

Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021
Dealers: For immediate delivery from the Leading Edge Inventory Bank™ call toll free 1-800-343-6833
In Massachusetts, call collect (617) 828-8150. Telex 951-624