THE #1 MAGAZINE FOR ATARI COMPUTER OWNERS

A.N.A.L.O.G. COMPUTING

T.M.

NOVEMBER 1988 ISSUE 66

DISK VERSION $12.95

FDC 50077

P
LAD

★ **POPS:** Stereo Sound for Your Atari

Also:

★ Boot Camp and Game Design Workshop

SLAVE II

★ Nimral's Grace— The Adventure Continues

Boot UP

BIG SAVINGS!

You can save time, and save a lot of money by subscribing to A.N.A.L.O.G. Computing Magazine. Save $14 off the cover price with the convenience of having A.N.A.L.O.G. delivered directly to your door before it even hits the newsstands. To order use the handy postage-paid order card located in the back of this magazine!

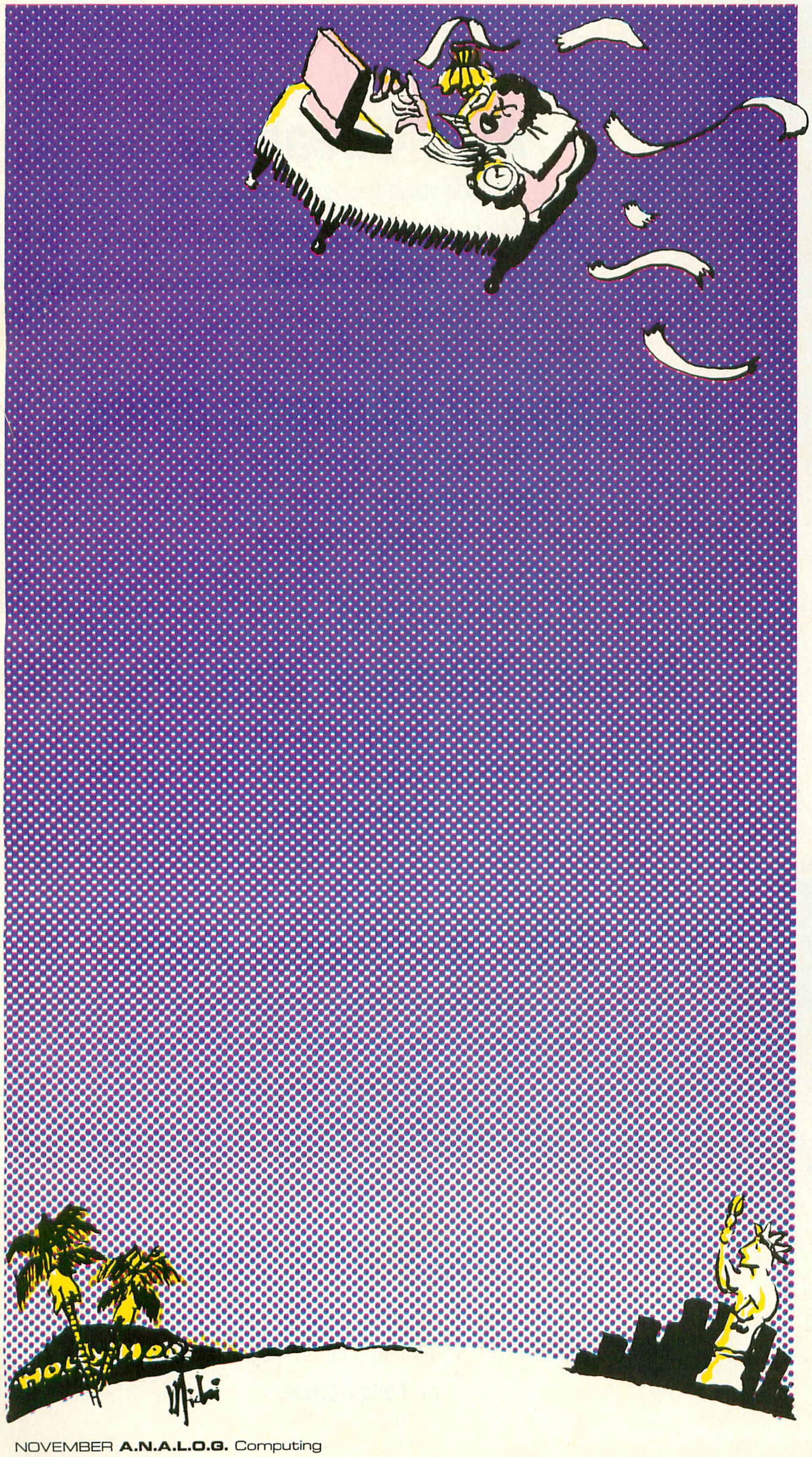
1 YEAR FOR ONLY $28

SAVE $14 OFF THE COVER PRICE

1 YEAR *WITH DISK* ONLY $79

new low pric

Editorial



This month's editorial comes to you from high up (about 37,000 feet) in the friendly skies, as I jot down some thoughts about what it's been like for the past few months, commuting from my home in Massachusetts to ANALOG's new offices in Los Angeles. Believe me, it hasn't been easy.

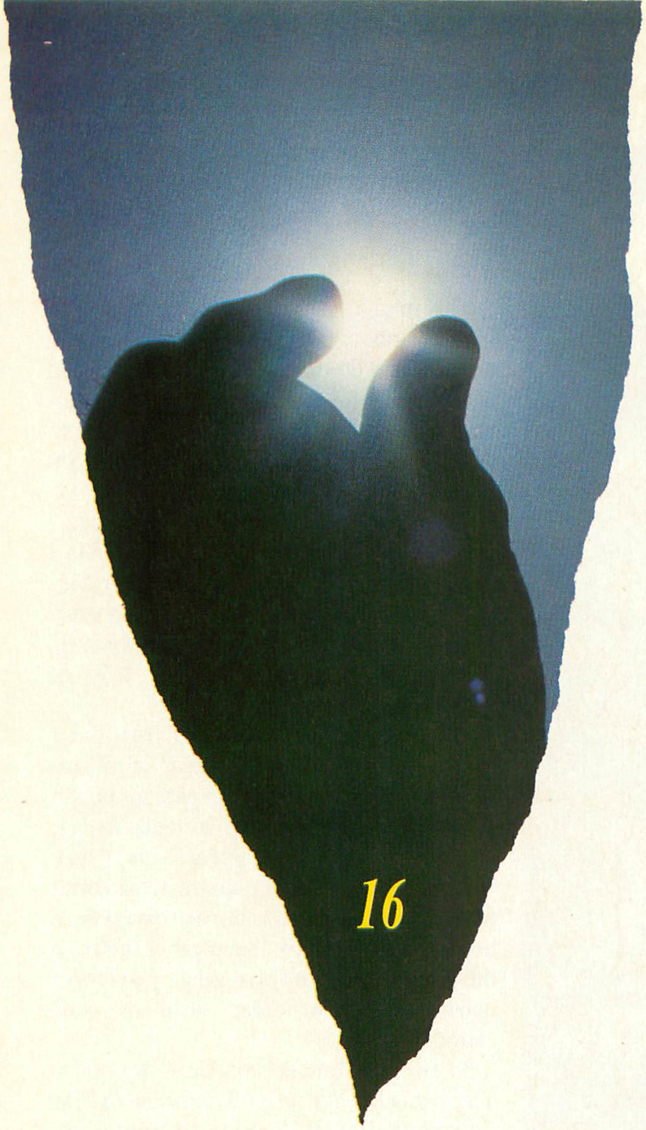
No sooner have I set foot at Logan International Airport in Boston than it seems I have to turn around and make another beeline for the West Coast, banging out a story or an editorial on my laptop en route to meeting publishing deadlines in the City of Angels.

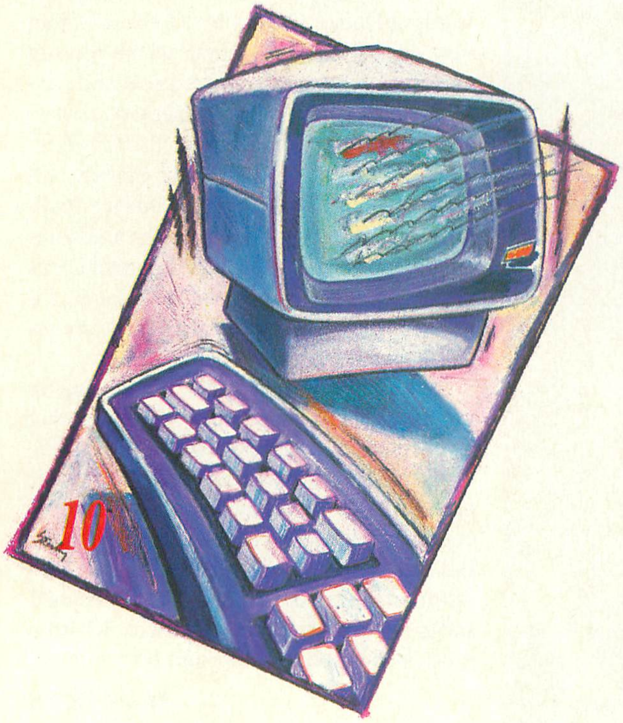
The wear and tear of rushing from coast to coast is taking its toll: I've acquired a haggard, desperate look from not getting enough sleep; my body suffers from terminal jet lag; my fiancée barely recognizes me ("Lee who?") and a sense of cultural schizophrenia has taken root due to the extreme polarities of the laid-back Southern California lifestyle and the East Coast grind. I no longer know whether to order sushi or clam chowder!

So I've made a decision. In order not to burn out before my time ("Publishing Tycoon Commits Hara-Kiri") and to be able to provide the kind of editorial guidance ANALOG needs to maintain the high standards we've worked so hard to reach, I'm heeding the words of Horace Greeley to "Go west." From now on, I'll make my home where the sun shines all year-round, where snow and subzero temperatures are only a memory. Instead of Boston Common, I'll roam Griffith Park. Instead of catching the Sox at Fenway, I'll down my foot-long hot dogs at Dodger Stadium (but still root for Wade Boggs and company). In place of the rocky Massachusetts coastline and chilly waters of the Atlantic, I'll sink my feet into Venice Beach's sandy shores and bodysurf in the moderate Pacific.

And so, like the Clampetts, I'm loading up my truck and moving to Beverly. Hills, that is. Swimming pools. Movie stars. . . .Well, maybe it's not all it's cracked up to be. I'll have to battle gridlocked freeways, out-of-control smog and the occasional earthquake. But that's a small price to pay to liberate myself from the red-eye express, bad airline food and in-flight movies. Massachusetts, it's been great, but so long. California, here I come!

by Lee Pappas





Conte

FEATURES

nts

NOVEMBER 1988
ISSUE 66

ANALOG COMPUTING STAFF

Publisher
LEE H. PAPPAS

Executive Editor
CLAYTON WALNUM

Art Director
MICHIO TSUZUKI

Managing Editor
DEAN BRIERLY

East Coast Editor
ARTHUR LEYENBERGER

West Coast Editor
CHARLES F. JOHNSON

Contributing Editors
LEE S. BRILLIANT, M.D.; MICHAEL BANKS; FRANK COHEN; ANDY EDDY; MAURICE MOLYNEAUX; STEVE PANAK; CRAIG PATCHETT; ROBIN SHERER; KARL E. WIEGERS MATTHEW J. W. RATCLIFF

Entertainment Editors
ANDY EDDY, DAVID PLOTKIN

Cover Photography
MICHIO TSUZUKI

Illustrations
MICHIO TSUZUKI
STEPHEN STERLING

Copy Chief
KATRINA VEIT

Copy Editors
ANNE DENBOK
SARA BELLUM
PAT ROMERO
KIM TURNER

Chief Typographer
KLARISSA CURTIS

Typographers
JUDY VILLANUEVA
DAVID BUCHANAN

Contributors
DAVE ARLINGTON
LEROY BAXTER
JIM PATTERSON
BRAD TIMMINS
HOWARD WEN

Vice President, Production
DONNA HAHNER

National Advertising Director
JE PUBLISHERS REPRESENTATIVE
(213) 467-2266
(For regional numbers, see map)

Advertising Production Director
JANICE ROSENBLUM

Advertising Manager
PAULA THORNTON

Subscriptions Director
IRENE GRADSTEIN

Vice President, Sales
JAMES GUSTAFSON

Where to Write

All submissions should be sent to: **ANALOG Computing,** P.O. Box 1413-M.O., Manchester, CT 06040-1413. All other editorial material (letters, press release, etc.) should be sent to: Editor, **ANALOG Computing,** 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210.

Correspondence regarding subscriptions, including problems and changes of address, should be sent to: **ANALOG Computing,** P.O. Box 16927, North Hollywood, CA 91615, or call (818) 760-8983.

Correspondence concerning a regular column should be sent to our editorial address, with the name of the column included in the address.

We cannot reply to all letters in these pages, so if you would like an answer, please enclose a self-addressed, stamped envelope.

An incorrectly addressed letter can be delayed as long as two weeks before reaching the proper destination.

Advertising Sales

Address all advertising materials to:
Paula Thornton — Advertising Production
ANALOG Computing
9171 Wilshire Blvd., Suite 300
Beverly Hills, CA 90210.

Permissions

No portion of this magazine may be reproduced in any form without written permission from the publisher. Many programs are copyrighted and not public domain.

Due, however, to many requests from Atari club libraries and bulletin-board systems, our new policy allows club libraries or individually run BBSs to make certain programs from **ANALOG Computing** available during the month printed on that issue's cover. For example, software from the July issue can be made available July 1.

This does not apply to programs which specifically state that they are not public domain and, thus, are not for public distribution.

In addition, any programs used must state that they are taken from **ANALOG Computing** Magazine. For more information, contact **ANALOG Computing** at (213) 858-7100, ext. 163.

Subscriptions

ANALOG Computing, P.O. Box 16927, North Hollywood, CA 91615; (818) 760-8983. Payable in U.S. funds only. U.S.: $28-one year, $54-two years, $76-three years. Foreign: Add $7 per year. For disk subscriptions, see the cards at the back of this issue.

Authors

When submitting articles and programs, both program listings and text should be provided in printed and magnetic form, if possible. Typed or printed text copy is mandatory, and should be in upper- and lowercase with double spacing. If a submission is to be returned, please send a self-addressed, stamped envelope.

For further information, write to **ANALOG Computing,** P.O. Box 1413-MO, Manchester, CT 06040-1413.

JE Publishers Representative
6855 Santa Monica Blvd., Suite 200
Los Angeles, CA 90038

Los Angeles	— (213) 467-2266
San Francisco	— (415) 864-3252
Chicago	— (312) 445-2489
Denver	— (303) 595-4331
New York City	— (212) 724-7767

ANALOG Computing (ISSN 0744-9917) is published monthly by L.F.P., Inc., 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210. © 1988 L.F.P., Inc. Return postage must accompany all manuscripts, drawings, photos, disks, etc., if they are to be returned, and no responsibility can be assumed for unsolicited materials. All rights reserved on entire contents; nothing may be reproduced in whole or in part without written permission from the publisher. U.S. subscription: $28 for one year (12 issues), $52 for two years (24 issues), $76 for three years (36 issues). Foreign subscription: Add $7 per year. Single copy $3.50 (add $1 for postage). Change of address: Six weeks advance notice, and both old and new addresses are needed. POSTMASTER: Send change of address to **ANALOG Computing** Magazine, P.O. Box 16927, North Hollywood, CA 91615. Second-class postage paid at Beverly Hills, CA, and additional mailing offices.

READER COMMENTS

A common cold?

I am writing this letter because my machine was recently infected with a virus. The damage resulting from the infection took three months to repair. The symptoms started when I was unable to write to the disks because they were reported as full by the operating system. This didn't seem logical since some of the disks had small data files on them which should not have caused a full-disk message to appear. I started using the backup disks, but they also became infected and thus damaged beyond repair. After weeks of trial and error, neither I nor my friends were able to discover what caused the problem. As a result of helping me, my friend's system also became infected via disks used on my system. How far the virus spread beyond my immediate group is problematic.

I frequently read your magazine and have not seen any articles forewarning your readers of software viruses. I eventually read a *Newsweek* magazine article which described the problem as a software virus. It was astounding to find the virus problem being discussed in a periodical which is dedicated to general news events, especially when the two computer magazines I read did not mention them. Since the *Newsweek* article, I have read articles in both the *Washington Post* and *Time*. By the way, the *Post* dedicated all of page 3 in the Sunday edition to the growing problem.

I think it is your responsibility to keep your readers informed about computer problems which could result in months of work being lost. The damage just to my system could buy a subscription to ten computer magazines for the next five years.

It is certain that many of your readers are as ignorant of the problem as I was. It is also evident that the problem is large enough to draw national media coverage. It would be a disservice to your readers not to give them information about this problem. Therefore I request that you keep your readers informed of software viruses, symptoms and cures.

—Ralph Allen
Arlington, Virginia

Action! translations

Is there any way an Action! compiled program could be converted to BASIC data statements and printed in your magazine? This would allow more of us Action!-less owners to enjoy some of those fine programs. I realize they are available on your disk version.

—Everett Rantanen
Milwaukee, Wisconsin

Any binary file may be converted to DATA statements for use in a BASIC loader or for typing in by M/L Editor. However, most Action! programs, after they've been compiled, are much too large to allow their printing in the magazine. We realize that many readers don't have Action!, but we feel that enough people are interested in the language to warrant the inclusion of an Action! program now and then. As you mentioned, most of the Action! programs (not all of them) are available on the disk version of the magazine. Also, they may be downloaded from ANALOG's Atari SIG on DELPHI.

Pen pal wanted

I am Polish, and I am 17 years old. I have an Atari 800XL computer and a 1050 disk drive. I want to collaborate with other Atari users. I also want to exchange Atari computer magazines. Can you help me?

—Artur Nowakowski
W. Wasilewskiej 5/8
08-110 Siedlce
Poland

Updating the GEnie update

There's a problem with Andy Eddy's piece on GEnie. I am not the SYSOP. I am one of the SYSOPs. To make it very clear:

There are two contracts for the Atari roundtables (RTs). Darlah holds one, and Atari Corp. holds the other. On Atari's behalf, I am the manager of the Atari areas, with 20 members of Atari's staff currently online there. Darlah, however, supervises all the activities of the assistant SYSOPs (Marty Albert, Mark Booth, Sandy Wilson, Holly Stowe, Craig Thom and Atarians John Townsend and Dan Scott).

Darlah works harder for the GEnie RTs than any SYSOP I've ever seen. She deserves credit, and listing me as SYSOP is not fair. I hope you can put something in a future issue to fix this. Thanks.

—Neil Harris
Atari Corp.

I will be the first to acknowledge the work that Darlah has done to make the Atari RTs what they are. Her contributions are everywhere. What I meant to say in the article was that you had taken over the management of the Atari RTs, but it didn't come out that way. . .the sun reflected off my monitor. . .I tripped on a deadline. My apologies to Darlah and her associates. —Andy Eddy

Train Crazy error report

The following lines were accidentally deleted from the end of Listing 2 of Issue 63's Train Crazy. *Sorry about that.*

```
JI 4007 FOR T=2 TO 17:V=V+128:FOR O=0 TO
   8:NEXT O:SOUND 1,V,8,6:POSITION T,1:?
   #6;" OveR":NEXT T:POSITION 17,1
JX 4008 ? #6;"        ":POSITION 14,0:? #6;"
   OveR"
NB 4010 POKE 559,42:POSITION 7,0:? #6;"GA
   ME OveR  ":? #6;"":SOUND 1,0,0,0
AW 4030 ? #6;"          sCORe ";SC+215:? #6;"
   "
FK 4040 ? #6;"          PrESS STArT":? #6;""
QW 4045 ? #6;"        fOR a NeW GaME":? #6;"
   ":POKE 559,42:GOSUB 4050
ZX 4047 FOR YY=1 TO 15:POSITION 6,4:? #6;
   "           ":FOR T=1 TO 30:IF PEEK(53
   279)=6 THEN RUN
PL 4048 NEXT T:POSITION 6,4:? #6;"PrESS S
   TArT":FOR T=1 TO 30:IF PEEK(53279)=6 T
   HEN RUN
RY 4049 NEXT T:NEXT YY:RUN
GZ 4050 IF SC>15000 THEN POSITION 4,8:? #
   6;"EXCELLENT SCORE":RETURN
JW 4051 IF SC>10000 THEN POSITION 6,8:? #
   6;"GREAT SCORE":RETURN
GT 4052 IF SC>4000 THEN POSITION 7,8:? #6
   ;"good score":RETURN
PR 4053 IF SC<4000 THEN POSITION 7,8:? #6
   ;"try again":RETURN
LQ 5000 PM$(P1+Y,P1+Y+19)=P$(I,I+19)
XK 5001 I=58:X=X-2.4:S=STICK(0)
PG 5002 IF PEEK(53253)>0 THEN 10000
XT 5003 IF S=14 OR S=13 THEN 422
JF 5004 IF STRIG(0)=0 THEN X=X+1:GOTO 455
F5 5010 POKE 53249,X:IF X<53 THEN GOTO 30
   00
OT 5030 GOTO 5000
KG 9000 IF PEEK(53253)=2 THEN SC=SC+200:G
   OTO 9002
QF 9001 GOTO 620
WW 9002 RESTORE 9004+LP:READ O:POKE DM+25
   6*1+O,0:SOUND 1,50,10,8:IF LP=4 THEN C
   X=-8:BA=6
RT 9003 IF LP=5 THEN CX=-8:BA=6
GT 9004 DATA 20
LW 9005 DATA 58
NG 9006 DATA 97
HR 9007 DATA 133
LS 9008 DATA 168
IN 9009 DATA 206
YF 9010 POSITION CX+LP+LP,BA:? "Z"
DM 9011 I=3:V=120:ZX=ZX+3:POSITION 30-TT+
   ZX,2:? "nnn":O=0:FOR Z=Y TO 37 STEP 2:
   V=V+10:SOUND 1,V,10,5
YY 9020 PM$(P1+Z,P1+Z+19)=P$(I,I+19):NEXT
    Z:I=58
ON 9025 PM$(P1+34,P1+53)=P$(I,I+19):Y=34:
   GOTO 421
ZZ 10000 IF PEEK(53253)=8 AND LP=5 THEN 9
   00
GC 10001 IF PEEK(53253)>0 THEN 9000
ZY 20000 Q=Q+37:POKE DM+256*3+Q+2,0:POKE
   DM+256*3+Q+1,0:LP=LP+1:O=0:GOTO 421
GL 30000 RUN
```

8-Bit NEWS

NOVEMBER '88

Zap goes your XE

Ever wonder what would happen if a lightning bolt hit the power transformer outside your home one night? Five-hundred thousand volts of electricity can run through your XL/XE's power supply. The resulting fireworks will permanently remove your home computer from service. The way to avoid this problem is with the purchase of surge suppressors. These devices remove the nasty pops, clicks and spikes from the electricity powering your computer.

Surge suppressors usually plug into a wall plug and look more or less like an extension outlet. Your Atari computer's power supply plugs into the surge suppressor and receives a calm and steady supply of electricity. Surge suppressors are also good for your home stereo, microwave oven and answering machines, to name just a few. Unfortunately, this means buying many suppressors, one for each device you want to protect.

CPS Electronics has introduced a novel new idea in surge suppressors: One unit covers your entire house. The *EG 240R* is a whole home, residential, circuit breaker surge suppressor that mounts to the panel of your home's circuit breaker box. The unit can also be fitted into fuse boxes. The EG 240R can arrest a surge of up to 20,000 amps, which is enough to send your computer into orbit. CPS guarantees the unit with a three-year warranty. The list price is $89.95, and it is now available.

CPS Electronics
4151-112th Terrace N.
P.O. Box 2460
Pinellas Park, FL 34290-2460
(800) 237-6010

8-Bit NEWS

Anyone call a doctor?

Mad Scientist Software produces medical eduction software to teach medical principles to students and hospital staff. Their latest offering is the *Advanced Cardiac Life Support* (ACLS) package. A four-disk series, the ACLS system covers EKG training, cardiac arrest simulations, ACLS terminology and protocols, and a general quiz to prepare students for the ACLS certification test. The ACLS package costs only $109 and is now available for the XE/XL.

Mad Scientist Software
2063 North 820 West
Pleasant Grove, UT 84062
(801) 785-3028

Stripless XE/XL

Recently Artworx Software released *Strip Poker II* for the Atari ST, Apple IIGS and IBM-PC. You might remember Strip Poker as a fun and inventive game where your computer opponent reveals more than just his/her hand of poker cards. Artworx seems to have chosen to not produce an XE/XL version of this quality product. ANALOG encourages its readers to write to Artworx to change their mind about the Atari 8-bit market.

Artworx Software
1844 Penfield Road
Penfield, NY 14526
(800) 828-6573

AL/65 development tools

Omega Soft has announced a new development system for the 8-bit Atari home computer. Unlike most assemblers, *AL/65* compiles 6502 source code into relocatable code, or code which can be used in other programs. The editor supports full-screen editing, macros and custom character sets.

AL/65 is a complete development system for only $44.95. The package includes an assembler, linker, editor and system utility software. AL/65 also comes with a command-line interpreter (CLI), which allows developers to use typed commands to maintain files, launch programs and modify the development environment.

Omega Soft
P.O. Box 139
Harrelis, NC 28444
(919) 532-2359

Mindscape signs SSI

In a move to increase its audience, Mindscape has signed an exclusive publishing agreement with Strategic Simulations Inc. (SSI). The agreement gives Mindscape rights to publish SSI's backlist of strategy/simulation titles. Well-known classics like *Fortress*, *Battalion Commander*, *Nam* and *Geopolitique 1990* are the first SSI programs to be published under the new Thunder Mountain label.

SSI started in the computer war-game market in 1980, and has held almost half of the war-game market since. Mindscape has also licensed *Cartel and Cutthroats*, *Combat* Leader, *Galactic Gladiators* and *Queen of Hearts*. Most of these titles will be available for the Atari 8-bits.

Mindscape
3444 Dundee Road
Northbrook, IL 60062
(312) 480-7667

Daisy-Dot II

Roy Goldman has created *Daisy-Dot II*, a powerful printing system for 8-bit Atari computers. The software package allows printing of near-letter-quality text with Epson and Star compatible printers. Based on the original Daisy Dot, the new system offers higher output quality and new formatting features. Fourteen fonts are included with Daisy-Dot II, including Roman2 (Times Roman), Ohio (Geneva), Senator (Helvetica), and Block2 (Modern).

Daisy-Dot II has been put into the public domain; the entire system can be found on DELPHI, GEnie and CompuServe. The double-sided disk and 24-page manual may be purchased from Roy Goldman for only $10.

The *Daisy-Dot II Accessory Disk* is also available and is priced at only $5. It comes with *TextPro*, a word processor for text entry, eight additional fonts and a utility program that allows Daisy-Dot II to work with the *AtariWriter*.

Roy Goldman
2440 South Jasmine
Denver, CO 80222

Branch Software sprouts new titles

A new company is producing low-cost, high-quality software for the 8-bit market. Branch Software has released several new titles, all designed to work on any 48K Atari 8-bit computer with DOS 2.0 or 2.5 installed.

In *Blockaid* you use your shield to defend yourself against the fiery ball of Zieweunthu. After learning how to pronounce Zieweunthu, you can destroy all the colorful blocks and advance into the next of 40 rooms. The game includes a *Blockaid Construction Set*, so you can custom design additional screens. The game even lets you insert your custom screens into the Blockaid game for future generations to enjoy.

Trivia Quiz is the improved version of the public-domain trivia game of the same name. The new version is filled with ready-to-use questions, with more companion data disks on their way.

In *Agent 16*, a role-playing game, three giant text adventures are stored on one disk. The central character, Agent 16, is a super-secret spy that tumbles from one deadly challenge to another. A special menu loader lets you see all the instructions and background information needed to play Agent 16.

All of the new titles are currently available and have a list price of less than $20.

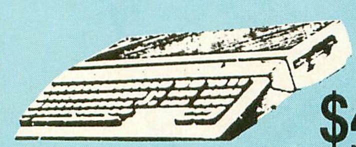
Branch Software
2750 Friday Lane
Cocoa, FL 32926
(407) 631-7149

BUY•SELL•TRADE BUY•SELL•TRADE BUY•SELL•TRADE

Computer Repeats, Inc.

UNBELIEVABLE DEALS EVERYDAY! ALL ORDERS PROCESSED IN 24 HRS!

Atari 1040ST



$449 NEW

with trade-in of 130XE, 1050, 1702 Mon., NP-10 Printer, modem
$CALL for your system

Products - Used

520ST Computer	$335	1040ST Computer	$585
SF354 Drive	$99	130XE Computer	$119
1200XL Computer	$59	800XL Computer	$69
400-16K Computer	$29	Hayes Smartmodem	$49
1050 Drive	$139	Atari 1027 LQ Printer	$79
Atari 1025 Printer	$79	Okimate 10 w/PIP	$99
ATR-8000 64K, Slaves	$199	Slave Drives from	$35
Atari 850 Interface	$69	Commodore 1702	$135
Koala Touch Tablet	$35	Software/Books from	$1

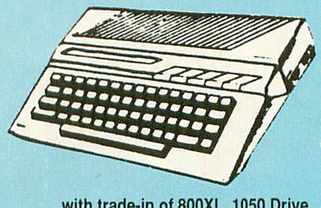
Products - New

520ST FM CPU	$499	520 ST FM color sys	$819
512K RAM FM Instlld	$199	Atari SF314 Drive	$219
SM124 mono Monitor	$159	SC1224 color Mon.	$325
130XE Computer	$249	XF-551 Drive	$179
1802C Color Monitor	$189	Magnavox 80 Mono	$99
Avatex 1200HC	$99	Avatex 2400HC	$199
Atari SX212 Modem	$89	Happy Rev. 7.1	$99
256K upgrades 0k from	$35	R-Time 8 Cartridge	$49
Star NX-1000 144cps	$189	Star NX-2400 24 Pin	$389

$Cash for your equipment
Thousands of software & book titles
Plus, MUCH, MUCH MORE!

2017 13th Street Suite A
Boulder, CO 80302

Atari 130XE



$65 NEW

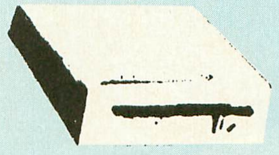
with trade-in of 800XL, 1050 Drive
$CALL for your system

Atari 520ST FM

$219 NEW

with trade-in of 800XL, 1050, 1702, NP-10 Printer, 1200 mdm.
$CALL for your system

Atari XF-551



$109 NEW

with trade-in of 1050 Drive
$CALL for your system

All references to trade-ins assume equipment to be in good working condition. Shipping/handling will be added to all prices. No additional charge for credit cards or COD. Mail order prices shown.
WE CHECK FOR CREDIT CARD THEFT!

1-303-939-8144
24 HR Modem Software Quotes: 1-303-939-8174

Authorized Dealers for
COMMODORE/AMIGA
and ATARI ST/XL/XE
Computers and Accessories.

VISA MASTERCARD DISCOVER AMERICAN EXPRESS COD

CIRCLE #102 ON READER SERVICE CARD.

Sterling

Atari Streamers

A*tari Streamers* is a machine-language utility which uses player-missile graphics to create fine-scrolling vertical-character displays. Excluding inverse characters (which are automatically unshifted), displays can be made up of any string of characters desired—that is, uppercase, lowercase and graphics characters in either single- or double-line resolution. Even custom-character sets can be displayed.

Atari Streamers has two entry points. The first entry point copies your string of characters into player-missile memory. It is called by the USR statement:

```
X=USR(1536,PMADDRESS,
STRINGADDRESS,FONT,LENGTH)
```

PMADDRESS is the address of the player you wish to put your character display in. STRINGADDRESS is the address of the string of characters that will be copied to player memory. The best way to store your character display is to define it as a string variable, and then use the ADR function to find its address. FONT is the address of the character set you wish to use. It should be set to 57344,which is the ROM address for the normal character set,but if you have an altered character set in memory, or if you wish to use the international character set (found only on the XL and XE computers), simply substitute the address of the character set you wish to display. The international character set is located at 52224. The last value, LENGTH, is the length of your string of characters. In double-line resolution, strings can be a maximum of 16 characters long. In single-line resolution, strings can be up to 32 characters long.

Excluding inverse characters, displays can be made up of any string of characters desired.

The second entry point of the routine will move everything in the player specified one byte up or one byte down with wraparound. It is called by the USR statement:

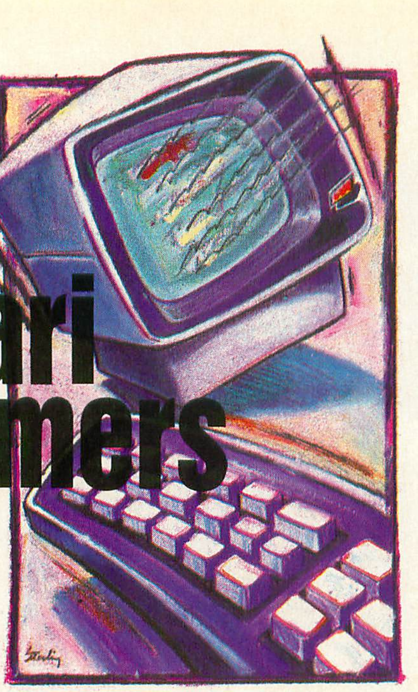
```
X=USR(1677,PMADDRESS,DIRECTION)
```

PMADDRESS is the address of the player you wish to move. DIRECTION is the direction you wish to move the player in. If DIRECTION equals zero, everything in the player will be moved up by one. If DIRECTION equals one, everything in the player will be moved down by one. When this routine is used in combination with the horizontal-position registers, you can easily move your character displays anywhere on screen and over any graphics mode.

The demo program displays and moves all four players in double-line resolution (if you wish to change the demo program to display in single-line resolution, substitute the lines in the REM statements). The four missiles are also displayed as a single player. This is not difficult since the missiles are mapped exactly the same way the players are mapped. The only difference is that the missiles have individual horizontal and collision registers. If you're going to combine the four missiles into a single player in your own program, remember to line them up in the correct order—that is, missile three, missile two, missile one, missile zero. They should be spaced two resolution lines apart. Poke location 623 with 17 to give all players priority over all playfields and to give all missiles their own color. You can change the missiles' color by poking to location 711.

by Brad Timmins

Atari Streamers



Listing 1: BASIC

```
SN 1 REM ATARI STREAMERS
XQ 2 REM BY BRAD TIMMINS
XW 3 REM Substitute the lines in REM
RL 4 REM statements for single-line
FH 5 REM resolution Players.
IR 6 REM COPYRIGHT 1988 BY ANALOG COMPUTI
   NG
AX 50 DIM P0$(16),P1$(16),P2$(16),P3$(16)
   ,MI$(16),CL$(16)
TW 51 REM DIM P0$(32),P1$(32),P2$(32),P3$
   (32),MI$(32),CL$(32)
DQ 60 ? CHR$(125):SETCOLOR 2,0,10:SETCOLO
   R 4,0,10
PX 70 GOSUB 500
TJ 80 A=PEEK(106)-8:REM A=PEEK(106)-16
FB 90 POKE 54279,A:PMBASE=256*A
EG 100 MISSILE=PMBASE+384:PM0=MISSILE+128
   :PM1=PM0+128:PM2=PM1+128:PM3=PM2+128
ZU 101 REM MISSILE=PMBASE+768:PM0=MISSILE
   +256:PM1=PM0+256:PM2=PM1+256:PM3=PM2+2
   56
IW 110 FONT=57344:REM ROM character set
LG 120 POKE 559,46:REM POKE 559,62
GM 130 POKE 53277,3:POKE 752,1:POKE 623,1
   7
UY 135 REM Position Players
JT 140 POKE 53248,60:POKE 53249,61:POKE 5
   3250,195:POKE 53251,196
RJ 144 REM Position Missiles
LM 145 POKE 53255,122:POKE 53254,124:POKE
    53253,126:POKE 53252,128
CN 150 POKE 704,128:POKE 705,134:POKE 706
   ,128:POKE 707,134:POKE 711,128
HI 155 REM Define a string of spaces
KT 156 REM the length of the players,
JG 157 REM to clear out Player-missile
MG 158 REM memory.
FC 160 CL$="                "
JS 161 REM CL$="
        "
QX 180 FOR T=0 TO 4
WJ 190 X=USR(1536,MISSILE+(T*128),ADR(CL$
   ),FONT,16)
DC 191 REM X=USR(1536,MISSILE+(T*256),ADR
   (CL$),FONT,32)
JY 200 NEXT T
YM 205 REM Define Strings
ID 210 P0$="   Streamers":P1$=P0$:P2$=P0$
   :P3$=P0$:MI$="      Atari"
RD 215 REM copy strings to Player memory
MV 220 X=USR(1536,PM0,ADR(P0$),FONT,12)
OO 230 X=USR(1536,PM1,ADR(P1$),FONT,12)
QH 240 X=USR(1536,PM2,ADR(P2$),FONT,12)
SA 250 X=USR(1536,PM3,ADR(P3$),FONT,12)
JQ 251 X=USR(1536,MISSILE,ADR(MI$),FONT,1
   0)
PX 255 REM Shift Players ONE and THREE
IC 256 REM down by one to create a
JC 257 REM shading effect.
KL 270 X=USR(1677,PM1,1)
LZ 290 X=USR(1677,PM3,1)
GL 295 FOR T=1 TO 500:NEXT T
VK 300 REM Move Missile and Players zero
IJ 305 REM and TWO
HI 306 FOR T=1 TO 250:NEXT T
SS 307 FOR I=1 TO 128:REM FORI=0 TO 255
IO 310 X=USR(1677,PM0,0)
KA 320 X=USR(1677,PM2,0)
CH 330 X=USR(1677,MISSILE,1):POKE 711,I
HK 340 NEXT I:GOTO 306
JQ 495 REM *****ATARI STREAMERS*****
TN 496 REM MACHINE LANGUAGE SUBROUTINE
EQ 500 FOR A=1536 TO 1766:READ B:POKE A,B
   :NEXT A
ZQ 505 RETURN
WW 510 DATA 104,104,133,204,104,133,203,1
   04,133,206,104,133,205,104,141,230,6,1
   04,141,229,6
TG 520 DATA 104,104,141,227,6,169,0,141,2
   28,6,169,32,141,231,6,173,47,2,201,62,
   240
GX 530 DATA 5,169,16,141,231,6,172,228,6,
   177,205,41,127,201,31,176,4,9,64,208,7
IK 540 DATA 201,95,176,3,56,233,32,141,22
   5,6,169,0,141,226,6,162,3,24,14,225,6
BM 550 DATA 46,226,6,202,208,247,24,173,2
   25,6,109,229,6,133,207,173,226,6,109,2
   30,6
PD 560 DATA 133,208,162,0,160,8,161,207,1
   29,203,230,207,230,203,136,208,245,238
   ,228,6,173
JQ 570 DATA 228,6,205,231,6,240,7,205,227
   ,6,240,2,208,164,96,104,104,133,204,13
   3,206
AQ 580 DATA 104,133,203,133,205,104,162,2
   55,160,255,173,47,2,201,62,240,4,162,1
   27,160,126
PM 590 DATA 104,201,1,240,19,160,0,177,20
   5,141,225,6,230,203,177,203,145,205,20
   0,202,208
VL 600 DATA 248,240,27,138,168,177,205,14
   1,225,6,136,177,203,141,226,6,138,168,
   173,226,6
YF 610 DATA 145,205,136,136,202,208,239,1
   38,168,173,225,6,145,205,96,0,0,0,0,0,
   0
XG 620 DATA 0,224,2,225,2,0
```

Listing 2

```
 ORG 1536

 ;------------------------------
 ;ATARI STREAMERS ML SUBROUTINE
 ;WRITTEN FOR THE MACRO ASSEMBLER
 ;--------------------------------
 ;THIS ROUTINE WILL TAKE A CHARACTER
 ;STRING AND COPY IT TO PLAYER
 ;MISSILE MEMORY.
 ;--------------------------------

 ;ZERO PAGE EQUATES

PLAYER: = 203
STRING: = 205
CHRGET: = 207
MOVEP: = 203
MP: = 205

 PLA ;GET UNUSED BYTE
 PLA ;GET MSB OF PLAYER ADDRESS
 STA PLAYER+1
 PLA ;GET LSB OF PLAYER ADDRESS
 STA PLAYER
 PLA ;GET MSB OF STRING
 STA STRING+1
 PLA ;GET LSB OF STRING
 STA STRING
 PLA ;GET MSB OF CHARACTER SET
 STA CHSET+1
 PLA ;GET LSB OF CHARACTER SET
 STA CHSET
 PLA ;GET UNUSED MSB OF LENGTH
 PLA ;GET LSB OF LENGTH
 STA LENGTH

 ;INIT. VALUES
```

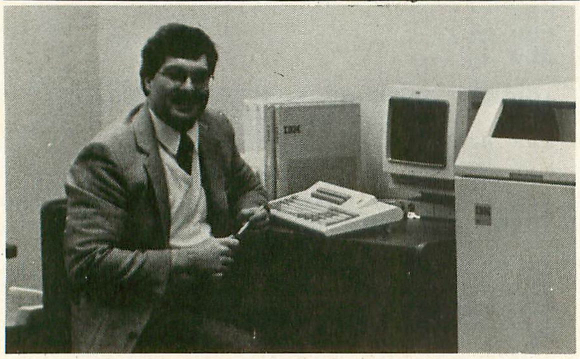
```
  LDA #0
  STA COUNT
  LDA #32
  STA MAX

  ;TEST FOR SINGLE OR DOUBLE LINE
  ;RESOLUTION.

  LDA 559
  CMP #62
  BEQ GETSTR
  LDA #16
  STA MAX
  ;-----------------------------------
  ;GET ATASCII CHARACTER FROM STRING
  ;AND CONVERT IT TO FIND ITS CORRECT
  ;ORDER IN MEMORY.
  ;-----------------------------------
GETSTR: LDY COUNT
  LDA (STRING),Y

  ;IF CHARACTER IS IN INVERSE,
  ;UNSHIFT IT.

  AND #127
C1: CMP #31
  BCS C2

  ;GRAPHICS CHARACTER 0-31
  ;ADD 64 TO ITS VALUE.

  ORA #64
  BNE C3
C2: CMP #95
  BCS C3

  ;UPPERCASE CHARACTER 32-95
  ;SUBTRACT 32 FROM ITS VALUE

  SEC
  SBC #32

  ;LOWERCASE CHARACTER 96-127
  ;DO NOTHING. THEY ARE ALREADY
  ;IN THE CORRECT ORDER.

  ;STORE NEW CHARACTER VALUE IN TEMP.

C3: STA TEMP
  LDA #0
  STA TEMP+1
  ;--------------------------------
  ;MUTIPLI CHARACTER BY 8
  ;--------------------------------
  LDX #3
  CLC
C4: ASL TEMP
  ROL TEMP+1
  DEX
  BNE C4
  ;--------------------------------
  ;ADD CHARACTER SET ADDRESS, AND
  ;PUT THE VALUE IN ZERO PAGE.
  ;--------------------------------
  CLC
  LDA TEMP
  ADC CHSET
  STA CHRGET
  LDA TEMP+1
  ADC CHSET+1
  STA CHRGET+1
  ;-----------------------------------
  ;COPY CHARACTER, FROM ITS ORIGINAL
  ;ADDRESS, TO PLAYER MEMORY.
  ;-----------------------------------
  LDX #0
  LDY #8
CP1: LDA (CHRGET,X)
  STA (PLAYER,X)
  INC CHRGET
  INC PLAYER
  DEY
  BNE CP1
  INC COUNT
  LDA COUNT
  CMP MAX;
  BEQ RETURN
  CMP LENGTH
  BEQ RETURN
  BNE GETSTR;NOT DONE.CONTINUE LOOP.
RETURN: RTS;DONE.GO BACK TO BASIC.
  ;---------------------------------
  ;THIS ROUTINE WILL MOVE THE PLAYER
  ;UP OR DOWN WITH RAP.
  ;---------------------------------
  PLA ;GET UNUSED BYTE.
  PLA ;MSB OF PLAYER.
  STA MOVEP+1
  STA MP+1
  PLA ;LSB OF PLAYER.
  STA MOVEP
  STA MP
  PLA ;UNUSED MSB OF DIRECTION.

  ;SINGLE LINE RESOLUTION SETUP.

  LDX #255
  LDY #255

  ;TEST FOR DOUBLE OR SINGLE LINE
  ;RESOLUTION.

  LDA 559
  CMP #62
  BEQ S1

  ;DOUBLE LINE RESELUTION SETUP.

  LDX #127
  LDY #126

  ;PULL LSB OF DIRECTION OFF STACK,AND
  ;FIND OUT THE DIRECTION TO MOVE.

S1: PLA
  CMP #1
  BEQ MDOWN
  ;---------------------------------
  ;MOVE PLAYER UP ONE BYTE AND RAP.
  ;---------------------------------
  LDY #0
  LDA (MP),Y
  STA TEMP
  INC MOVEP
UP: LDA (MOVEP),Y
  STA (MP),Y
  INY
  DEX
  BNE UP

  ;DONE.GO RAP LAST BYTE AROUND.

  BEQ S3
  ;---------------------------------
  ;MOVE PLAYER DOWN ONE BYTE AND RAP.
  ;---------------------------------
MDOWN: TXA
  TAY
  LDA (MP),Y
  STA TEMP
  DEY
S2: LDA (MOVEP),Y
  STA TEMP+1
  TXA
  TAY
  LDA TEMP+1
  STA (MP),Y
  DEY
  DEY
  DEX
  BNE S2
  ;-----------------------------
  ;RAP LAST BYTE AROUND AND EXIT
  ;ROUTINE.
  ;-----------------------------
  TXA
  TAY
S3: LDA TEMP
  STA (MP),Y
  RTS

  ;BYTE FIELD EQUATES

TEMP: DB 0,0
LENGTH: DB 0
COUNT: DB 0
CHSET: DB 0,0
MAX: DB 0
```

Lyco Computer
Marketing & Consultants

Since 1981

Air orders processed within 24 hours.

Lyco Means Total Service.



Mark "Mac" Bowser, Sales Manager

I would personally like to thank all of our past customers for helping to make Lyco Computer one of the largest mail order companies and a leader in the industry. Also, I would like to extend my personal invitation to all computer enthusiasts who have not experienced the services that we provide. Please call our trained sales staff at our toll-free number to inquire about our diverse product line and weekly specials.

First and foremost our philosophy is to keep abreast of the changing market so that we can provide you with not only factory-fresh merchandise but also the newest models offered by the manufacturers at the absolute best possible prices. We offer the widest selection of computer hardware, software and accessories.

Feel free to call Lyco if you want to know more about a particular item. I can't stress enough that our toll-free number is not just for orders. Many companies have a toll-free number for ordering, but if you just want to ask a question about a product, you have to make a toll call. Not at Lyco. Our trained sales staff is knowledgeable about all the products we stock and is happy to answer any questions you may have. We will do our best to make sure that the product you select will fit your application. We also have Saturday hours — one more reason to call us for all your computer needs.

Once you've placed your order with Lyco, we don't forget about you. Our friendly, professional customer service representatives will find answers to your questions about the status of an order, warranties, product availability, or prices.

Lyco Computer stocks a multimillion dollar inventory of factory-fresh merchandise. Chances are we have exactly what you want right in our warehouse. And that means you'll get it fast. In fact, orders are normally shipped within 24 hours. Free shipping on prepaid orders over $50, and there is no deposit required on C.O.D. orders. Air freight or UPS Blue/Red Label shipping is available, too. And all products carry the full manufacturers' warranties.

I can't see why anyone would shop anywhere else. Selection from our huge in-stock inventory, best price, service that can't be beat—we've got it all here at Lyco Computer.

TO ORDER, CALL TOLL-FREE: 1-800-233-8760
New PA Wats: 1-800-233-8760
Outside Continental US Call: **1-717-494-1030**

Hours: 9AM to 8PM, Mon. - Thurs.
9AM to 6PM, Friday — 10AM to 6PM, Saturday

For Customer Service, call 1-717-494-1670,
9AM to 5PM, Mon. - Fri.
Or write: Lyco Computer, Inc.
P.O. Box 5088, Jersey Shore, PA 17740

VISA® MasterCard

C.O.D. Risk-Free Policy: • full manufacturers' warranties • no sales tax outside PA • prices show 4% cash discount; add 4% for credit cards • APO, FPO, international: add $5 plus 3% for priority • 4-week clearance on personal checks • we check for credit card theft • sorry, compatibility not guaranteed • return authorization required • due to new product guarantee, return restrictions apply • price/availability subject to change • prepaid orders under $50 in Continental US, add $3.00

ATARI®
520 ST-FM Computer

- Built-in Drive

ONLY **$459.95**

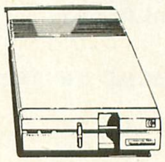
ATARI®
520 ST-FM Color System

Internal drive included

$749.95

INDUS
GTS-100

- Atari ST Drive
- 3.5" DSDD



$195.95

ATARI® 130 XE System

System Includes:

- 130 XE Computer
- 551 Drive

$299.95

HEADSTART
COLOR SYSTEM

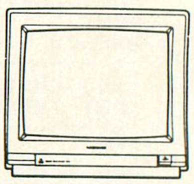
- plug in and use immediately
- IBM-XT compatible
- 2-360K Drives
- Free 1-year limited warranty

$989.95

• Hi Res color monitor included!

MAGNAVOX
CM-8502

- Composite Color
- Green Text Switch
- Speaker
- Suggested Use 130 XE



$179.95

ATARI® HARDWARE

520 RGB	$749.95
520 Mono	$599.95
520 Keyboard	$459.95
130 XE	$135.95
GTS 100 Drive	$195.95

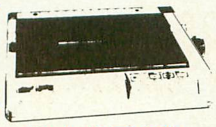
JOYSTICKS

Tac 3	$9.95
Tac 2	$10.95
Tac 5	$12.95
Boss	$11.99
3 Way	$19.99
Winner 909	$24.95
Wic IBM/AP	$29.95
I Controller	$13.95
Epyx 500XJ	$13.95
Kraft KCIII AP/PC	$16.95

1-800-233-8760

CIRCLE #103 ON READER SERVICE CARD.

star micronics



NX-1000
- 144 cps Draft
- 36 cps NLQ
- EZ Font Panel Control

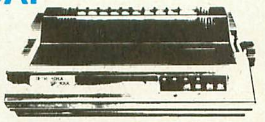
$165.95*

*w/cable purchase

NX-1000 Rainbow Color Printer $225.95

SEIKOSHA

SP-180AI



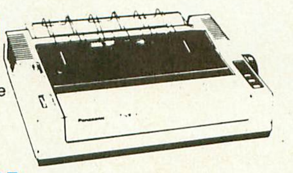
- 100 cps Draft
- 20 cps NLQ
- Std, Par, and IBM Graphics Compatible

Quantities Limited

$125.95

Panasonic Office Automation

1080i Model II



- 150 cps Draft Mode
- NLQ Mode
- Friction & Tractor Feed

$159.95

PRINTERS

star micronics-inc

NX-1000	$165.95*
NX-1000 Color	$225.95
NX-15	$289.95
NR-10	$319.95
NR-15	$419.95
NB-15 24 Pin	$669.95
NX-2400	$309.95
NB24-10 24 Pin	$399.95
NB24-15 24 Pin	$545.95
Laser 8	$1759.95
ND-15	$349.95
NL-10	$149.95

*w/cable purchase

Toshiba

321SL	$489.95
341 SL	$659.95
P351 Model II	$899.95
351 SX 400 cps	$979.95

SEIKOSHA

SP 180Ai	$125.95*
SP 1200Ai	$159.95
SP 1200AS RS232	$159.95
SL 80Ai	$289.95
MP5420FA	$999.95
SP Series Ribbon	$7.95
SK3000 Ai	$339.95
SK3005 Ai	$419.95
SPB 10	$CALL
SL 130Ai	$599.95

* Quantities Limited

BROTHER

M1109	$159.95
M1509	$335.95
M1709	$459.95
Twinwriter 6 Dot & Daisy	$899.95
M1724L	$619.95
HR20	$345.95
HR40	$559.95
HR60	$649.95

EPSON

LX800	$184.95
FX86E	$329.95
FX286E	$424.95
EX800	$399.95
LQ500	$339.95
LQ2500	$789.95
GQ3500	$LOW
LQ850	$525.95
LQ1050	$699.95

Attention Educational Institutions:

If you are not currently using our educational service program, please call our representatives for details.

Panasonic Office Automation

1080i Model II	$159.95
1091i Model II	$189.95
1092i	$299.95
1592	$375.95
1595	$419.95
3131	$289.95
3151	$459.95
KXP 4450 Laser	$1649.95
1524 24 Pin	$529.95
Fax Partner	$579.95
Optical Scanner	$859.95

CITIZEN

120 D	$149.95
180 D	$169.95
MSP-10	$259.95
MSP-40	$289.95
MSP-15E	$335.95
MSP-50	$399.95

OKIDATA

Okimate 20	$129.95
Okimate 20 w/cart	$189.95
180	$219.95
182	$209.95
182+	$225.95
183	$239.95
292 w/interface	$449.95
293 w/interface	$585.95
294 w/interface	$819.95
393	$955.95
Lazer 6	$CALL
390	$479.95
391	$649.95
320	$345.95
321	$445.95

Interfacing available for IBM C 64, Apple and Atari

ATARI

Access:	
Triple Pack	$11.95
Leader Board Pack	$9.99
Activision:	
Music Studio	$19.95
Solid Gold Vol.#1	$10.95
Batteries Included:	
Paperclip 80 Col	$31.95
Broderbund:	
Print Shop	$25.95
Print Shop Compan.	$22.95
Graphic Lib. I, II, III	$13.95
Bank St. Writer	$27.95
Electronic Arts:	
Pinball Con Set	$8.95
Lords of Conquest	$8.95
Starfleet I	$32.95
Chess Master 2000	$25.95
Music Con Set	$8.95
Super Boulderdash	$8.95
One on One	$8.95
Microleague:	
Microleag. Baseball	$22.95
General Manager	$16.95
Stat Disk	$13.95
'87 Team Disk	$13.95
Microprose:	
Conflict in Vietnam	$22.95
F-15 Strike Eagle	$19.95
Kennedy Approach	$13.95

ATARI ST

Access:	
Leader Board	$22.95
Tournament #1	$11.95
10th Frame	$22.95
Activision:	
Champion. Baseball	$22.95
GFL Football	$22.95
Music Studio	$27.95
GBA Basketball	$22.95
Beyond Zork	$28.95
Zork Trilogy	$27.95
Broderbund:	
Superbike Challenge	$11.95
Electronic Arts:	
Arctic Fox	$25.95
Starfleet I	$32.95
Chess Master 2000	$25.95
Gridiron	$32.95
Marble Madness	$23.95
Epyx:	
Sub Battle Simulator	$22.95
World Games	$22.95
Wrestling	$11.95
Winter Games	$11.95
Firebird:	
Pawn	$13.95
Starglider	$25.95
Golden Path	$11.95
Guild of Thieves	$25.95
Tracker	$11.95

ATARI ST

Microleague:	
Microleague Baseball	$33.95
General Manager	$16.95
Wrestling	$25.95
Microprose:	
Silent Service	$22.95
F-15 Strike Eagle	$24.95
Gunship	$28.95
Strategic Simulations:	
Phantasie	$22.95
Phantasie II	$22.95
Wargame Con. Set	$19.95
Phantasie III	$22.95
Sublogic:	
Flight Simulator II	$30.95
Scenery Disk	$14.95
Timeworks:	
Wordwriter ST	$44.95
Partner ST	$27.95
Data Manager ST	$44.95
Unison World:	
Art Gallery 1 or 2	$14.95
Print Master	$19.95
Fonts & Borders	$17.95
Art Gallery Fantasy	$13.95

We stock over 3,000 software titles!

Monitors

Thomson:	
230 Amber TTL/12"	$69.95*
4120 CGA	$199.95
4160 CGA	$199.95
Blue Chip:	
BCM 12" Green TTL	$64.95
BCM 12" Amber TTL	$69.95
NEC	
Multisync II	$589.95
Magnavox:	
BM7652	$79.95
BM7622	$79.95
7BM-613	$79.95
7BM-623	$79.95
CM8502	$179.95
CM8505	$199.95
9CM-053	$CALL
CM8762	$245.95
8CM-515	$259.95
CM9043	$CALL
8CM-873	$499.95

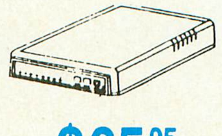
Modems

Avatex:	
1200e	$65.95
1200i PC Card	$65.95
1200p	$89.95
1200hc Modem	$79.95*
2400	$149.95
2400i PC Card	$139.95

*w/cable purchase

Hayes:	
Smartmodem 300	$139.95
Smartmodem 1200	$279.95
Smartmodem 2400	$419.95

Avatex™ 1200E



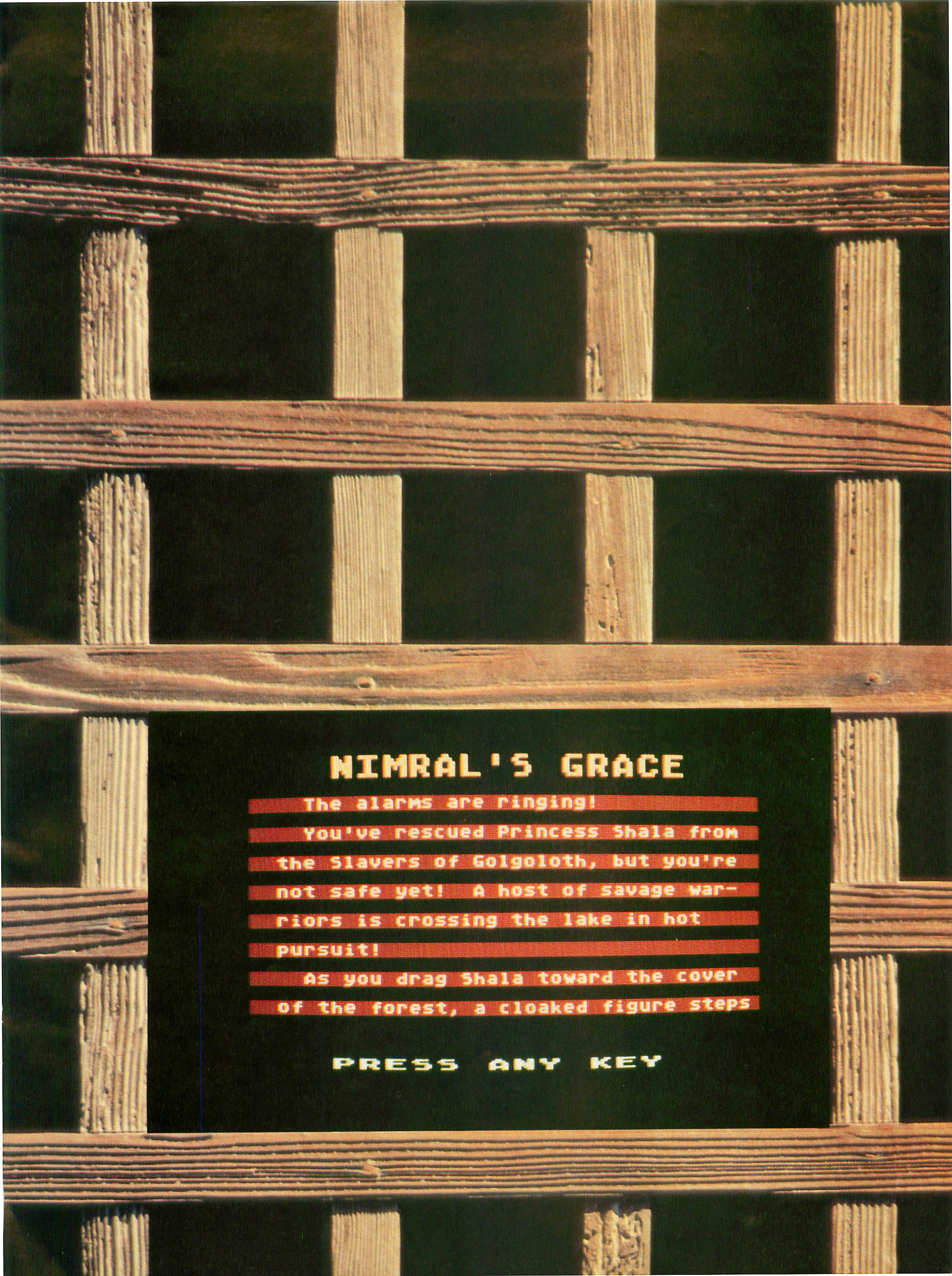
$65.95

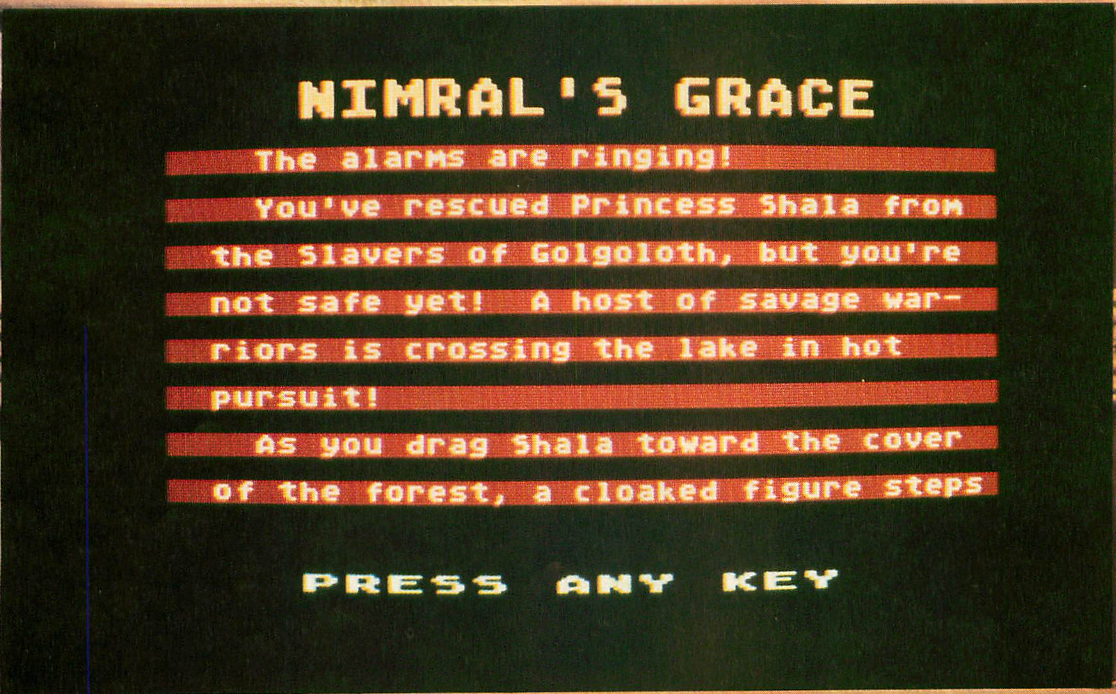
Join the thousands who shop Lyco and Save

SLAVE II
Nimral's Grace

by Clayton Walnum

As promised last month, here's the sequel to *The Slave Cellars of Golgoloth,* and if you thought just *rescuing* the princess was tough, wait until you try to find out who's trying to put her six feet under. Yep, *Slave II: Nimral's Grace* is a mystery, and in order to be successful in your attempt to bring the would-be assassin to justice, you're going to have to search high and low for ten vital clues. Only when you've gathered all the evidence, will you be able to put the culprit behind bars.





NIMRAL'S GRACE

The alarms are ringing!

You've rescued Princess Shala from the Slavers of Golgoloth, but you're not safe yet! A host of savage warriors is crossing the lake in hot pursuit!

As you drag Shala toward the cover of the forest, a cloaked figure steps

PRESS ANY KEY

If you thought rescuing the princess was tough, wait until you try to find out who's trying to put her six feet under.

Typing it in

To make your copy of Slave II, follow exactly the instructions shown below:

1) Type in Listing 1, using *BASIC Editor II* to verify your work. Once you have the listing accurately typed, save it to disk.

2) With a disk in Drive 1, run the program you typed from Listing 1. A file named LINES.LST will be written to your disk.

3) Type in Listing 2, using *BASIC Editor II* to verify your work. Once you have the listing accurately typed, save it to disk.

4) With the program you typed from Listing 2 still in memory, type ENTER "D:LINES.LST" and press Return. The file created in Step 2 will be merged with the program lines already in memory.

5) Type SAVE "D:SLAVE2.BAS" and press Return to save the complete program to disk. Slave II is now ready to run.

Playing a text adventure game

Like most simple text adventure games, you communicate with Slave II by typing two-word commands in a verb/noun format (for example, OPEN DOOR). There are a few exceptions to this format, however. All directional commands should be abbreviated to one letter (N, S, E, W, U or D). To move north, for instance, you would simply type *N* and press Return. There are also times when Slave II will accept one-word responses. You'll discover these as you play.

One command that you'll want to use before trying anything "dangerous" is SAVE GAME. Saving your position in the game allows you to continue where you left off should your character be killed. You should also save your game when you're finished with a session, so you can pick up where you left off when you're ready to play again. To resume a saved game from within Slave II, simply type the command LOAD GAME.

Sometimes, a response to a command will contain more text than will fit in the text window. When this happens you will see the first few lines and the computer will beep. When you've finished reading the text on the screen, press Return, and the next few lines will be shown.

Hint department II

The following adventure hints have been encoded by a forward cycling of the alphabet. That is, the letter *A* has been changed to *B*, the letter *B* has been changed to *C* and so on. To use the hints, find the question that applies to your problem, and decode the first hint in the list following the question. If, after decoding the first hint you're still stuck, decode the next. The last hint in the list is the solution.

How do I get past the fire?
1) Uifsf't b mblf ofbscz.
2) Xbufs lffqt zpv dppm.
3) Kvnq jo uif mblf boe hfu xfu.

How can I get to Nimral's Grace?
1) Ju't upp gbs up xbml.
2) Zpv dpvme vtf b ipstf.
3) Qsbz bu uif tubuvf.

How do I get past the guard?
1) Zpv dbo'u ep ju bmpof.
2) Zpv offe tpnfpof gspn uif djuz xjui zpv.
3) Tibmb nvtu bddpnqboz zpv.

How do I read books?
1) Gjstu zpv nvtu pqfo uifn.
2) Tpnfujnft zpv dbo uvso qbhft.

How can I get from night to day or day to night quickly?
1) Tmffqjoh jt b hppe xbz up qbtt uif ujnf.
2) Zpv nvtu cf jo zpvs sppn up tmffq.
3) Kvtu uzqf HP CFE.

How can I get out of jail?
1) Zpv dbo'u.

How can I get the priest to talk?
1) Uzqf UBML QSJFTU.
2) Zpv ibwf up cf b nfncfs.
3) Sfbe uif cppl jo uif mjcsbsz.
4) Zpv'mm offe tpnf cmbdl qbjou.
5) Qbjou b cmbdl epu po zpvs gpsfifbe.

Where can I get the password?
1) Ibwf zpv gpvoe b xbsespcf zfu?
2) Uifsf't tpnfuijoh cfijoe uif xbsespcf.
3) Npwf uif xbsespcf.
4) Mppl bu uif bmubs.

How can I get into the king's quarters without getting arrested?
1) Zpv dbo'u mfu ijn tff zpv.
2) Xbju voujm if't tmffqjoh.
3) Hp bu ojhiu.

What's with the throne?
1) Uifsf't tpnfuijoh cfijoe ju.
2) Npwf uif uispof.

How do I get past the steel door?
1) Zpv offe b lfz.
2) Hfu uif lfz gspn tpnfpof.
3) Uif qsjftu ibt uif lfz.

What about registering animals?
1) Zpv ibwf up sfhjtufs zpvs ipstf.
2) Hp up uif tnbmm pggjdf.
3) Ufmm uif nbo up SFHJTUFS XIJUF.

How can I survive being stabbed?
1) Xfbs tpnfuijoh qspufdujwf.
2) Ibwf zpv gpvoe uif dibjo nbjm?
3) Zpv nvtu ibwf uif dibjo nbjm.

How do I turn in the culprit?
1) Gjstu zpv nvtu ibwf fopvhi fwjefodf.
2) Hp up uif qfbdflffqfs.
3) Uzqf BDDVTF gpmmpxfe cz uif qfstpo't obnf.

What are the ten pieces of evidence?
1) Cfgpsf zpv sjef up Ojnsbm't Hsbdf, gjoe uif ejsu boe mppl bu ju. Uifo mppl bu uif gppuqsjout.
2) Gjoe uif bnvmfu boe mppl bu ju.
3) Hp joup uif tipfnblfs't tipq boe sfbe ijt cppl. Epo'u gpshfu up uvso uif qbhf.
4) Sfnfncfs uif cspxo ipstf zpv tbx sjejoh bxbz? Hp joup uif tubcmft boe mppl bu uif cspxo ipstf. Uifo mppl bu uif tbeemf.
5) Gjoe tpnf bmf boe hjwf ju up uif nbo jo uif tnbmm pggjdf. Xifo if tfut epxo uif tjmwfs cppl, qjdl ju vq boe sfbe ju.
6) Gjoe uif kfxfmsz cpy boe mppl bu ju.
7) Qpvs uif qpjtpofe xjof gspn uif ubolbse. Gjoe uif gjohfsqsjou lju boe gjohfsqsjou uif fnquz ubolbse.
8) Npwf uif xbsespcf boe mppl bu uif bmubs.
9) Gjoe uif opufcppl boe mppl bu uif dpwfs.
10) Pqfo uif opufcppl boe sfbe xibu't jotjef.

SLAVE II

Listing 1: BASIC

```
TH 10 DIM L$(120)
ES 20 OPEN #1,8,0,"D:LINES.LST"
CR 30 L$="5820 CC$=":L$(10)=CHR$(34)
PZ 40 FOR X=1 TO 77:READ A:L$(10+X)=CHR$(
   A):NEXT X
KM 50 L$(10+X)=CHR$(34):? L$:? #1;L$
FF 60 L$="5840 D$=":L$(9)=CHR$(34)
DH 70 FOR X=1 TO 37:READ A:L$(9+X)=CHR$(A
   ):NEXT X
YE 80 L$(9+X)=CHR$(34):? L$:? #1;L$
GP 90 L$="5860 E$=":L$(9)=CHR$(34)
KF 100 FOR X=1 TO 78:READ A:L$(9+X)=CHR$(
   A):NEXT X
MP 110 L$(9+X)=CHR$(34):? L$:? #1;L$
VZ 120 L$="5880 L$=":L$(9)=CHR$(34)
GA 130 FOR X=1 TO 35:READ A:L$(9+X)=CHR$(
   A):NEXT X
MV 140 L$(9+X)=CHR$(34):? L$:? #1;L$
IH 150 CLOSE #1:END
QO 999 REM ********** CC$ DATA **********
SS 1000 DATA 104,104,133,204,104,133,203,
   104,133,206,104,133,205,104,104,133,20
   7,169,0,141,255,6,170,133,213
ZT 1010 DATA 232,160,0,177,203,209,205,20
   8,8,200,192,4,208,245,134,212,96,173,2
   55,6,24,105,4,197,207
BQ 1020 DATA 240,20,141,255,6,165,205,24,
   105,4,133,205,165,206,105,0,133,206,24
   0,211,208,209,169,0,133,212,96
DD 1099 REM ********** D$ DATA **********
PG 1100 DATA 216,104,104,133,204,104,133,
   203,104,104,133,205,160,0,177,203,201,
   61,240,11,56,233,1,145,203
KU 1110 DATA 200,196,205,208,240,96,169,3
   3,240,244,208,242
DZ 1199 REM ********** E$ DATA **********
HZ 1200 DATA 104,104,104,141,254,6,104,10
   4,141,255,6,165,88,133,203,165,89,133,
   204,162,0,236,255,6,240
NC 1210 DATA 18,165,203,24,105,40,133,203
   ,165,204,105,0,133,204,232,240,235,208
   ,233,169,0,170,160,39,145
HW 1220 DATA 203,136,16,251,232,236,254,6
   ,240,17,165,203,24,105,40,133,203,165,
   204,105,0,133,204,169,0,240,226,96
JL 1299 REM ********** L$ DATA **********
YL 1300 DATA 104,104,104,141,255,6,104,13
   3,204,104,133,203,160,0,177,203,201,32
   ,240,8,200,204,255,6,208
QM 1310 DATA 244,160,0,132,212,169,0,133,
   213,96
```

SLAVE II

Listing 2: BASIC

```
JI 0 REM * BY CLAYTON WALNUM *
        * REVISED 7/22/88    *
OZ 1 N11=11:N12=12:N13=13:N14=14:N15=15:N
   16=16:N17=17:N18=18:N19=19:N20=20:N100
   0=1000:N6760=6760
WY 2 N1=1:N2=2:N3=3:N4=4:N5=5:N6=6:N7=7:N
   8=8:N9=9:N10=10:NV=26:NN=59:SZ=19:GOTO
    5200
SQ 3 A=USR(ADR(D$),ADR(A$),LEN(A$)):RETUR
   N
JV 4 GOSUB N3:? A$:GOSUB N11:A$="":RETURN
JY 5 CL=N0:FOR X=N1 TO N10:CL=CL+CL(X):NE
   XT X:RETURN
HN 8 FOR X=255 TO N0 STEP -0.5:SOUND N0,X
   ,N10,N8:NEXT X:FOR X=N16 TO N0 STEP -0
   .1:SOUND N0,100,N8,X:NEXT X:RETURN
QY 9 FOR X=N1 TO 1500:NEXT X:RETURN
JI 10 A=USR(ADR(E$),N5,C):POSITION N2,C:R
   ETURN
ZD 11 CLOSE #1:OPEN #N1,N4,N0,"K:":GET #N
   1,A:CLOSE #N1:RETURN
EY 12 ? "I don't understand.":GOTO 1000
BY 40 A=USR(ADR(E$),N1,N1):POSITION N2,N1
OX 45 N=N0:S=N0:E=N0:W=N0:U=N0:D=N0:GOSUB
    R*N10:POSITION N2,N1:GOSUB N3:? #6;A$
   :A$="":RETURN
VO 50 A$="Po!b!cfbdi":I(41)=-N5:RETURN
EU 51 A$="Uif!tmbwfst!bsf!bqqspbdijoh!gbt
   u!!!!!gspn!uif!xftu=↓":E=N7:RETURN
ZD 70 A$="Jo!uif!gpsftu":E=N8:N=N12:I(41)
   =-R:I(24)=-R:RETURN
BN 80 A$="Cfhjoojoh!pg!b!spbe":N=N9:E=N13
   :W=N7:RETURN
DE 90 A$="Jo!uif!gpsftu":N=80:S=N8:W=N12:
   I(24)=-R:RETURN
NZ 120 A$="Jo!b!dmfbsjoh":E=N9:S=N7:N=79:
   RETURN
MR 130 A$="Po!b!mpoh!spbe":W=N8:E=N15:RET
   URN
EC 131 A$="Jo!uif!ejtubodf!zpv!tff!tpnfpo
   f!hbm.!!mpqjoh!bxbz!po!b!cspxo!ipstf/▌
   ":GOSUB N4:RETURN
ZU 150 A$="Ojnsbm(t!Hsbdft(!gspou!hbuf":W
   =N13:RETURN
IW 160 A$="Jo!b!dpvsuzbse":N=21:E=25:S=N1
   7:CY=N1:RETURN
KD 161 A$="B!nbo!tufqt!gpsxbse/!!Tibmb!tb
   zt-!!!!!(Epmops-!nz!cfuspuife-!xiz!bsf
   !zpv!!!!ifsf!jo!Ojnsbm(t!Hsbdf@(▌"
UC 162 GOSUB N4:A$="Epmops!ublft!Tibmb!jo
   !ijt!bsnt!boe!!!!tbzt-!(Tibmb-!nz!mpwf
   -!uibol!Ojnsbm!!!zpv!bsf!tbgf=(▌"
KO 163 GOSUB N4:A$="(Xifo!J!ifbse!pg!zpvs
   !dbquvsf!J!dbnf!!up!pggfs!nz!tfswjdft!
   up!zpvs!gbuifs-!!uif!ljoh/(▌"
VT 164 GOSUB N4:A$="(Dpnf-!Tibmb-!zpvs!gb
   njmz!bxbjut!zpv/(Epmops!mppl t!bu!zpv/!
   (Xbssjps-!zpv!!!!xjmm!cf!sfxbsefe=(▌"
HJ 165 GOSUB N4:A$="(Hvbset=!!Tipx!pvs!hv
   ftu!up!ijt!!!!!!!!!rvbsufst/(▌"
MR 166 GOSUB N4:A$="":R=39:UL=N1:UI=N1:EN
   T=N1:I(N5)=-43:I(N7)=-76:GOTO N1000
LR 170 A$="Po!b!tusffu":N=N16:RETURN
VR 210 A$="Po!b!tusffu":W=22:N=23:S=N16:R
   ETURN
VL 220 A$="Cftjef!uif!djuz!xbmm":N=24:E=2
   1:I(49)=-R:RETURN
JA 230 A$="Po!b!tusffu":W=24:S=21:RETURN
SK 240 A$="Cftjef!uif!djuz!xbmm":E=23:S=2
   2:CY=N1:I(49)=-R:RETURN
RC 250 A$="Bu!uif!qbmbdf!hbuf":W=N16:E=27
   :RETURN
KY 270 A$="Jo!b!ibmm":W=25:U=38:N=28:D=49
   :E=30:I(25)=-R:RETURN
BX 280 A$="Jo!uif!Qfbdflffqfst!pggjdf":S=
   27:RETURN
BM 290 A$="Jo!b!kbjm!dfmm":RETURN
JV 300 A$="Jo!b!ibmm":W=27:N=31:E=33:S=32
   :RETURN
OL 310 A$="Jo!uif!hvbse!sppn":S=30:RETURN
BB 320 A$="Jo!uif!spzbm!ejojoh!sppn":N=30
   :RETURN
NN 330 A$="Jo!b!ibmm":W=30:N=34:S=35:U=44
   :I(25)=-R:RETURN
NQ 340 A$="Jo!uif!uispof!sppn":S=33:RETUR
   N
DZ 350 A$="Jo!uif!tvqqmz!sppn":N=33:RETUR
   N
AO 380 A$="Jo!b!ibmm":N=39:S=40:D=27:E=41
   :I(25)=-R:RETURN
JU 390 A$="Zpvs!rvbsufst":S=38:RETURN
JI 391 A$="B!tfswbou!foufst!uif!sppn!xjui
   !b!!!!!!ubolbse!pg!xjof/!!(Dpvsuftz!pg
   !uif!!!!ljoh-!nz!Mpse/(▌":GOSUB N4
PX 392 RM=N1:RETURN
NB 400 A$="Jo!Ojloj l!uif!Kftups(t!rvbsufs
   t":N=38:RETURN
TE 410 A$="Jo!b!ibmm":W=38:N=42:E=44:S=43
   :RETURN
CQ 420 A$="Jo!Epmops(t!rvbsufst":S=41:RET
   URN
ZW 430 A$="Jo!Tibmb(t!rvbsufst":N=41:SH=N
   1:RETURN
XW 440 A$="Jo!b!ibmm":W=41:N=45:D=33:S=46
   :I(25)=-R:RETURN
AQ 450 A$="Jo!Epnojt!uif!Bewjtps(t!rvbsuf
   st":S=44:RETURN
FV 460 A$="Jo!uif!ljoh(t!rvbsufst":N=44:R
   ETURN
PG 490 A$="Jo!b!ibmm":N=52:E=50:U=27:RETU
   RN
UE 500 A$="Jo!b!ibmm":W=49:N=53:E=54:RETU
   RN
IL 520 A$="Jo!uif!difnjtu(t!mbc":S=49:RET
   URN
QJ 530 A$="Jo!uif!rvffo(t!rvbsufst":S=50:
   RETURN
PM 540 A$="Jo!uif!mjcsbsz":W=50:RETURN
WR 650 A$="Jo!uif!dbubdpnct":W=34:N=65:E=
   66:S=65:RETURN
TS 660 A$="Jo!uif!dbubdpnct":W=66:S=65:E=
   65:N=71:RETURN
GP 710 A$="Jo!b!ebsl!ufnqmf":N=72:S=66:RE
   TURN
BK 720 A$="Jo!b!tupsbhf!sppn":S=71:RETURN
HG 730 A$="Jo!uif!pggjdf!pg!bojnbm!sfhjtu
   sz":E=N17:RETURN
UC 740 A$="Bu!uif!tipfnblfs(t":W=23:RETUR
   N
VG 750 A$="Bu!uif!qbxo!tipq":W=21:RETURN
MI 760 A$="Bu!uif!tubcmft":W=N17:RETURN
WM 770 A$="Jo!b!tnbmm!dibqfm":S=23:RETURN
UR 780 A$="Jo!b!ebsl!uvoofm":E=24:CY=N0:R
   ETURN
IF 790 A$="Jo!b!nfbepx":S=N12:E=80:RETURN
QY 800 A$="Jo!b!nfbepx":W=79:S=N9:RETURN
IA 810 A$="Jo!b!ufou":W=N8:RETURN
KD 900 RESTORE 6120:FOR X=N1 TO NN:READ A
   $,A:Q=SZ-LEN(A$):I$(X*SZ-(SZ-N1),X*SZ-
   Q)=A$:I(X)=A:NEXT X
YN 905 RESTORE 6100:FOR X=N1 TO NV:READ A
   :V(X)=A:NEXT X
QT 925 GOTO 6280
TX 950 C=N5:GOSUB N10
SB 951 FOR X=N1 TO NN:IF ABS(I(X))=R THEN
    ? #N6;I$(X*SZ-SZ+N1,X*SZ):IT=N1
NQ 953 NEXT X:IF IT=N0 THEN ? #N6;"Nothin
   g"
AB 954 RETURN
HI 960 A=USR(ADR(E$),N1,N11):POSITION N2,
   N11:IF N+S+E+W+D+U=N0 THEN ? #N6;"None
   ":RETURN
BF 962 IF N>N0 THEN ? #N6;"North  ";
KU 963 IF S>N0 THEN ? #N6;"South  ";
LA 964 IF E>N0 THEN ? #N6;"East  ";
LZ 965 IF W>N0 THEN ? #N6;"West  ";
YD 966 IF U>N0 THEN ? #N6;"Up  ";
DY 967 IF D>N0 THEN ? #N6;"Down"
ZT 970 RETURN
JW 980 C=N14:GOSUB N10
UL 981 FOR X=N1 TO NN:IF I(X)=-N1 THEN ?
   #N6;I$(X*SZ-SZ+N1,X*SZ):INV=N1:NEXT X:
   RETURN
AN 982 NEXT X:IF INV=N0 THEN ? #N6;"Nothi
   ng"
AE 983 RETURN
MA 1000 IF LEN(A$)>N0 THEN A=USR(ADR(D$),
   ADR(A$),LEN(A$)):? A$:A$=""
KV 1020 IF UL THEN GOSUB 40:GOSUB 950:GOS
   UB 960
VF 1040 IF US THEN GOSUB 950
QP 1060 IF UD THEN GOSUB 960
UP 1080 IF UI THEN GOSUB 980
FI 1100 IT=N0:INV=N0:UL=N0:US=N0:UD=N0:UI
   =N0
QB 1120 IF R=N16 AND  NOT ENT THEN GOSUB
   N11:GOTO 161:A$=""
ET 1130 IF R=46 AND DY>N0 THEN A$="Hvbset
   =!Bssftu!uijt!nbo=▌":GOTO 6600
PY 1140 IF DD THEN GOSUB N9:GOTO 7240
JU 1160 IF R=39 AND  NOT RM THEN GOSUB 39
   1:A$=""
BC 1165 IF R=N13 AND  NOT SE THEN GOSUB 1
   31:SE=N1
AE 1166 IF R=42 AND DY>N0 AND I(46)=-R TH
   EN GOTO 4606
ZD 1167 IF DY>N0 AND I(46)=-R THEN I(46)=
   N0
BN 1180 IF TURN/100=INT(TURN/100) THEN GO
   SUB 6500
EW 1200 IF R=43 AND DY>N0 THEN GOTO 6560
TK 1210 IF R=43 AND I(N18)=-40 THEN I(N18
   )=-41
OU 1211 IF R<>41 OR I(55)=-N1 OR I(N18)<>
   -R OR NK<>N0 THEN 1213
OW 1212 A$="Ojloj l!qmvohft!b!lojgf!jo!zpv
   s!diftu=▌":GOSUB N4:GOTO 7240
SM 1213 IF R=41 AND I(N18)=-R AND NK=N0 T
   HEN A$="Ojloj l!tubct!zpv-!cvu!uif!dibj
   o!nbjm!!tbwft!zpv=":NK=N1:GOTO N1000
PT 1214 IF SH AND I(36)<>-N2 AND DY>N0 TH
   EN X=RND(N0):IF X>0.9 THEN 6741
JE 1215 IF R<>41 AND NK THEN I(N18)=-40
ZI 1220 IF R=29 THEN CNT=CNT+N1:IF CNT=N5
    THEN 6640
HH 1230 IF R=N5 AND TURN=N8 THEN 6744
GM 1240 TRAP 1240:POSITION N1,N15:? :? "C
   ommand";:POKE 752,N0
JU 1260 SOUND N0,N20,N10,N8:FOR X=N1 TO N
   10:NEXT X:SOUND N0,N0,N0,N0:INPUT IN$:
   POKE 752,N1
HO 1280 TURN=TURN+N1:IF TURN>200 AND CY A
   ND  NOT RG THEN RG=N1:GOTO 6720
IJ 1300 IF R=N8 AND IN$="E" AND I(N7)<>-N
   1 THEN A$="Ju(t!upp!gbs!up!xbml/":GOTO
    N1000
IP 1340 IF R=N12 AND IN$="PRAY" THEN 5060
SW 1360 IF I(N10)<>-N1 OR IN$<>"W" OR R<>
   74 THEN 1400
UR 1380 A$="Uif!tipfnblfs!tdsfbnt-!(Tupq-
   !uifjg=!!Uibu(t!nz!cppl=(▌":UI=N1:I(N1
   0)=N0:GOTO 6600
OJ 1400 IF I(33)<>-N1 OR IN$<>"E" THEN 14
   60
VI 1420 A$="(Tupq-!uijfg=!!Uibu!cppl!jt!!
   !!!!!!!!!hpwfsonfou!qspqfsuz=(▌":I(33)
   =N0:UI=N1:GOTO 6600
SE 1460 L=LEN(IN$):IF L=N1 THEN V$=IN$:GO
   TO 1820
CV 1480 A=USR(ADR(L$),L,ADR(IN$)):IF A=N0
    THEN GOTO N12
ML 1500 V$=IN$(N1,A):N$=IN$(A+N2,LEN(IN$)
   )
EM 1520 IF V$="SAVE" THEN 7060
YU 1540 IF V$="LOAD" THEN 6820
XB 1560 IF LEN(V$)<N2 OR LEN(N$)<N3 THEN
   GOTO N12
FI 1580 IF LEN(V$)=N2 THEN V$(N3)="  "
PD 1600 IF LEN(V$)=N3 THEN V$(N4)=" "
DO 1610 IF LEN(N$)=N3 THEN N$(N4)=" "
XA 1620 Y=USR(ADR(CC$),ADR(N$),ADR(NN$),L
   EN(NN$))
SS 1640 Z=USR(ADR(CC$),ADR(V$),ADR(VB$),L
   EN(VB$))
NP 1660 IF N$="PAGE" AND Z=25 THEN 1740
LE 1680 IF N$(N1,N4)="HORS" OR N$="BOOK"
   THEN ? "Refer to it by color.":GOTO N1
   000
BS 1685 IF N$="DOOR" THEN ? "Which one?":
   GOTO N1000
DE 1700 IF Z=N15 AND N$(N1,N4)="FORE" THE
   N 1740
HQ 1720 IF Y=N0 OR Z=N0 THEN GOTO N12
ZD 1740 Z=V(Z)
KY 1760 IF Z>N16 THEN Z=Z-N16:GOTO 1800
JM 1780 ON Z GOSUB 2020,2540,2730,2900,29
   61,3020,3080,3205,3245,3300,3380,3580,
   3700,3980,4160,4220
IM 1800 ON Z GOSUB 4601,4681,4740,4780,48
   40,4980
PD 1820 IF V$="N" AND N>N0 THEN R=N:GOTO
   1960
BW 1840 IF V$="S" AND S>N0 THEN R=S:GOTO
   1960
TC 1860 IF V$="E" AND E>N0 THEN R=E:GOTO
   1960
MI 1880 IF V$="W" AND W>N0 THEN R=W:GOTO
   1960
GM 1900 IF V$="U" AND U>N0 THEN R=U:GOTO
   1960
QF 1920 IF V$="D" AND D>N0 THEN R=D:GOTO
   1960
EL 1940 ? "▚NO SUCH DIRECTION!":GOTO N100
   0
NT 1960 UL=N1:GOTO N1000
FU 2020 IF R=N12 AND Y=N4 THEN A$="Uifsf(
   t!b!qmbrvf!po!ju/":I(43)=-R:US=N1:GOTO
    N1000
KK 2040 IF R=N12 AND Y=43 THEN A$="Uifsf(
   t!xsjujoh!po!ju/":GOTO 1000
LA 2060 IF R=N9 AND Y=29 THEN I(N3)=-R:US
   =N1:A$="Gppuqsjout=":GOTO N1000
```

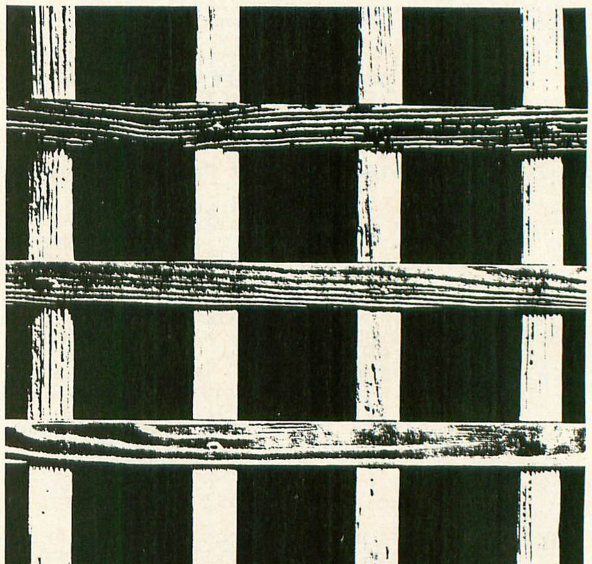
```
WB 2080 IF R=N9 AND Y=3 THEN A$="Uifz(sf!
   bcpvu!tj↑f!21/":CL(N8)=N1:GOTO N1000
BP 2100 IF R=24 AND N$(N1,N3)="GRO" AND I
   (35)=N0 THEN A$="Uif!hspvoe!mpplt!ejtu
   vscfe/":GOTO N1000
XA 2160 IF R=N5 AND Y=42 THEN A$="Uifz!mp
   pl!bxgvmmz!nfbo=":GOTO N1000
XL 2180 IF R=N5 AND Y=41 THEN A$="Zpv!tff
   !uif!gpsftu!uispvhi!uif!gmbnft":GOTO N
   1000
IF 2200 IF R=N5 AND Y=28 THEN A$="Uif!xbu
   fs!mpplt!dpme=":GOTO N1000
CP 2220 IF (I(Y)=R OR I(Y)=-N1) AND Y=N5
   THEN A$="Tif(t!cfbvujgvm=":GOTO N1000
SM 2240 IF R=43 AND Y=N5 THEN A$="Tif!mpp
   lt!ibqqz!up!tff!zpv=":GOTO N1000
YS 2260 IF R=74 AND Y=N10 THEN A$="Uif!dp
   wfs!tbzt;!SPZBM!TJ[FT":GOTO N1000
QN 2280 IF R=76 AND Y=N7 THEN A$="Ju(t!zp
   vst-!evnnz=":GOTO N1000
UW 2300 IF R=76 AND Y=26 THEN A$="Ju(t!xf
   bsjoh!b!tbeemf/":I(37)=-R:US=N1:GOTO N
   1000
JN 2320 IF R=76 AND Y=37 AND I(Y)=-R THEN
    A$="Ju!ibt!uif!obnf!UPQQFS!po!ju/":CL
   (N5)=N1:GOTO N1000
VI 2340 IF (R<>22 AND R<>24) OR Y<>49 THE
   N 2400
PT 2360 A$="Mput!pg!gmpxfst!ifsf/":IF R=2
   4 AND I(35)=N0 THEN A$(LEN(A$)+N1)="!!
   Gvooz-!uif!!!!ejsu!tffnt!ejtuvscfe/"
CE 2380 GOTO N1000
SM 2400 IF R=73 AND Y=38 AND I(33)=N0 THE
   N A$="If(t!ipmejoh!b!tjmwfs!cppl/":GOT
   O N1000
YH 2420 IF I(Y)=-N1 AND Y=33 AND SIL=N0 T
   HEN A$="Uif!dpwfs!tbzt;!Spzbm!Bojnbmt"
   :GOTO N1000
XY 2440 IF R=77 AND Y=44 THEN A$="If!mppl
   t!b!cju!tjojtufs/":GOTO N1000
LN 2460 IF R=42 AND DY>N0 AND (Y=N2 OR Y=
   21) THEN A$="(Uibu(t!qsjwbuf!qspqfsuz=
   !!Hvbset=(▌":GOTO 6600
PW 2480 IF R=42 AND Y=N19 AND DY>N0 THEN
   A$="If(t!xbudijoh!zpv!xbsjmz/":GOTO N1
   000
XU 2481 IF R<>42 OR Y<>46 OR I(Y)<>-R THE
   N 2483
DJ 2482 A$="Uifsf(t!uif!xpse!EBSLOFTT!cfm
   px!uif!!!gjhvsf!pg!Hpmhpmpui/":CL(N1)=
   N1:GOTO N1000
ZQ 2483 IF Y=22 AND I(Y)=-N1 THEN A$="Uif
   !dpwfs!tbzt;!QSJFTUT!PG!HPMHPMPUI":CL(
   N9)=N1:GOTO N1000
GB 2485 IF Y=N2 AND I(N2)=-N1 THEN A$="Ui
   fsf(t!b!tubs!tibqf!qsfttfe!joup!!!!ui
   f!wfmwfu/":CL(N2)=N1:GOTO N1000
NG 2486 IF Y=N1 AND I(Y)=-N1 THEN A$="Ju(
   t!jo!uif!tibqf!pg!b!tubs/":CL(N10)=N1:
   GOTO N1000
KL 2500 ? "You see nothing of interest.":
   GOTO N1000
LM 2540 IF I(Y)=-R THEN ? "YOU CAN'T GET
   THAT!":GOTO N1000
SN 2560 IF I(Y)=-N1 THEN ? "YOU ALREADY H
   AVE IT!":GOTO 1100
VI 2600 IF ABS(I(Y))<>R THEN ? "I DON'T S
   EE A(N) '";N$;"'":GOTO 1100
LC 2610 IF R=42 AND DY>N0 AND (Y=N2 OR Y=
   21) THEN A$="(Uibu(t!qsjwbuf!qspqfsuz=
   !!Hvbset=(▌":GOTO 6600
ZP 2620 LOCATE N2,N18,A:IF A<>32 THEN ? "
   YOU CAN'T CARRY ANYMORE!":GOTO N1000
RI 2660 ? "Okay↑":I(Y)=-N1:US=N1:UI=N1:GO
   TO 1000
LL 2680 GOTO N6760
JP 2730 IF R<>41 OR I(N18)<>-41 OR Y<>36
   OR I(36)<>-N1 THEN 2740
US 2731 A$="If!ublft!uif!hpme-!qspnjtft!o
   pu!up!!!!nfoujpo!zpv!xfsf!jo!Tibmb(t!s
   ppn/":I(36)=-N2:UI=N1:GOTO N1000
OP 2740 LOCATE N2,N8,A:IF A<>32 THEN ? "T
   HERE'S NO MORE ROOM HERE!":? :GOTO 110
   0
SX 2760 IF I(Y)<>-N1 THEN ? "YOU DON'T HA
   VE IT!":? :GOTO 1100
NJ 2780 IF R<>73 OR I(40)<>-N1 THEN 2840
SO 2800 A$="If!ublft!uif!cpuumf-!bctfounj
   oefemz!!!tfuujoh!epxo!uif!cppl/"
MN 2820 I(33)=R:I(40)=N0:US=N1:UI=N1:GOTO
    N1000
IL 2840 ? "Okay↑":I(Y)=R:US=N1:UI=N1:GOTO
    N1000
LJ 2860 GOTO N6760
SQ 2900 IF Y=N10 AND I(Y)=-N1 AND PG=N0 T
```

SLAVE II

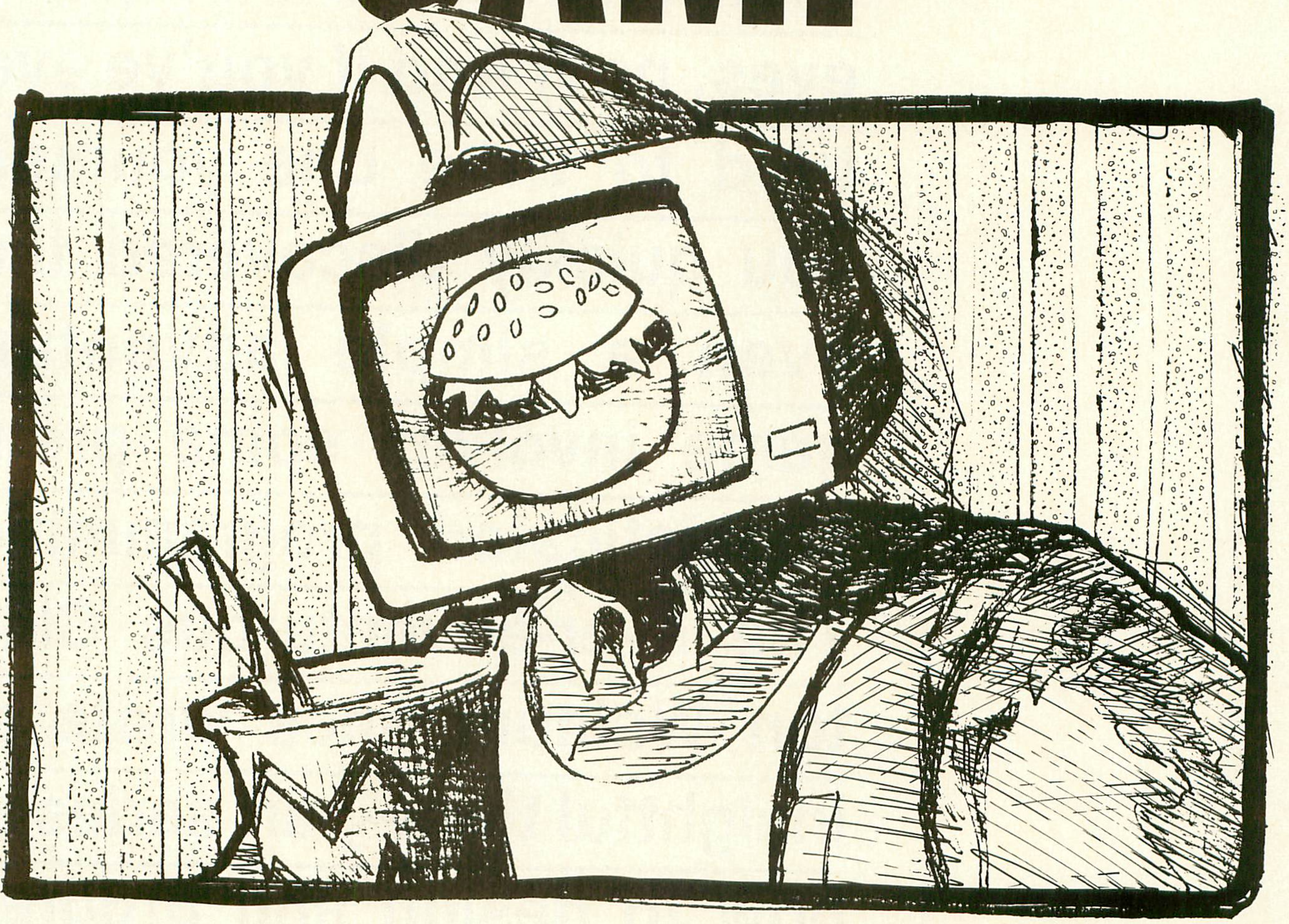
```
   HEN A$=O$:PG=N1:GOTO N1000
EU 2920 IF SIL=N0 AND Y=33 AND I(Y)=-N1 T
   HEN A$=O$:SIL=N1:GOTO N1000
PY 2921 IF Y=22 AND I(Y)=-N1 AND  NOT NB
   THEN A$=O$:NB=N1:GOTO N1000
DV 2924 IF Y=N14 AND I(Y)=-N1 AND  NOT DI
   A THEN A$=O$:DIA=N1:GOTO N1000
ZJ 2926 IF Y=39 AND I(Y)=-N1 AND  NOT GR
   THEN A$=O$:GR=N1:GOTO N1000
LF 2940 GOTO N6760
RQ 2961 IF R=34 AND Y=53 AND I(N12)=-N1 A
   ND I(Y)=-R AND  NOT UNL THEN A$="Uif!l
   fz!vompdlfe!ju/":UNL=N1:GOTO N1000
LR 2980 GOTO N6760
VN 3020 IF Y=45 AND I(Y)=-N1 THEN A$="Tpn
   fuijoh!ubtuft!gvooz!ifsf=":GOTO N1000
KO 3040 GOTO N6760
XU 3080 IF R<>75 OR I(N11)<>-N1 OR Y<>N11
    THEN GOTO N6760
CB 3100 A$="If!ublft!zpvs!txpse!boe!mbzt!
   b!hpme!!!qjfdf!po!uif!dpvoufs/"
CO 3120 I(36)=R:US=N1:UI=N1:I(N11)=N0:GOT
   O N1000
KW 3160 GOTO N6760
YW 3205 IF R<>28 THEN 3220
QI 3210 GOSUB N5:IF CL=N10 AND Y=N19 THEN
    GOTO 6780
SE 3215 A$="Zpv!epo(u!ibwf!fopvhi!fwjefod
   f=▌":GOTO N1000
KM 3220 GOTO N6760
IH 3245 IF Y=45 AND I(45)=-N1 THEN A$="Pv
   u!uif!xjoepx///":I(45)=N0:I(57)=-N1:UI
   =N1:GOTO N1000
KY 3260 GOTO N6760
XZ 3300 IF R=43 AND Y=N5 AND KS THEN A$="
   Tif!sfuvsot!zpvs!ljttft!xjui!hsfbufs!!
   qbttjpo/":GOTO N1000
UU 3320 IF R=43 AND Y=N5 THEN A$="Tif(t!b
   !mjuumf!tvsqsjtfe-!cvu!sfuvsot!zpvs!lj
   tt/":KS=N1:GOTO N1000
KU 3340 GOTO N6760
YY 3380 IF R<>31 OR (Y<>51 AND Y<>N8) THE
   N 3540
SI 3400 IF CT+N20>TURN OR GD=N4 THEN A$="
   Uifz!tbx!zpv!boe!tupqqfe!ubmljoh/":GOT
   O N1000
UZ 3415 GD=GD+N1:IF GD>N3 THEN GD=N1
EK 3420 CT=TURN:IF GD=N1 THEN A$="(Op!pof
   !xjmm!cf!hvbsejoh!uif!qsjodftt(!rvbsuf
   st!upojhiu/("
AB 3440 IF GD=N2 THEN A$="(Uif!sfhjtusbs!
   sfbmmz!mpwft!ijt!bmf/("
XO 3460 IF GD=N3 THEN A$="(J!ifbs!Ojloj!
   sftqpoet!xfmm!up!uif!!!!qspnjtf!pg!hpm
   e/("
BL 3500 GOTO N1000
KY 3540 GOTO N6760
XN 3580 IF I(N13)<>-N1 OR N$(N1,N4)<>"FOR
   E" THEN 3620
UA 3600 A$="Zpv!qvu!b!cmbdl!epu!po!zpvs!g
   psfifbe/":PT=N1:GOTO N1000
KU 3620 GOTO N6760
EE 3700 IF R<>N12 OR Y<>43 THEN 3740
KT 3720 A$="Qsbzfs!boe!hppe!effet!bsf!uif
   !!!!!!!!!tjodfsftu!gpsn!pg!xpstijq/":G
   OTO N1000
DY 3740 IF R=N8 AND Y=N6 THEN A$="OJN5BM(
   T!H5BDF!..!31!MFBHVFT":GOTO N1000
SW 3760 IF I(N10)<>-N1 OR Y<>N10 THEN 382
   0
ZL 3780 IF PG=N1 THEN A$="LJOH!......!tj
   ♠f!23!!!!!!!!!!!!!!!!!!Rvffo!......!tj
   ♠f!9":GOTO N1000
OM 3800 IF PG=N2 THEN A$="Epmops!.....!tj
   ♠f!21!!!!!!!!!!!!!!!!!!Ojloj!.....!tj
   ♠f!21":CL(N3)=N1:GOTO N1000
JE 3820 IF R=N16 AND Y=47 THEN A$="Ju!tbz
   t;!Bmm!bojnbmt!nvtu!cf!!!!!!!!!!sfhjtu
   fsfe!jnnfejbufmz=":GOTO N1000
TC 3840 IF Y=23 AND I(23)=-N1 THEN A$="Xb
   udi!gps!TMBWF!JJJ;!UIF!HPET!!!!!!!!!UI
   FNTFMWFT/":GOTO N1000
KB 3860 IF Y<>33 OR I(Y)<>-N1 OR SIL=N0 T
   HEN 3920
ZH 3880 A$="Cpxxpx/////////////Ljoh(t!eph
   !!!!!!!!!!Upqqfs////////////Epmops(t!i
   pstf!!!!!":CL(N6)=N1
VP 3900 A$(LEN(A$)+N1)="Dbuojq//////////
   //Rvffo(t!qbsblffu▌":GOSUB N3:? A$:A$=
   "":GOSUB N11:GOTO N1000
EC 3920 IF Y=22 AND I(Y)=-N1 AND NB THEN
   A$="Hfgofu!!!!!!Cpccfm!!!!!!Epmops":
   CL(N7)=N1:GOTO N1000
UD 3922 IF Y<>N14 OR I(Y)<>-N1 OR  NOT DI
   A THEN 3928
KR 3924 A$="J!uijol!uif!ljoh!ibt!mfbsofe!
   uibu!!!!!tibmb!jt!opu!usvmz!ijt!ebvhiu
   fs/!J!!!!gfbs!gps!ifs!mjgf-"
HL 3925 A$(LEN(A$)+N1)="!bt!xfmm!bt!nz!px
   o/↓":GOTO N1000
BJ 3928 IF Y<>39 OR I(Y)<>-N1 OR  NOT GR
   THEN 3931
UW 3929 A$="B!cmbdl!epu!po!uif!gpsfifbe!j
   t!gsf.!!!rvfoumz!vtfe!cz!uif!gpmmpxfst
   !pg!!!!!!Hpmhpmpui!up"
IS 3930 A$(LEN(A$)+N1)="!jefoujgz!fbdi!pu
   ifs/↓":GOTO N1000
AM 3931 IF Y=N15 AND I(Y)=-N1 THEN A$="Ui
   f!rvffo!ibt!cfusbzfe!nf/!!J!xjmm!!!!ib
   wf!nz!sfwfohf=":GOTO N1000
LG 3940 GOTO N6760
IV 3980 IF R<>N5 OR Y<>28 THEN 4020
UJ 4000 A$="Uif!xbufs(t!tp!dpme!zpv!kvnq!
   sjhiu!!!!pvu=!!Zpv!boe!Tibmb!bsf!tpblf
   e=":WT=N1:GOTO N1000
AW 4020 IF R<>N5 OR Y<>41 OR  NOT WT THEN
    4060
AX 4040 A$="Uif!xbufs!ifmqfe!zpv!hfu!uisp
   vhi=!)B!!mjuumf!tjohfe-!uipvhi/*":R=N7
   :UL=N1:GOTO 1000
GT 4060 IF (R=N7 AND Y=41) OR (R=N5 AND Y
   =41 AND  NOT WT) THEN A$="Zpv(wf!cffo!
   cbscfdvfe=▌":GOSUB N4:GOTO 7240
RU 4100 IF R=24 AND Y=35 AND I(Y)=-R THEN
    A$="Plbz=":R=78:UL=N1:GOTO N1000
KH 4102 IF R=34 AND Y=53 AND  NOT UNL THE
   N A$="Ju(t!mpdlfe/":GOTO N1000
VN 4104 IF R=34 AND Y=53 AND UNL THEN R=6
   5:UL=N1:GOTO N1000
PN 4105 IF (R=23 OR R=21) AND (Y=50 OR Y=
   N9 OR Y=30 OR Y=31) AND DY<N0 THEN A$=
   "Ju(t!dmptfe!gps!uif!ojhiu/":GOTO 1000
II 4106 IF R=N17 AND (Y=32 OR Y=52) AND D
   Y<N0 THEN A$="Ju(t!dmptfe!gps!uif!ojhi
   u/":GOTO N1000
BU 4110 IF R=23 AND Y=50 THEN R=77:UL=N1:
   GOTO N1000
ZY 4111 IF R=23 AND (Y=N9 OR Y=30) THEN R
   =74:UL=N1:GOTO N1000
YM 4112 IF R=21 AND (Y=30 OR Y=31) THEN R
   =75:UL=N1:GOTO N1000
VE 4113 IF R=N17 AND Y=32 THEN R=76:UL=N1
   :GOTO N1000
TG 4114 IF R=N17 AND Y=52 THEN R=73:UL=N1
   :GOTO N1000
NQ 4115 IF R=39 AND Y=54 THEN A$="Zpv!tmf
   fq!b!mpoh!ujnf-!uifo!xblf/▌":GOSUB N4:
   GOSUB 6500:A$="":GOTO N1000
SH 4117 IF R=N8 AND Y=59 THEN R=81:UL=N1:
   GOTO N1000
KL 4120 GOTO N6760
FD 4160 IF Y=45 AND I(Y)=-N1 THEN A$="Xib
   u(t!uibu!bxgvm!ubtuf@!!Zpv(wf!cffo!qpj
   tpofe=▌":GOSUB N4:GOTO 7240
LD 4180 GOTO N6760
PZ 4220 IF R=N15 AND I(N5)<>-N1 AND Y=N8
   THEN A$="Hfu!mptu-!tusbohfs=":GOTO N10
   00
LG 4240 IF R=N15 AND I(N5)=-N1 AND Y=N8 T
   HEN A$="Zpv!ibwf!uif!qsjodftt=!!Dpnf!j
   o=▌":UL=N1:R=N16:GOTO N1000
OS 4260 IF (R=75 AND Y=48) OR (R=73 AND Y
   =38) OR (R=28 AND Y=27) OR (R=74 AND Y
   =N9) THEN A$="(Zft@(":GOTO N1000
BO 4280 IF R<>77 OR Y<>44 THEN 4420
EM 4300 IF I(N12)<>N0 THEN A$="(Zpv!nvtu!
   hp!opx-!cspuifs/(":GOTO N1000
XQ 4320 IF PT=N0 THEN A$="If!ufmmt!zpv!up
   !hfu!mptu=":GOTO N1000
GA 4340 A$="(Xibu(t!uif!qbttxpse-!cspuifs
   @(":GOSUB N3:? A$:A$="":INPUT PW$
WG 4360 IF PW$="DARKNESS" THEN A$="(Ifsf(
   t!uif!lfz-!cspuifs/(":US=N1:I(N12)=R:G
   OTO N1000
BZ 4380 A$="If!hsbct!b!txpse!boe!svot!!!!
   !!!!!!!!!zpv!uispvhi=▌":GOSUB N4:GOTO
   7240
AQ 4420 IF R<>43 OR Y<>N5 THEN 4480
ZB 4440 IF KS THEN A$="(Tpnfuijoh!tffnt!u
   p!cf!cpuifsjoh!!!!!!Epmops/!!If(t!cffo
   !bwpjejoh!nf/(":CL(N3)=N1:GOTO N1000
OR 4460 A$="Tif!tbzt-!(J!bn!qmfbtfe!zpv!i
   bwf!dpnf!up!wjtju!nf!upojhiu/(":GOTO N
   1000
GX 4480 IF R=45 AND Y=N20 AND DY>N0 THEN
   A$="(Uif!ljoh!upme!nf!up!lffq!bo!fzf!p
   o!!!!zpv///(":GOTO N1000
FC 4482 IF R=41 AND Y=N18 AND I(Y)=-R THE
   N A$="(J!ifbse!b!opjtf!boe!dbnf!up!efg
   foe!!!uif!qsjodftt/(":GOTO N1000
MO 4484 IF (R=42 AND Y=N19 AND DY>N0) OR
   (R=40 AND Y=N18) THEN A$="If(t!hpu!opu
   ijoh!up!tbz/":GOTO N1000
KN 4500 GOTO N6760
TL 4601 IF R=34 AND Y=N16 AND I(53)=N0 TH
   EN A$="Uifsf(t!b!epps!cfijoe!ju=":I(53
   )=-R:US=N1:GOTO N1000
IQ 4604 IF R<>42 OR Y<>21 OR I(46)<>N0 TH
   EN 4620
EL 4605 IF DY<N0 THEN A$="Uifsf(t!tpnfuij
   oh!cfijoe!ju=":I(46)=-R:US=N1:GOTO N10
   00
FI 4606 A$="Epmops!zfmmt-!(Hvbset=!!Bssft
   u!!!!!!!!uijt!nbo=(▌":GOTO 6600
KV 4620 GOTO N6760
LM 4670 IF (R=40 AND Y=N18 AND I(Y)=-R) O
   R (R=46 AND Y=N17) OR (R=42 AND Y=N19)
    THEN A$="If!tbzt!opuijoh/":GOTO N1000
OW 4681 IF Y<>57 OR I(Y)<>-N1 OR I(56)<>-
   N1 THEN GOTO N6760
SS 4682 A$="Tpnf!qsjout!nbudi!b!tfu!jo!ui
   f!lju/!!!!Uifz(sf!Epmops(t/":CL(N4)=N1
   :GOTO N1000
KR 4700 GOTO N6760
ZF 4740 IF (R=N6 OR R=N7) AND Y=24 THEN A
   $="Uif!usfft!bsf!upp!cvsofe/":GOTO N10
   00
LX 4745 GOTO N6760
JE 4780 IF I(35)=N0 AND R=24 AND Y=49 AND
    I(34)=-N1 THEN A$="Zpv!vodpwfsfe!b!us
   bq!epps=":I(35)=-R:US=N1:GOTO N1000
KT 4800 GOTO N6760
RR 4840 IF I(N10)<>-N1 OR N$<>"PAGE" THEN
    GOTO N6760
SP 4860 IF PG>N0 THEN A$="Plbz-!qbhf!jt!u
   vsofe/":PG=PG+N1:IF PG<N3 THEN GOTO N1
   000
EQ 4880 IF PG=N3 THEN PG=N0:A$(LEN(A$)+N1
   )="!Zpv!dmptfe!uif!!cppl/":GOTO N1000
LH 4940 GOTO N6760
UD 4980 IF R<>73 OR Y<>N7 THEN 5020
JR 5000 A$="Uif!nbo!btlt!uif!ipstf(t!obnf
   !boe!!!!!uifo!nbslt!ju!jo!b!cppl/":RG=
   N1:GOTO N1000
KK 5020 GOTO N6760
NK 5060 IF I(N7) THEN ? "Nothing happens.
   ":GOTO N1000
FR 5080 FOR X=N1 TO 175:SOUND N0,X,N8,N6:
   SOUND N1,X+N2,N8,N6:SETCOLOR N2,N0,14:
   SETCOLOR N2,N0,N0:NEXT X
RV 5100 SOUND N0,N0,N0,N0:SOUND N1,N0,N0,
   N0:POKE 710,N8
PF 5120 ? "A voice says, ";CHR$(34);"Ask
   and receive";CHR$(34);"."
RY 5140 ? :? "ONE WORD:";:INPUT C$
NQ 5160 IF C$="HORSE" THEN ? :? "Granted!
   ":I(N7)=R:US=N1:GOTO N1000
TK 5180 ? "You have no need for that!":GO
   TO N1000
PY 5200 GRAPHICS N18:POSITION N5,N2:POKE
   712,N14:? #N6;"SLAVE II:":POSITION N3,
   N4:? #N6;"NIMRAL'S GRACE"
YN 5220 FOR Y=N1 TO N4:FOR X=N14 TO N0 ST
   EP -0.45:POKE 712,X:SOUND N0,N20,X,N8:
   NEXT X:NEXT Y
DH 5230 POSITION N1,N10:? #N6;"BY ClaYTOn
    WAlnUM"
UT 5240 SOUND N0,N0,N0,N0:GOSUB 5740
FL 5260 GOSUB 5280:GOTO 5340
GF 5280 GRAPHICS N0:POKE 710,48:DL=PEEK(5
   60)+256*PEEK(561)+N4:POKE DL-N1,71:FOR
    X=2 TO 24 STEP N2:POKE DL+X,N6:NEXT X
JX 5300 POKE DL+N19,N6:POKE DL+21,N6:POKE
    DL+23,N6:POKE 82,N0:POKE 752,N1
EE 5320 POSITION N3,N0:? "NIMRAL'S GRACE"
   :POSITION N9,N13:RETURN
GA 5340 POSITION N4,N1:? "The alarms are
   ringing!":POSITION 24,N2:? "You've res
   cued Princess Shala from"
QJ 5360 POSITION 2,4:? "the Slavers of Go
   lgoloth, but you're":POSITION 22,N5:?
   "not safe yet!  A host of savage war-"
EQ 5380 POSITION N2,N7:? "riors are crossi
   ng the lake in hot":POSITION 22,N8:? "
   pursuit!"
LE 5400 POSITION 4,10:? "As you drag Shal
   a toward the cover":POSITION 22,11:? "
   of the forest, a cloaked figure steps"
VP 5420 GOSUB 5720:GOSUB 5280
KK 5440 POSITION N2,N1:? "from the trees.
     A survivor from":POSITION 22,N2:? "S
   hala's ambushed caravan?  You rush"
AR 5460 POSITION 2,N4:? "forward with joy
   , not noticing the":POSITION 22,N5:? "
   odor of oil drifting on the air.  The"
HQ 5480 POSITION N2,N7:? "dark figure str
   ikes a match and tos-":POSITION 22,N8:
   ? "ses it into the oil-soaked brush."
QE 5500 POSITION N2,N10:? "As a curtain o
   f flame leaps up, you":POSITION 22,N11
   :? "see the insignia of the city of"
VR 5520 GOSUB 5720:GOSUB 5280
KT 5540 POSITION N2,N1:? "Nimral's Grace
   on the figure's cloak."
UJ 5550 POSITION 22,N2:? "A traitor from
   Shala's home city? You"
WK 5560 POSITION N2,N4:? "realize that th
   e caravan's ambush was"
BD 5570 POSITION 22,N5:? "planned, that s
   omeone wants Shala out"
PG 5580 POSITION N2,N7:? "of the way.":PO
   SITION 24,N8:? "The slavers, worshippe
   rs of foul"
JX 5600 POSITION N2,N10:? "Golgoloth, are
    clamoring into their"
VD 5610 POSITION 22,N11:? "boats.  An inf
   erno blocks your path."
VT 5620 GOSUB 5720:GOSUB 5280
RH 5640 POSITION N4,N1:? "Great Nimral pr
   otect you!":POSITION 24,N2:? "How will
    you escape?  Who is the"
BN 5660 POSITION N2,N4:? "cloaked assassi
   n?  Who wants to keep":POSITION 22,N5:
   ? "Shala from her wedding?  Can you"
EA 5680 POSITION N2,N7:? "solve the myste
   ry before the assassin":POSITION 22,N8
   :? "strikes a fatal blow?  You must!"
LY 5700 POSITION 23,N11:? " THE ADVENTURE
   'S ONLY JUST BEGUN ":GOSUB 5720:POKE 8
   2,N2:GOTO 5760
DV 5720 POSITION N3,N14:? "press any key"
   :OPEN #N1,N4,N0,"K:":GET #N1,A:CLOSE #
   N1:RETURN
FF 5740 FOR X=N1 TO 200:NEXT X:RETURN
TR 5760 POSITION N3,N14:? "  one moment
    "
OM 5780 DIM VB$(NV*N4),I$(NN*SZ),A$(160),
   DT$(N8),H$(N12),IN$(N16),V$(N10),N$(N1
   0),C$(N5),U$(N19),CC$(77),D$(37)
HP 5800 DIM NN$(NN*N4),I(NN),E$(78),CL$(4
   2),V(NV),PW$(N8),L$(35)
DT 5810 DIM O$(N16),CL(N10)
JR 5815 FOR X=N1 TO N10:CL(X)=N0:NEXT X
HM 5900 O$="Plbz-!ju(t!pqfo/"
YW 5960 VB$(N1,80)="EXAMLOOKTAKEGET DROPG
   IVEOPENUNLOTASTPAWNACCUPOURKISSLISTPAI
   NREADGO  ENTEDRINTALK"
YB 5980 VB$(81,104)="MOVEFINGCLIMDIG TURN
   REGI"
YI 6000 NN$(N1,92)="AMULBOX FOOTSTATSHALS
   IGNWHITGUARSHOEBLUESWORKEY PAINDIARPAP
   ETHROKINGNIKNDOLNDOMNWARDNOTELETT"
HL 6020 NN$(93,184)="TREESTAIBROWPEACLAKE
   DIRTSHOPPAWNSTABSILVSHOVTRAPGOLDSADDRE
   GIGRAYALE FIRESLAVPLAQPRIEWINEALTA"
DS 6040 NN$(185,236)="PROCATTEGARDCHAPGOS
   SOFFISTEEBED CHAIKIT TANKSOLDTENT"
DZ 6060 I$(N1)=" ":I$(NN*SZ)=" ":I$(N2)=I
   $:GOTO 900
AV 6100 DATA 1,1,2,2,3,3,4,5,6,7,8,9,10,1
   1,12,13,14,14,15,16,17,18,19,20,21,22
JF 6120 DATA AMULET,7,JEWEL BOX,42,FOOTPR
   INTS,0,STATUE OF NIMRAL,-12,PRINCESS S
   HALA,5,SIGN,-8,WHITE HORSE,0
EB 6140 DATA GUARD,-15,SHOEMAKER,-74,BLUE
    BOOK,74,SWORD,-1,KEY,0,BLACK PAINT,35
   ,DIARY,53
UN 6160 DATA PAPER,46,THRONE,-34,KING,-46
   ,NIKNIK THE JESTOR,-40,DOLNOR,-42,DOMN
   IS,-45
AV 6180 DATA WARDROBE,-42,NOTEBOOK,72,LET
   TER,78,TREES,0,STAIRS,0,BROWN HORSE,-7
   6,ROYAL PEACEKEEPER,-28,LAKE,-5
ER 6200 DATA DIRT,-9,SHOEMAKER'S SHOP,-23
   ,PAWN SHOP,-21,ROYAL STABLES,-17,SILVE
   R BOOK,0
YI 6220 DATA SHOVEL,76,TRAP DOOR,0,GOLD P
   IECE,0,SADDLE,0,REGISTRAR,-73,GRAY BOO
   K,54,BOTTLE OF ALE,32
AK 6240 DATA FIRE,0,SLAVERS,-5,PLAQUE,0,P
   RIEST,-77,TANKARD OF WINE,39,ALTAR IN
   WALL,0,PROCLAMATION,-16
PD 6260 DATA SHOP ATTENDANT,-75,GARDEN,0,
   SMALL CHAPEL,-23,GOSSIPING GUARDS,-31,
   SMALL OFFICE,-17,STEEL DOOR,0
JW 6270 DATA BED,-39,CHAIN MAIL,31,FINGER
   PRINT KIT,52,EMPTY TANKARD,0, DEAD SOL
   DIER,-80,TENT,-8
ZL 6280 GRAPHICS N0:POKE 559,N0:POKE 703,
   4:DL=PEEK(560)+256*PEEK(561)+N4:POKE D
```

SLAVE II

```
   L+N20,130
LY 6300 RESTORE 6320:FOR X=N0 TO N19:READ
    A:POKE 1664+X,A:NEXT X
XJ 6320 DATA 72,138,72,169,192,162,10,141
   ,10,212,141,24,208,142,23,208,104,170,
   104,64
RG 6340 POKE 512,128:POKE 513,N6:POKE 542
   86,192:POKE 709,N0:POKE 710,N8:POKE 71
   2,112:POKE 752,N1
FI 6360 POSITION N1,N0:? #N6;"LOCATION:":
   POSITION N1,N4:? #N6;"YOU SEE:"
UX 6380 POSITION N1,N10:? #N6;"SOME EXITS
   :":POSITION N1,N13:? #N6;"INVENTORY:"
PE 6400 POKE 559,34:IF DY<N0 THEN POKE 70
   9,12:POKE 710,N0
MB 6420 IF FLAG THEN UL=N1:UI=N1:GOTO N10
   00
WK 6440 R=N5:WT=N0:ENT=N0:DD=N0:DY=N1:CNT
   =N0:PT=N0:UNL=N0:NB=N0:DIA=N0:GR=N0:NK
   =N0:SH=N0
HU 6445 CY=N0:SE=N0:RG=N0:PG=N0:SIL=N0:KS
   =N0:CT=N0:GD=N0
IC 6460 TURN=N1:UL=N1:UI=N1:A$=""
NZ 6480 GOTO 1000
FH 6500 DY=-DY:IF DY<N0 THEN POKE 709,N12
   :POKE 710,N0:I$(343,357)="SLEEPING DOL
   NOR":A$="Ojhiu!ibt!gbmmfo/"
XH 6505 IF DY<N0 THEN I$(305,317)="SLEEPI
   NG KING":I$(362,376)="SLEEPING DOMNIS"
OL 6520 IF DY>N0 THEN POKE 709,N0:POKE 71
   0,N8:I$(343,357)="DOLNOR         ":A$=
   "Ebzmjhiu!ibt!sfuvsofe/"
ZI 6525 IF DY>N0 THEN I$(305,317)="KING
        ":I$(362,376)="DOMNIS          "
UH 6540 GOSUB N3:? A$:A$="":RETURN
CO 6560 A$="Uif!hvbset!cvstu!jo=!!Zpv!!ib
   wf!!!!!!!dpnqspnjtfe!uif!qsjodftt!cz!c
   fjoh!!!!!dbvhiu!jo!ifs!sppn=▌"
LR 6600 GOSUB N4:A$="Zpv(sf!uispxo!joup!k
   bjm=":GOSUB N3:? A$:A$="":R=29:UL=N1:G
   OTO N1000
BK 6640 A$="Tveefomz-!zpv!tff!tpnfpof!uis
   pvhi!uif!cbst/!!If!uispxt!tpnfuijoh!bu
   !zpv-!boeju!ijut!zpv!jo!uif!ofdl=▌"
HB 6660 GOSUB 6700:A$="Bt!uif!ebsu(t!qpjt
   po!xpslt!jut!xbz!!!!joup!zpvs!tztufn-!
   zpv!sfdphoj♠f!!!!!!!zpvs!buubdlfs/▌"
KD 6680 GOSUB 6700:A$="(Zpv=(-!zpv!tdsfbn
   /!(Ju!dbo(u!cf!zpv=(▌":GOSUB 6700:GOTO
    7240
BQ 6700 GOSUB N4:RETURN
XE 6720 IF R=29 THEN GOTO N1000
AJ 6730 A$="Tveefomz-!hvbset!bqqfbs!boe!h
   sbq!zpv-!zfmmjoh!tpnfuijoh!bcpvu!opu!s
   fhjtufs.!joh!zpvs!ipstf=▌":GOTO 6600
XO 6741 IF R=29 THEN GOTO N1000
FW 6742 A$="Tveefomz!hvbset!bqqfbs!boe!hs
   bc!zpv=!!Tpnfpof!upme!uifn!bcpvu!zpvs!
   wjtju!!!!up!Tibmb(t!cfesppn=▌"
TC 6743 GOTO 6600
HV 6744 A$="Zpv(sf!upp!mbuf=!Uif!tmbwfst!
   ibwf!!!!!sfbdife!zpv=▌":GOSUB N4:GOTO
   7240
PZ 6760 ? "YOU CAN'T DO THAT!":GOTO 1100
KS 6780 GRAPHICS N0:POKE 710,N0:POKE 709,
   N10:POKE 752,N1:POSITION N2,N2
XB 6781 ? "Based on the evidence you've d
   iscov-  ered the assassin has been app
   re-"
MT 6782 ? "hended.  Unfortunately he had
   a lot   of accomplices and they're eve
   n now"
TH 6783 ? "combing the city for you.  You
    have   to leave the city (and a broke
   n"
ZA 6784 ? "hearted Shala) and search out
   the     only being who can put an end
   to"
EZ 6785 ? "the activities of the follower
   s of    Golgoloth, the diety Nimral hi
   mself."
RM 6786 ? "It will be a dangerous mission
   , but   you must succeed if Shala and
   the"
HK 6787 ? "rest of the world are ever to
   live    in peace.  The city gates clos
   e"
HD 6788 ? "behind you, and your next adve
   nture   is only a few steps down the r
   oad."
OE 6789 POSITION N14,20:? "Watch for":POS
   ITION N4,22:? "SLAVE III: THE GODS THE
   MSELVES"
YE 6790 GOTO 6790
QL 6820 TRAP 7040
FP 6840 ? "LOAD FROM TAPE OR DISK";:INPUT
    A$:IF A$="D" THEN 6900
YP 6860 IF A$<>"T" THEN ? :GOTO 6820
JK 6880 ? :? "CUE TAPE THEN PRESS RETURN
   TWICE.":INPUT A$:OPEN #N1,N4,N0,"C:":G
   OTO 6920
GY 6900 OPEN #N1,N4,N0,"D:SLAVE2.DAT":A$=
   ""
IR 6920 INPUT #N1,R,CY,PG,WT,TURN,ENT,DD,
   DY,RM
MB 6940 INPUT #N1,CNT,SIL,PT,KS,GD,CT,RG
RT 6950 INPUT #N1,UNL,NB,DIA,GR,SH,NK,SE
BK 6951 FOR X=N1 TO N10:INPUT #N1,A:CL(X)
   =A:NEXT X
NQ 6960 FOR X=N1 TO NN*SZ STEP SZ:INPUT #
   1,U$:I$(X,X+SZ-N1)=U$:NEXT X
EE 6980 FOR X=N1 TO NN:INPUT #N1,A:I(X)=A
   :NEXT X
XL 7000 IF DY<N0 THEN POKE 709,N12:POKE 7
   10,N0:GOTO 7020
DB 7010 POKE 709,N0:POKE 710,N8
FB 7020 CLOSE #N1:UL=N1:UI=N1:? :? :TRAP
   1260:GOTO N1000
GJ 7040 ? :? "NO GAME DATA SAVED!":END
LB 7060 ? "SAVE TO TAPE OR DISK";:INPUT A
   $:IF A$="D" THEN A$=",":GOTO 7120
NH 7080 IF A$<>"T" THEN 7060
NS 7100 ? :? "CUE TAPE THEN PRESS RETURN
   TWICE.":INPUT A$:A$=",":OPEN #N1,N8,N0
   ,"C:":GOTO 7140
NH 7120 OPEN #N1,N8,N0,"D:SLAVE2.DAT"
KW 7140 ? #N1;R;A$;CY;A$;PG;A$;WT;A$;TURN
   ;A$;ENT;A$;DD;A$;DY;A$;RM
IW 7160 ? #N1;CNT;A$;SIL;A$;PT;A$;KS;A$;G
   D;A$;CT;A$;RG
RE 7170 ? #N1;UNL;A$;NB;A$;DIA;A$;GR;A$;S
   H;A$;NK;A$;SE
NJ 7171 FOR X=N1 TO N10:? #N1,CL(X):NEXT
   X
GM 7180 FOR X=N1 TO NN*SZ STEP SZ:U$=I$(X
   ,X+SZ-N1):PRINT #N1;U$:NEXT X
GR 7200 FOR X=N1 TO NN:PRINT #N1;I(X):NEX
   T X
FE 7220 CLOSE #N1:? :? :A$="":GOTO 1100
BA 7240 GRAPHICS N17:POSITION N4,N4:? #N6
   ;"YOU'RE DEAD!":POSITION N3,N8:? #N6;"
   DO YOU WANT TO"
OU 7260 POSITION N2,N10:? #N6;"PLAY AGAIN
   ? (Y/N)"
YN 7280 GOSUB N11:IF A=ASC("Y") THEN RUN
EZ 7300 END
```



BOOT CAMP



Light Torch...Gird Loins ...Boot Assembler

by Karl E. Wiegers

How many of you have ever played a computer adventure game of some sort? I see a lot of hands in the air. (Great eyes, no?) And if you've ever tried to write one yourself, you quickly discovered that even a simple adventure game involves some pretty sophisticated programming. The ever-adventurous Clayton Walnum once wrote an insightful three-part series on how to design and program your own adventure games.

Blow the cobwebs off issues 39, 40 and 41 of ANALOG Computing, from early 1986, and re-read what Clayton had to say. It's okay, I'll wait here until you're done.

Back already? Then you've learned many things (or else you were watching *Dallas* reruns while you were supposed to be studying). Clayton's articles told you (among other useful stuff) that the heart of an adventure game is its "parser." The parser is the program code that lets the computer interpret commands you type and take some appropriate action. It is the parser that gives the computer some appearance of being intelligent. Of course, computers aren't intelligent in the least; good parser programmers are.

In reality, parsers are useful for much more than simply exploring dungeons. Many kinds of computer applications can benefit from a user interface that at least attempts to understand natural language communications. While it's pretty hard to get a computer to understand spoken instructions, the written word can be interpreted a bit more easily.

In the next *Boot Camp* or two, we'll see how a very simple parser can be implemented on the 8-bit Atari, using some assembly language for the time-intensive parts. You really aren't likely to write a complete application program in assembler around this word-searcher nucleus. Hence, we'll set up a simple BASIC program structure that interfaces to the machine-language parser routine, to show you how it all fits together.

Now, what subject area should we use to illustrate the fine art of parsing? Adventure games are kind of passe by now. Wait! I've got it! Imagine the kitchen of the future, automatically assembling ingredients in the quantities and sequence you specify, popping the result into the oven, and doing everything for you except eating the food. Let's write a general-purpose parser in assembly language, then cook up a BASIC program that might be used someday in Karl's Komputerized Kitchen.

The joy of parsing

There are three main aspects to a natural language-processing program or parser: 1) to take the input string apart into separate words and/or numbers, 2) to attempt to identify the individual words by looking them up in a vocabulary list, and 3) to try to understand what the input "means"; that is, see if the words identified in the input string constitute a recognizable instruction that can be executed by the program.

Let's look at all three of these functions in more detail.

Dissecting the input

The basic idea of natural language processing is that the user (who is presumably a human being of some sort) can present instructions or queries to the computer much as he would communicate with another human being. I'm sure you recognize how amazingly complex this kind of communication really is. After all, you use all sorts of shortcuts in your verbal and written communications, yet other people who know the same language can usually figure out what you're trying to say. We have to be pretty creative to do something similar with a microcomputer.

Take a simple instruction of the sort you might give to a computerized kitchen when you want to bake a cake: "Slowly mix in two cups of brown sugar." Most of you should have a picture in your mind of what this means. But how do we get the computer to understand it?

The first step is to break the input string into separate words. The simplest way to do this is to look for blank characters as delimiters between words. But what if the program user entered more than one blank between words, or used a punctuation mark (comma, semicolon, period, etc.) to separate words instead of (or in addition to) the blank? For simplicity, we'll decree that only single instructions can be entered. This means we don't have to look for complex sentences such as, "Melt the butter, then stir in the flour." So, most punctuation marks can be disregarded.

However, we can't just ignore periods. What if you want to add 3.5 cups of flour? The period here is really a decimal point in a number. Obviously, when we split the input string into words, we must distinguish numbers from true words that we'll be trying to find in the vocabulary list.

The moral of the story is that the natural language interface involves some "preprocessing" of the user's input. This step discards symbols like certain punctuation marks and builds a list of words for which we must search in the known vocabulary. Any numbers or other anticipated special character strings will be identified and set aside until we get to Step 3, in which we try to make some sense out of the input.

The preprocessor can be a part of the parser code itself or it can be a separate routine. For simplicity in this example, I'll put the preprocessor into the BASIC program. If execution time is critical, you would want to recode it into assembler, but BASIC will be fine for our purposes.

Do I know you?

Once your preprocessing step has come up with a list of words from the user's input string, we need to see if the words are "known" to the program. Your vocabulary list should be separate from the parser code itself, since a general-purpose parser routine could then be used for many applications having different vocabularies. The limited RAM

In reality, parsers are useful for much more than simply exploring dungeons.

You have to consider how much latitude you wish to give the user regarding different ways to enter equivalent commands.

in 8-bit microcomputers can really restrict the size of the vocabulary in a given program, because you still need some memory for the rest of the program.

The simplest approach is to put all the words you want the program to recognize into the vocabulary list. Some alternative techniques provide more efficient use of memory, thus allowing larger vocabularies. Have you ever noticed that some spelling-checker programs appear to require far less memory than it seems like they need to handle; say, 30,000 words? Data compression methods and clever algorithms can be used to substantially reduce the amount of memory consumed by a block of information. However, we'll leave such techniques to the experts and stick to the brute force approach.

Another consideration is how much latitude you wish to give the user regarding different ways he can enter equivalent commands. For example, do you wish to distinguish between uppercase and lowercase letters? This can be important for proper names, like in a quiz on U.S. presidents. Do you want the user to be able to get away with a certain degree of misspelling? One way to handle this is to add anticipated misspellings of particular words to the vocabulary. A more sophisticated approach uses some algorithm to determine just how closely words must match vocabulary entries to be considered acceptable.

In today's example, we'll translate all lowercase letters to uppercase, and the vocabulary entries will all be in uppercase. Only exact matches with vocabulary entries will be accepted.

Yet another one of many characteristics of human, that is, interpersonal communication is that we tend to be more or less, I mean pretty much, wordy lots of the time, you know? Think back to "slowly mix in two cups of brown sugar." The words "in" and "of" certainly are superfluous to the meaning of this instruction. Prepositions and articles (a, an, the) can generally be ignored without disturbing the meaning of an instruction. Hence, we'll leave them out of the vocabulary. A word of warning: Be very careful with negation words like "not" and "don't." "Don't boil the milk" is rather different from "boil the milk"!

How about words like "slowly" and "brown"? Adjectives and adverbs like these *can* be important, but not necessarily. The specific application dictates whether the actions to be taken depend on the presence of modifiers like these in the command string.

Another important aspect of building a vocabulary is the handling of synonyms. Instructions to "add flour" and "add sugar" are obviously different. But are there differences between "add sugar," "mix sugar," "stir sugar," "beat sugar," "fold sugar" and so on? If not, these instructions are all equivalent. So, even though our parser has to locate the verb (add, mix, etc.) in the vocabulary, only one piece of program logic is required to handle all these inputs.

A token gesture

Okay, so we've split the input string into words and found the words in the vocabulary list. Each word is assigned a numeric value, or "token." Each category of vocabulary entries will have a different set of tokens, and specific arrangements of tokens will make up valid instructions. Synonyms are given the same token.

Listing 1 illustrates what I mean. This is an Atari BASIC program that creates the vocabulary file (VOCAB.DAT) for our computerized kitchen example. You can modify this program to create vocabulary files for other applications by changing the DATA statements in Lines 1000-1120. The DIM statements in Line 100 limit the length of a single vocabulary entry to 20 characters and the total length of the vocabulary file to 2,000 bytes, but of course you could change these restrictions.

The first block of vocabulary words (Lines 1000-1030) pertains to ingredients that we think someone might want to use in describing a recipe. FLOUR is assigned the token 1, SUGAR is 2 and so on. Notice that I'm regarding BUTTER, MARGARINE and SHORTENING as equivalent ingredients, so they all are given the same token, a 5 (Line 1010). In the adventure game sense, these words correspond to the nouns that could be entered in a simple two-word command.

Another section of our vocabulary concerns operations (verbs) the user might want to perform while cooking up something tasty. All of these have tokens in the range 20-29 (Lines 1040-1060). Again, some of them are considered to be synonymous (MIX, STIR, ADD, FOLD), and instructions containing any of these words will be handled in exactly the same way by the evaluator part of the parser.

Vocabulary words with tokens in the 30-39 range refer to the units on some meaningful numbers that could be part of a command. These units refer to cooking time (HOURS, MINUTES) or temperature (DEGREES). Words with tokens in the 40s are units pertaining to the quantities of ingredients that are to be added (CUPS, TSP and so forth).

When we actually get to the part of the parser that determines whether a valid instruction was entered, the program will be looking at tokens, not at actual words. Certain patterns of tokens constitute valid commands. For example, suppose the instruction string entered said, "ADD 2 COCOA." The parser would tokenize this into ingredient = 8, amount = 2 (identified as a number), and operation = 21. However, the parser logic should recognize that something is missing: units. "ADD 2 *what* COCOA?" Cups? Ounces? Tablespoons? It makes a difference in the final product, or so I've heard. Hence, "ADD 2 COCOA" would be flagged as an invalid instruction, because no units were specified. More about the third portion of the parser next time.

Vocabulary building

Enough preliminaries; let's look at some more code! I said that Listing 1 is a utility program for creating a file containing a vocabulary list. The entire vocabulary list is treated as one giant character string variable, VOCAB$. For convenience of editing and for ease of reading the file, the contents of the string are written out to the VOCAB.DAT file in 40-byte records, in Lines 220-300.

The data statements in Listing 1 contain the individual vocabulary entries and their corresponding tokens. A complete vocabulary entry in string VOCAB$ consists of: one character whose ATASCII value equals the number of characters in the word (Lines 130-140); the word itself (Line 150); and a character whose ATASCII value equals the token value for that word (Line 160). This method for storing the vocabulary limits you to 255 unique tokens, but if you needed more, you could go to a two-byte representation for the tokens; 65,535 tokens should be adequate.

An example: MARGARINE has a length of nine characters and a token value of 5. The vocabulary entry for this word consists of CHR$(9) (Control-I), MARGARINE and CHR$(5) (Control-E). Make sense?

Line 1130 marks the end of the vocabulary data with an exclamation mark and a token value of 0. If the vocabulary searching part of the parser gets to the end of the vocabulary list without a match, a token of 0 is returned.

The word quest

Next month we'll look at the preprocessing and evaluator parts of the parser. For now, you'll have to settle for something simpler. Listing 2 is a BASIC program that simply loads the word-finder machine-language parser routine into RAM, reads the VOCAB.DAT file into string variable VOCAB$, lets you enter a word at the keyboard, and tells you if the word you entered is found in the vocabulary list. (Enter QUIT to exit.) This is a useful way to test whether there are any errors in your vocabulary file. For today, Listing 2 will serve as a framework for illustrating how the machine-language routine operates.

Line 40 of Listing 2 DIMs the VOCAB$ variable to the actual length of the vocabulary file or thereabouts. The 40-byte records from VOCAB.DAT are read in Lines 140-190.

Oh, yeah, I almost forgot: Boot Camp is supposed to be about 6502 assembly language. Well, look at Listing 3. This is the kernel of the parser, the routine that searches for a particular word in the vocabulary file. It produces only 79 bytes of object code. Shorter than you expected, eh? Well, it really isn't doing anything particularly fancy. Of course, if you were to write the entire preprocessor and evaluator parts of the parser in assembly language, you'd be talking about some serious code. The preprocessor could be written so as to be generally useful in any natural language program. However, the evaluation code is necessarily specific to each application.

If you were to write the entire preprocessor and evaluator parts of the parser in assembly language, you'd be talking about some serious code.

The machine-language routine in Listing 3 is intended to be called from BASIC by means of the USR function. It is relocatable, so it can be loaded at any address you like. Assemble this listing and create a disk file called PARSER.OBJ.

Listing 2 reads the machine-language code from PARSER.OBJ. You may recall that binary (object code) files contain six bytes of header information. Lines 60-80 of Listing 2 read these six bytes and throw them away. Lines 90-120 read the actual object code and load it into a string variable called ML$. An alternative approach is to read the object code and poke it into some safe place in RAM. Since the code is relocatable, this can be secure anywhere, so long as you know the starting address.

The assembly routine uses six RAM locations for specific purposes. Zero-page bytes $CB-$CE (decimal 203-206) are needed for the indirect indexed addressing mode, as we have seen so many times before. Location $6FE (decimal 1790) contains the length of the word being sought in the vocabulary; this serves as input into the machine-language routine. Location $6FF (1791) contains the word's token value (or a zero if not found); this is the parser's output. You can change these if they conflict with other uses for those

It's not a bad idea to do some error checking to make sure that the accumulator contains a "2" after the PLA operation.

addresses in your own programs.

To call this machine-language routine, first poke the length of the target word (in variable WORD$ in Listing 2) into location 1790 (Line 240 of Listing 2). Then call the machine-language routine with the USR function as shown in Line 250, specifying the addresses of the ML routine itself, the vocabulary variable string, and the target-word variable string. Faster than a speeding bullet, the token value for the word appears at Address 1791, unless the contents of WORD$ aren't found in VOCAB$, in which case a zero appears in address 1791. Pretty simple, eh?

Let's look at the assembly code a little more closely. Lines 450-530 in Listing 3 illustrate how to handle arguments passed from BASIC to machine language. These, you may recall, are passed via the stack, in two-byte chunks. The PLA in Line 450 removes a one-byte counter of how many arguments were actually passed. It's not a bad idea to do some error checking to make sure that the accumulator contains a "2" after the PLA operation; otherwise, a computer lockup is likely. Next on the stack are the high byte and low byte of Argument 1 (address of VOCAB$), followed by the high byte and low byte of Argument 2 (address of WORD$).

The searching algorithm is really very simple. It begins by comparing the length of the target word to the length of the current vocabulary entry being pointed to by VOCAB (Lines 680-700, with a branch down to Line 790). If the lengths are different, the words obviously don't match, so control passes from Line 810 to label NEXTWORD at Line 1090. Lines 1100-1320 simply change the contents of VOCAB to point to the next word in VOCAB$, by skipping ahead a number of bytes equal to the length of the current word plus 2 (one for the length, one for the token).

If the target length did match the current vocabulary entry length, a character-by-character comparison is done in Lines 820-960. This comparison actually starts with the last character in the word and works backwards. If all characters match (*Ta-da!*), we have a hit. Lines 970-1010 fetch the token value at the end of the current vocabulary entry, store it at address RESULT ($6FF, 1791), and return to the BASIC program. If the entire vocabulary list is searched with no match, RESULT contains a zero (Line 710).

Conclusion

So there you have it. A very simple assembly-language program for searching an arbitrary list of vocabulary entries to see if a target character string can be identified. Next time, we'll see a way to package this vocabulary searching part of the parser with preprocessor and evaluator routines in BASIC to show just how smart a program has to be to make Karl's Komputerized Kitchen a reality.

Acknowledgement

I'm indebted to Dr. Bruce Argyle of Mad Scientist Software for sharing his parser code and concepts with me. Thanks, Bruce.

Listing 1: BASIC

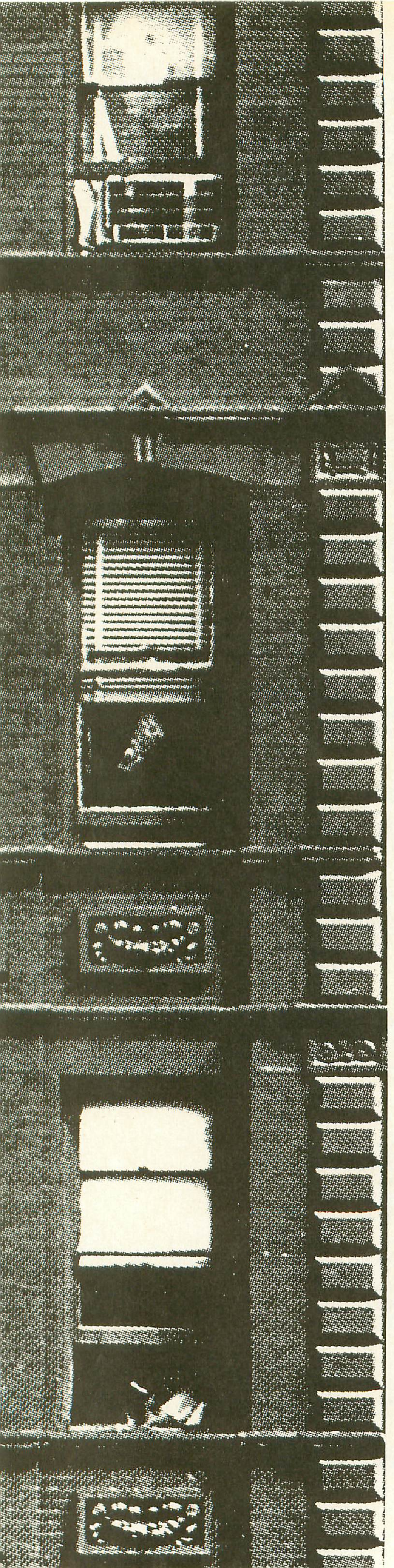
```
IE 10 REM Program name:  VOCAB.BAS
AT 20 REM Utility program to build
FL 30 REM vocabulary list file for parser
ZM 40 REM demo program in "Boot Camp"
BC 50 REM
DD 60 REM by Karl E. Wiegers
BE 70 REM
KD 80 REM First build VOCAB$ string
BG 90 REM
BY 100 DIM TERM$(20),VOCAB$(2000),TEMP$(4
   1)
YU 110 A=1:PRINT "Building vocabulary..."
ZS 120 READ TERM$,TOKEN
AS 130 INLEN=LEN(TERM$)
IK 140 VOCAB$(A,A)=CHR$(INLEN)
MO 150 VOCAB$(A+1,A+INLEN)=TERM$
OD 160 VOCAB$(A+1+INLEN,A+1+INLEN)=CHR$(T
   OKEN)
AZ 170 IF TERM$="!" THEN GOTO 200
YG 180 A=A+INLEN+2
MU 190 GOTO 120
DU 200 REM Then print data to file
PB 210 PRINT "Saving in file...."
BV 220 OPEN #2,8,0,"D:VOCAB.DAT"
WM 230 MAX=INT(LEN(VOCAB$)/41)+1
UQ 240 FOR I=1 TO MAX-1
LV 250 TEMP$=VOCAB$(40*I-39,40*I)
WY 260 PRINT #2;TEMP$
GG 270 NEXT I
XG 280 TEMP$=VOCAB$(40*MAX-39,LEN(VOCAB$)
   )
XE 290 PRINT #2;TEMP$
LL 300 CLOSE #2
XM 310 PRINT "All done!"
NW 320 END
EC 1000 DATA FLOUR,1,SUGAR,2,OIL,3,MILK,4
AP 1010 DATA BUTTER,5,MARGARINE,5,SHORTEN
   ING,5
AF 1020 DATA NUTS,6,WATER,7,COCOA,8
WU 1030 DATA EGG,10,EGGS,10
ZC 1040 DATA MIX,21,STIR,21,FOLD,21,ADD,2
   1
SB 1050 DATA WHIP,22,BEAT,22
JR 1060 DATA PREHEAT,23,COOK,24,BAKE,24
AI 1070 DATA HOURS,31,MINUTES,32,DEGREES,
   32
IV 1080 DATA CUP,41,CUPS,41,C,41
BO 1090 DATA TEASPOON,42,TEASPOONS,42,TSP
   ,42,TSPS,42
ZX 1100 DATA TABLESPOON,43,TABLESPOONS,43
UK 1110 DATA TBSP,43,TBSPS,43
PM 1120 DATA OUNCE,44,OUNCES,44,OZ,44,OZS
   ,44
VF 1130 DATA !,0
```

Listing 2: BASIC

```
NN 10 REM Sample program to demonstrate
UD 20 REM vocabulary searching for words
XD 30 REM entered by user
BK 35 REM
AS 40 DIM VOCAB$(280),TEMP$(40),WORD$(20)
   ,ML$(79)
NN 50 OPEN #2,4,0,"D:PARSER.OBJ"
FP 60 FOR I=1 TO 6
EF 70 GET #2,A
IW 80 NEXT I
PJ 90 FOR I=1 TO 79
CB 100 GET #2,A
AR 110 ML$(I)=CHR$(A)
FV 120 NEXT I
LP 130 CLOSE #2
ZY 140 OPEN #2,4,0,"D:VOCAB.DAT"
OC 150 FOR I=0 TO 6
RE 160 INPUT #2,TEMP$
YJ 170 VOCAB$(I*40+1)=TEMP$
GH 180 NEXT I
MB 190 CLOSE #2
TZ 200 PRINT CHR$(125)
AO 210 PRINT "Enter a word (QUIT to exit)
   :"
XA 220 INPUT WORD$
IM 230 IF WORD$="QUIT" THEN STOP
ZL 240 POKE 1790,LEN(WORD$)
JH 250 X=USR(ADR(ML$),ADR(VOCAB$),ADR(WOR
   D$))
GP 260 IF PEEK(1791)=0 THEN PRINT "Sorry,
    I don't know ";WORD$
BI 270 IF PEEK(1791)>0 THEN PRINT "Token
    for ";WORD$;" is ";PEEK(1791)
TT 280 PRINT
MU 290 GOTO 210
```

Listing 3: Assembly

```
0100       .OPT OBJ,NO LIST
0110 ;
0120 ;Vocabulary searching routine, to
0130 ;be used as part of a natural
0140 ;language parser program
0150 ;
0160 ;by Karl E. Wiegers
0170 ;
0180 ;This machine language subroutine
0190 ;is designed to be called from a
0200 ;BASIC program.  It takes two
0210 ;arguments:  the address of the
0220 ;vocabulary data string, and the
0230 ;address of the variable that
0240 ;contains the word being searched
0250 ;for, like this:
0260 ;
0270 ;X=USR(loc,ADR(VOCAB$),ADR(WORD$)
0280 ;
0290 ;
0300 VOCAB = $CB
0310 WORD =  $CD
0320 LENGTH = $06FE
0330 RESULT = $06FF
0340 ;
0350 ;routine is orged at $600, but is
0360 ;relocatable
0370 ;
0380       *=  $0600
0390 ;
0400 ;-------------------------------
0410 ;set up arguments passed from
0420 ;BASIC, into page 0 variables
0430 ;-------------------------------
0440 ;
0450       PLA
0460     PLA          ;pointer to start
0470     STA VOCAB+1 ;of vocabulary
0480     PLA          ;character string
0490     STA VOCAB
0500     PLA          ;pointer to start
0510     STA WORD+1  ;of word being
0520     PLA          ;searched for in
0530     STA WORD    ;vocabulary list
0540 ;
0550 ;--------------------------------
0560 ;search routine starts here with
0570 ;next word being pointed to by
0580 ;VOCAB variable; branch to label
0590 ;ANALYZE to look for match; if
0600 ;no match, return to here; last
0610 ;'entry' in VOCAB$ has token of
0620 ;0, so store that in RESULT and
0630 ;return to BASIC program
0640 ;--------------------------------
0650 ;
0660     CLD
0670 BEGIN
0680     LDY #0
0690     LDA (VOCAB),Y
0700     BNE ANALYZE
0710     STA RESULT
0720     RTS
0730 ;
0740 ;--------------------------------
0750 ;see if length matches that of
0760 ;next word in vocabulary
0770 ;--------------------------------
0780 ;
0790 ANALYZE
0800     CMP LENGTH  ;lengths match?
0810     BNE NEXTWORD ;no, go on
0820     LDY LENGTH  ;yes,check chars
0830 ;
0840 ;--------------------------------
0850 ;compare characters in target
0860 ;word with those in current word
0870 ;in vocabulary
0880 ;--------------------------------
0890 ;
0900 CYCLE
0910     LDA (VOCAB),Y ;get next char
0920     DEY           ;and compare to
0930     CMP (WORD),Y ;target word
0940     BNE NEXTWORD ;no match,go on
0950     TYA           ;matches, check
0960     BNE CYCLE    ;next char
0970     LDY LENGTH   ;found! point to
0980     INY           ;token value, get
0990     LDA (VOCAB),Y ;it, and store
1000     STA RESULT   ;in RESULT byte
1010     RTS           ;all done, so exi
t
1020 ;
1030 ;-------------------------------
1040 ;skip to next word by adding
1050 ;length of current word to
1060 ;pointer to vocabulary list
1070 ;-------------------------------
1080 ;
1090 NEXTWORD
1100     CLC
1110     LDY #0
1120     LDA VOCAB
1130     ADC (VOCAB),Y
1140     STA VOCAB
1150     BCC NOINC1
1160     INC VOCAB+1
1170 NOINC1
1180     CLC
1190     LDA VOCAB
1200     ADC #2
1210     STA VOCAB
1220     BCC NOINC2
1230     INC VOCAB+1
1240 NOINC2
1250 ;
1260 ;-------------------------------
1270 ;continue search with next word
1280 ;in the vocabulary
1290 ;-------------------------------
1300 ;
1310     CLC
1320     BCC BEGIN
```



Database DELPHI

Once you've downloaded what's accumulated in the Atari Group's databases, maybe you'll want to submit some of your own files.

by Michael A. Banks

As I write this, it's 103° outside. The ground is riddled with crackly brown stuff that used to be soft green grass. My air conditioner is chugging away, giving me occasional pause when the compressor cuts in and the lights dim. (Yes, I do have a surge protector and 100-amp service; the voltage drops are the result of the high demand of constantly running air conditioners throughout the neighborhood. . . *Sigh*.)

But that's okay—I'm dealing with it creatively. Each problem is a challenge. We have a ban on watering, so I catch the air-conditioner condensation with a bucket and use that to water our small flower garden and large rosebush (Japanese beetles gotta eat too!) And because Southwestern Ohio, where I live, is particularly hard hit by the drought, the water is shut off during the day on occasion, which makes it tough when I've forgotten to stock up on bottled water. But I can melt a couple trays of ice cubes for fresh drinking water and go on enjoying my iced tea without interruption.

So, it's not so bad—in fact, meeting challenges creatively is kinda fun. But I hope a cold front moves in soon to help dampen the effects of Father Sol's fusion plant so I can turn my creative energies to more interesting pursuits. . . .

Of course, all that's history now, as you read this. Those of you who live in the temperate zones (and even you in the Southwest, Texas and Florida) are enjoying cooler weather and earlier sundowns. And like me, you're probably spending less time on heat survival and more on fun and creative activities.

As I noted in October's column, this time of the year is a good time to catch up on programming, disk-housekeeping and checking out what's accumulated in the Atari Group's databases while you were on vacation or busy coping with the drought. And, now that you have the time, maybe you'll want to share some of your own creativity with other Atari group members via those same databases. So, as promised, this month's column focuses on downloading files from the Atari database, as well as submitting files of your own.

Downloading...step by step

If you read October's column, you know that you must go through two steps preparatory to downloading a database file: Select a database and read the file's description. To select a database, type DA at the Atari Group's main menu, then the name of the database that you wish to access (or, simply type DA and the first few letters of the database name together: DA GEN).

Once you're at the database prompt, type READ to see the description of the database item. Now, if you just type READ when you enter the database, you're going to see the description of the first (newest) file in the database. So, you might want to do a directory first (press Return), or search for database items by keyword as I described last month.

In case you missed the October issue, here's a thumbnail sketch of the search process: Type SEARCH and a keyword, and you'll be presented with a list of items containing the specified keyword. All commands—including READ—will then operate on the selected items; in effect, you will have created a temporary subset of the database that contains only the files matching your keyword. You can also narrow or expand your search, by selecting either command from the database menu.

Once you've read a file's description you'll be at the ACTION prompt. Here's an example, of what you'll see (in this case, I typed DA GEN at the Atari Group's main menu, to reach the general database, after which I typed READ BOOT CAMP #61):

```
<<< ANALOG Databases >>>

General Interests Menu:

Directory of Groups    Set Topic
Read (and Download)    Submit
Search (by Keyword)    Workspace
Narrow search          Help
Widen search           Exit

DBASES:Gen> (Dir, Read, Set, Exit) READ BOOT CAMP #61
Name: BOOT CAMP #61
Type: PROGRAM
Date: 7-JUN-1988 20:39 by ANALOG4

Karl Wiegers supplies a handful of useful graphics macros in
June's Boot Camp.  Please see ANALOG #61 for complete
documentation.

This program Copyright 1988 by ANALOG Computing.

Keywords: PROGRAM, WIEGERS, GRAPHICS, ASSEMBLY, JUNE, #61

Contents:

   1  MAC/65 FILE (Size: 4096 Count: 9)
   2  MAC/65 FILE (Size: 5248 Count: 8)

ACTION> (Next, Down, Xm, List)
```

The Action menu

The ACTION prompt is one of the few places on DELPHI where a menu is not automatically displayed, even if you are running with menu prompting. (The ACTION prompt shows the major commands: "Next," which takes you to the next file; "Down," which sends the designated file to your computer using your default download method (more on this in a bit); "Xm," which initiates an Xmodem file transfer to your computer; and "List," which displays the designated file as unformatted ASCII text.)

To see the ACTION menu, type MENU or ?. This menu is displayed:

```
ACTION Menu:

Next Group/File            Show Files in Group
Download                   Set Topic
List (Unformatted)         Help
Display (Word-wrapped)     Exit
Description of Group
```

The first three items on the menu (Next, Download and List), I've just explained. Display is like List, except that the file is displayed wordwrapped and with "More?" prompts. (Note: List and Display should normally be used only with text files.) Description of Group redisplays the current item's description for your review. Show displays a list of the files in the group (as you've noticed in the preceding description, there can be more than one file). Set Topic allows you to select a different database. Help and Exit are self-explanatory.

All of these commands are available whether you display the menu or not.

Download commands

To download a file, simply type DOWNLOAD, and DELPHI will send the file to you using your default file-transfer method. You can view your current default-file download method by typing /FX_METHOD.(The default method is set at the "Set Preferences" selection at the Atari Group's main menu, via the SETTINGS selection in Workspace, or in the SETTINGS area of USING DELPHI).

In multiple-group database items (as in the "BOOT CAMP #61" example), the first file is sent, unless you specify a different file by number.

If you don't want to use your default file-transfer method (perhaps you want to try Xmodem, but your default is Kermit), you can change it temporarily in one of three ways:

1. Type DOWNLOAD followed by the name of the file-transfer method you wish to use. (Example: DOWN XMODEM.)

2. Type /FX_METHOD followed by the name of the file-transfer method you wish to use. (Example: /FX_METHOD XMODEM.)

3. Type DOWNLOAD MENU to view this menu:

```
Download Method Menu:

XMODEM (128 byte blocks)     RT Buffer Capture
Kermit                       YB (YMODEM batch)
WXMODEM (Windowed XMODEM)    Help
YMODEM (1024 byte blocks)    Exit
Buffer Capture

DOWNLOAD> (Xm,Kermit,WXm,Ym,Buff,RT,YB)
```

Database DELPHI

At this point, you can enter your choice of file-transfer methods and proceed with the download. DELPHI will tell you to prepare your computer to receive the file and provide any other information necessary (like how to abort the transfer), then signal you to begin the necessary procedure to receive the file at your end.

If your terminal software has the capability, I highly recommend using Xmodem, Kermit or Ymodem over ASCII/buffer capture, even for text files. There's much less chance of "garbage" getting into a file with one of those error-checking protocols.

By the way, if you have a comment on a file, you can E-mail it to the person who uploaded the file right at the ACTION prompt. Simply type REPLY, and a DELPHI mail message will be addressed to the membername listed as the file's "Owner" (that's the person who submitted it to the database), with the subject header filled in. Type your comments, then CNTL-Z to send the message, and you're back at the ACTION prompt.

Submit!

As you know, a major portion of the files in the Atari Group's databases is contributed by DELPHI members like you. The files are placed there through a process called "submitting." At present, you can receive free time when you submit files (select "Request Free Upload" at the Atari Group's main menu for more information).

Before you can submit a file to a database, the file you wish to submit must be in your Workspace. Files are put in Workspace in one of three ways: by extracting them from E-mail, by creating them (text files only) with the CREATE command or by uploading them.

You can upload files to your Workspace using almost any file-transfer method: ASCII upload, Xmodem, Kermit, Ymodem, Ymodem Batch or WXmodem. The method you use is up to you. (Type HELP UPLOAD, HELP XUPLOAD, etc., at the Workspace prompt for more information on uploading files.)

Preparing a file for submission

If you're submitting a program or binary data file, you can upload the file with no special preparation, other than, perhaps, archiving it to save upload and download time.

If you're uploading a text file, there are several things to keep in mind. For openers, I recommend that you upload long text files, rather than try to create them online—the DELPHI text editors are rather difficult to use compared with your average word processor. For files which may be read online—even partially—you should keep your line width (number of columns) down to 70 or so, to accommodate wordwrap. Also, it's a good idea to begin any text file with .lt (this is a command that causes DELPHI to display any text online formatted in the same way you uploaded it).

Oh yes—text files should generally be 7-bit ASCII.

The submission process

Assuming you have a file in your Workspace to submit, the next step is to type SUBMIT at any database prompt (that's at the database name prompt, which says DBASES: and the first few letters of the name of the database, like this: DBASES:Gen>). Or, better yet, type SUBMIT at the Workspace prompt to submit it right after you upload it.

The actual submission process is merely a matter of responding to a series of prompts, and you can abort the process at any time by entering CNTL-C. When you type SUBMIT, you'll be asked if you are ready to submit a file. Type *Y*. Next, you'll be asked to enter the number of files you are submitting (usually 1). If you are submitting more than one file, DELPHI will ask if the files are so related that they can be entered as a group in the database. Answer *Y*.

Additional prompts will request the following information:

The TYPE of file you are submitting. This tells DELPHI whether the file you are submitting is a Program, Article, etc. If you are submitting a program file, DELPHI will ask you if the file must have a special download filename (this is normally the same filename as on your disk).

The TOPIC of the file. This is the database topic under which your submission will be stored. (You need not be in the database topic to which you wish to submit a file.)

A DESCRIPTION of the file. This is the file description that is displayed when the READ command is given by members in the database. It's a good idea to begin this with .lt on a line by itself, just as you would begin any text file; that way, the description will be displayed as you enter it. Make this as short as possible—but don't scrimp on necessary details! (And if the file is archived, be sure to include this fact, along with the name of the program necessary to de-archive it.)

The DISPLAY NAME of the file. This is the name that is displayed in the database directory. It can be up to 32 characters in length and should be as descriptive as possible.

KEYWORDs for the file. You'll have to select one major keyword from a list of six keywords established for the database topic in question (type ? at the Keyword prompt to see this list). After the initial keyword specification, you can add several keywords of your own choice. Remember that keywords are search aids, so think of the major keywords under which other DELPHI members might search for your file. If you're submitting a printer utility, you might use the keywords PRINTER, UTILITY, the name(s) of the printer(s) with which the utility operates and any other distinguishing features of the program.

After you've entered the all of this information, you'll be prompted for the WORKSPACE FILENAME of the file you are submitting. If you forget the filename, type a question mark or press Return at this prompt; DELPHI will display a list of your Workspace files.

After you have provided all the necessary information, you will be asked if you wish the file you have submitted deleted from your workspace. Answer YES or NO, as you wish, and the submission process will be completed.

The file you submit will not be immediately available in the database to which you submitted it. It is temporarily stored in a special preview area, to await review by the Atari group manager. The manager will review your submission, edit the description or keywords if necessary, and have it in the appropriate database and available to all Atari group members within a day or two.

Conference reminder

Don't forget the real-time conference held in the Atari Users' Group every Tuesday at 10 p.m., EST. To join, type CO at the SIG menu, and then type WHO at the conference menu. You'll see a conference group name, with a list of the members participating beneath the group name. The name will be preceded by a number. To join, simply type JOIN followed by the number and you're in! Type to talk. If you get stuck, ask those in the conference group for help or type /HELP.

Be a Game Designer!

Ever wanted to design your own video game? You're on! MATRAT (Matthew Ratcliff) and the ANALOG's Atari Users' Group are giving you the opportunity to help write a video game. MATRAT has volunteered to do the programming, and a thread in the Forum contains all the design elements and suggestions to be used in the game. Things are progressing nicely, and at this point the action gamers seem to have the upper hand over the adventure gamers. Your input is needed! To get in on the fun, go to the Forum and read Message 37596. Then type FOLLOW to read what everyone else has to say about the design.

Once you have a handle on what's happening, add your comments with ADD or by typing REPLY 37596.

Once the game is under software construction, the thread will be posted in the database. Once completed, the game design may be a feature game and article in ANALOG Computing.

That's it for now. Next month: An eclectic mix of tips and tricks. Until then, see you online!

In addition to having published science fiction novels and books on rocketry, Michael A. Banks is the author of DELPHI:The Official Guide *and* The Modem Reference*—both from Brady Books/Simon & Schuster. Look for his articles on telecommunications and tips on using DELPHI in the Atari Users' Group databases. You can contact Banks on DELPHI by sending E-mail to membername KZIN.*

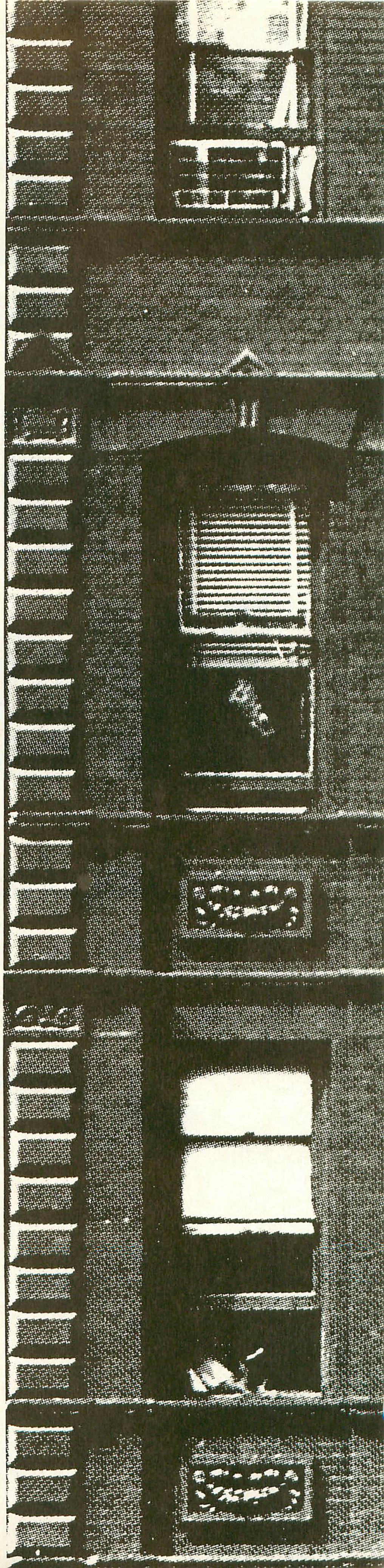
Make the DELPHI Connection!

As a reader of *ANALOG Computing,* you are entitled to take advantage of a special DELPHI membership offer. For only $19.95 plus postage and handling ($30 off the standard membership price!), you will receive a lifetime subscription to DELPHI, a copy of the 500-page *DELPHI: The Official Guide* by Michael A. Banks and a credit equal to one free evening hour at standard connect rates. Almost anyone worldwide can access DELPHI (using Tymnet, Telenet or other networking services) via a local phone call. Make the DELPHI connection by signing up today!

To join DELPHI:
1. Dial 617-576-0862 with any terminal or PC and modem (at 2400 bps, dial 576-2981).
2. At the Username prompt, type JOINDELPHI.
3. At the Password prompt enter ANALOG.

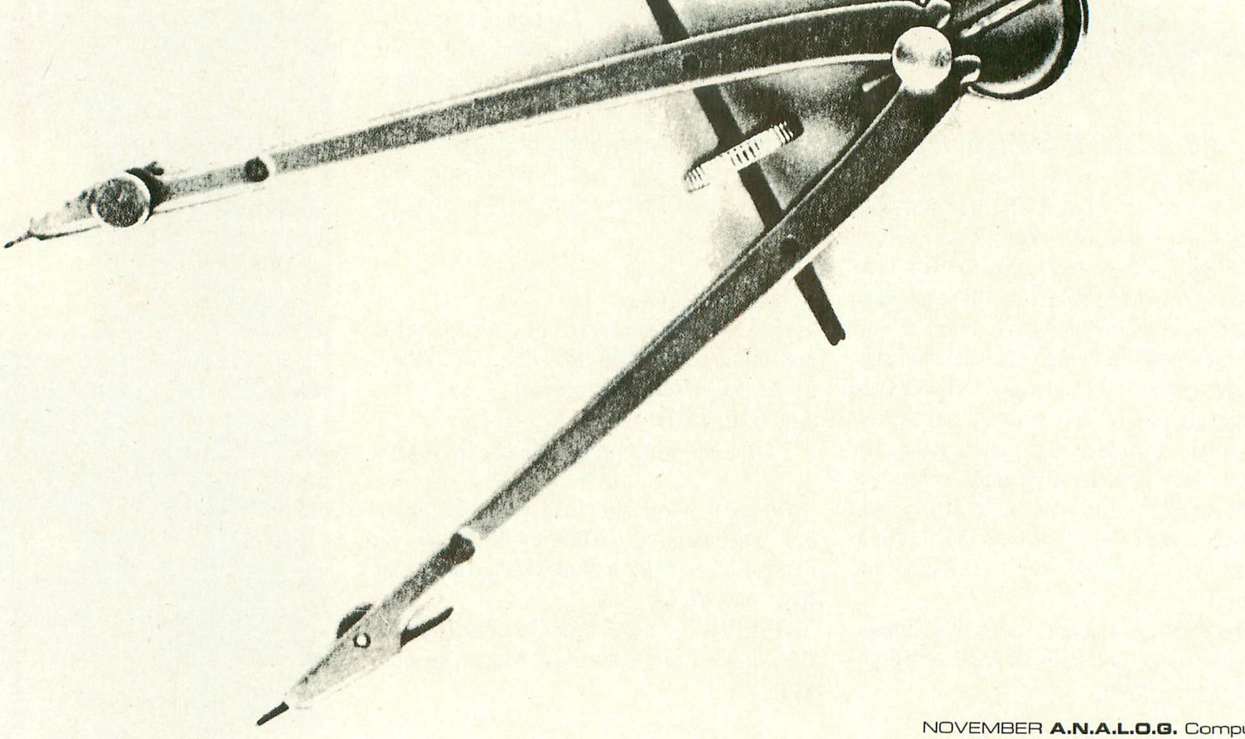
For more information, call DELPHI Member Services at 1-800-544-4005, or at 617-491-3393 from within Massachusetts or from outside the U.S.

DELPHI is a service of General Videotex Corporation of Cambridge, Massachusetts.



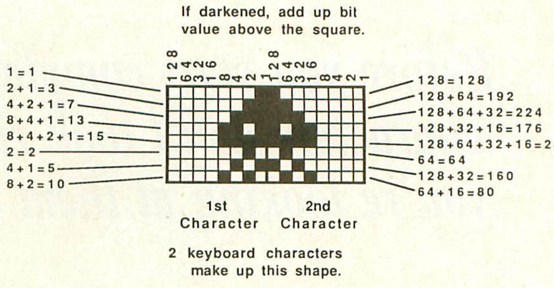
Game Design Workshop

by Craig Patchett



I take the "ground up" approach when I program a game: the graphics first, then the game logic.

Figure 1 - How to create a shape



A little character

Perhaps one of the toughest parts of programming a game is deciding where to start (you may, however, disagree). The best thing to do is begin with the most dominant part of the game, the part that everything else tends to rely on. For example, I would start with the maze if I were doing *Pac-Man*, the terrain if I were doing *Scramble* and the centipede if I were doing *Centipede*. Once these elements are in place, the rest can be added piece by piece. It's sort of the equivalent of building a house floor by floor, starting with the basement.

Of course, this is only my personal preference. You may prefer to start with the framework and then fill in the details (design the game logic first and then do the graphics). The reason I take the "ground up" approach is because in most games the logic relies on the graphics being in place, so that it can detect collisions and respond appropriately. Also, it's a lot easier to program when you can see the results on the screen.

Needless to say, we'll be programming our BASIC Invaders example from the ground up, which means we'll begin with the invaders themselves. As a result, this column will be dealing with character graphics. Please keep in mind that this may not be the case in your game. Your key element may use bit-mapped graphics or PMG. The important thing to take note of here, then, is not the fact that we're beginning with character graphics, but rather the techniques that we use to implement the character graphics.

Our first step is to take the invader shapes we came up with earlier and translate them into values that the computer can understand. As you probably already know, this is done by treating each shape as a series of bytes stacked on top of each other. Each byte consists of eight bits which, like the dots, can be turned on or off. So we treat each dot as a bit. Because the invaders are more than eight dots wide, we'll be using two characters for each invader.

So let's get started. Because this is a relatively straightforward process, we'll single-step through the first invader. The first invader is shown in Figure 1.

Now that we have our values, we're almost ready to give them to the computer. First of all, though, we have to reserve a place to put them, along with the rest of the character set. We do this with the following program lines:

```
3150 CB=PEEK(740)-4:POKE
 106,CB-4:CA=CB*256
3160 GRAPHICS 0:POKE 756
,CB
```

Now we have 1024 bytes reserved at the end of memory, and have told the computer that it will a find a character set there. So we'd better set one up quickly! To make things easier, I'll give you the complete program listing first, and then we'll go over it line by line:

```
3030 DIM MLANG$(90)
3050 DIM MOVMEM$(41):GOS
UB 29500:MOVMEM$=MLANG$
3150 CB=PEEK(740)-4:POKE
 106,CB-4:CA=CB*256
3160 GRAPHICS 0:POKE 756
,CB
3210 X=USR(ADR(MOVMEM$),
57344,CA,1023)
3220 MEM=CA+512:FOR SEC=
0 TO 1:GOSUB 32500+10*SE
C:X=USR(ADR(MOVMEM$),ADR
(MLANG$),MEM,LEN(MLANG$)
-1)
3230 MEM=MEM+LEN(MLANG$)
:NEXT SEC
3240 X=USR(ADR(MOVMEM$),
CA+128,CA+640,335)
28999 END
29500 MLANG$="hh┐Oh┐Nh┐Q
h┐Ph* )\P |h(1N P/@)PwfOf
QJ◆)Pi◆":RETURN
32500 MLANG$="♥♥♥♥♥♥♥♥┘
\¯ ┐ h◣♥@◆0p@▌P┘\¯ ┐┐◢H♥@
◆0p ♣ ◢H ↓→┬● |♣ p8xh(@ |●
┬↓→┘┤◢@(h8xp♣┘.→▌→/.0@px
_x◆♣ ┘.":RETURN
32510 MLANG$="→▌→¯┴┘@px_
x0┴0♥ ┌┐♥ ♥┐.♥ @♥◆♥@ ♣♣88
||FF":RETURN
```

Don't forget that lines 29500 through 32510 come from a previous column. In case you're wondering about the rest of the program, here's the complete explanation:

3030; You may remember MLANG$ from the program lines we generated previously. It's used for temporary storage of the machine language routines and character set data.

3050; MOVMEM$ holds a machine language routine that we'll be using throughout BASIC Invaders to move things around in memory. You'll see how it's used later in this explanation.

3150; You saw this already, but I'll explain in case it wasn't clear. Location 740 points to the top (or end) of memory. Location 106 points to where the computer thinks the end of memory is. So, we set a variable called CB to point to where the beginning of our character set will be (PEEK(740)-4, which is four pages below the top of memory, with a page being 256 bytes). Then we reset location 106 to point 1024 bytes (256*4) below the character set.

Why? Because the area of memory right after that pointed to by location 106 isn't always safe. Anyway, it isn't really that important for you to understand this. Just keep in mind that this line will reserve memory for the character set. You may also want to know that if you're using a program like BASIC A+, which also changes location 106, then you should change CB=PEEK(740)-4 to CB=PEEK(106)-4.

3160; The screen is usually kept at the top of memory, and is right now in the space we've reserved for our character set. By using a GRAPHICS command, we move it down below the new top of memory. Also, we tell the computer where our new character set will be.

3210; This line moves the regular character set to our reserved space, and is an example of using MOVMEM. In general, the way to use MOVMEM is with the following command:

```
X=USR(ADR(MOVMEM$),FROM,
TO,LENGTH-1)
```

FROM is the address of the first memory location you want to move from, TO is the address of the first memory location you

Before you start complaining that your invader characters don't look right, keep in mind that you're looking at them in graphics mode zero.

want to move to and LENGTH is the number of bytes you want to move. So, in our example, we are moving the 1024 bytes starting at location 57344 (the address of the Atari character set) to the memory area starting at location CA (the address of our character set).

3220-3230; Now we put our redefined characters into the new character set. The graphics characters being at CA+512. We've already put our character data into strings in lines 32500 and 32510, so we GOSUB to these lines, then move the data from MLANG$ to the character set. Notice that we once again use MOVMEM. You'll find that there are a lot of times that MOVMEM will come in handy, and not just the ones that we'll cover in this book.

3240; Here's MOVMEM again! Actually, this line isn't really necessary at this point, but I included it because it follows along with everything else here. All it does is move the numbers and uppercase characters into the lowercase part of the character set. Why? In graphics mode one, you have to choose between lowercase/graphics and uppercase/numbers. We want to be able to have uppercase, numbers and graphics at the same time (so we can include the score), so we simply move things around a little.

28999; That's it folks.

29500-32510; These are just the subroutines to set up MOVMEM and the redefined characters. See the previous column on how to set them up.

I know there are a few things that I haven't mentioned yet, and I'll get into them soon. First, however, why don't we make sure that this program really works. Run it, and then try typing CTRL-A, CTRL-B and so forth, all the way up to CTRL-N (preferably all right next to each other). This should give you all our invaders characters. Now before you start complaining that they don't look right, keep in mind that you're looking at them in graphics mode zero. We'll be using them in graphics mode one, and in a minute you'll see what they look like in that mode. But first, it's time to clear up a few things.

In the process of getting all of this up and working, I've conveniently neglected to fill in a few of the details. For example, you may have noticed that the first character we redefined was all zeros or a blank space. Why? Try going into graphics mode zero, and then POKE 756,226. See how the screen fills with hearts? What we just did was switch to lowercase/graphics. In lowercase/graphics, there is no space character. What the computer uses instead is the heart. So, to avoid this problem, we redefine the heart to be a space. Okay?

Well, that was easy to explain. Next up is a problem that many people run into when working with the character set: the character values. You already know about ATASCII values, right? Each character is assigned an ATASCII value between zero and 255. If you want to find out the value for a particular character, use the ASC command from BASIC (PRINT ASC("E"), for example).

Anyway, each character has a value, which tends to put the characters in a specific order. Now it would make sense that the characters would be stored in this order in the character set, right? Of course it would, but when was the last time a computer made sense? Instead, there is another order for the character set, called the internal order. There is an easy way to figure out the internal value from the ATASCII value. The following table shows how to do it:

TYPE OF CHARACTER	ATASCII	INTERNAL
Graphics	0-31	Add 64
Uppercase/Numbers	32-95	Subtract 32
Lowercase	96-127	Same

(Anything with a value greater than 127 is just the inverse of the character with the same value minus 128.)

All this does is switch the graphics and uppercase/numbers as far as order is concerned. No big deal, but you have to remember to use the internal order when dealing directly with the character set.

One more detail that tends to trip a lot of people up. Suppose you want to change a character with an internal value of *n*, and the character set begins at CA. Do you start changing bytes at location CA+n? No, because each character takes up eight bytes in the character set. That means that the character starts at CA+n*8. A silly little detail like this has left a lot of people wondering what went wrong!

Okay, now it's time to put our invaders into action. What we're going to do is put them into a long string and then just print this string on the screen. Remember, though, that there are two versions of each invader. So our string will actually hold two versions of the invader screen. We'll alternately print each version, thus getting some animation out of the invaders. This will make more sense after you try it out, so make the following changes to the previous program:

```
3030 DIM MLANG$(90),INV$
(480),DAT$(16)
3160 GRAPHICS 17:POKE 75
6,CB+2
5370 INV$="♥":INV$(480)=
"♥":INV$(2)=INV$
5380 RESTORE 5410
5390 FOR LP=0 TO 4 STEP
2:READ DAT$:INV$(LP*40+1
,LP*40+16)=DAT$:INV$(LP*
40+281,LP*40+296)=DAT$
5400 READ DAT$:INV$(LP*4
0+41,LP*40+56)=DAT$:INV$
(LP*40+241,LP*40+256)=DA
T$:NEXT LP
5410 DATA ┣┃┣┃┣┃┣┃┣┃┣┃┣┃
┣┃,┘┤┘┤┘┤┘┤┘┤┘┤┘┤,┐╱┐
┐╱┐╱┐╱┐╱┐╱,◥◤◥◤◥◤◥◤◥◤
◢◣◢◣,◢◣◢◣◢◣◢◣◢◣,
5420 POSITION 0,0:PRINT
#6;INV$(1,240)
5430 POSITION 0,0:PRINT
#6;INV$(241,480)
5440 GOTO 5420
```

You can go ahead and run the new program and watch the results. When you're done, here's the explanation of the changes:

3030; We've added two more string variables to this line. INV$ is going to hold the

Game Design Workshop

invaders screens, and we'll use DAT$ to help us set up INV$.

3160; We're using a full screen graphics mode one (GRAPHICS 1+16), and the graphics/lowercase section of our new character set.

5370; Okay, now we start setting up INV$. This line is a tricky way to set the entire string to hearts, which we've redefined to a space character. Incidentally, CTRL— will get you the heart character.

5380; In a long program where you'll be reading different data over and over again, it's a good idea to RESTORE the data first, just to make sure that you won't be reading the wrong stuff. That's why this line is here.

5390-5400; Now things start to get a little complicated, so let's take a good, close look at this loop. LP keeps track of the invader row that we're working on. We'll be doing two rows at a time, so we use STEP 2 (there are three types of invaders and two rows of each). The first thing we do is read a row of invaders into DAT$. Then DAT$ gets transferred to two places in INV$ (more about this in a minute). Finally, another row is read into DAT$ and then transferred to INV$, and the loop repeats.

The thing that really needs explaining is the process of transferring DAT$ to INV$. What exactly is going on? Let's look at the whole thing in English from start to finish. When matching the English explanation to the program lines, keep in mind that the first invader screen starts at INV$ (1,1) and the second at INV$ (240,240). You should also be aware that a line on the screen is twenty characters long, and we keep a blank line between each row of invaders.

Now that you're all geared up for the incredibly complicated explanation that you're sure is about to follow, relax. The first time through, the loop reads a row of invaders (version one of invader one) and puts it into the first row of the first screen and the second row of the second screen. Then it reads another row (version two of invader one) and puts it into the second row of the second screen and the first row of the second screen (by alternating versions like this, there will be more variety in the rows).

The next time through the loop, version one of invader two gets put into the third row of screen one and the fourth row of screen two. Then version two of invader two gets put into the fourth row of screen one and the third row of screen two. The third and last time through the loop, version one of invader three gets put into the fifth row of screen one and the sixth row of screen two. Version two of invader three then gets put into the sixth row of screen one and the fifth row of screen two. And that's all there is to it.

5410; Here's the data for the rows. How, you may be wondering, am I supposed to type this in? Either that, or you somehow managed to type it in already and are now extremely upset that I waited until now to tell you how. Sorry about that. Anyway, it's made up of eight CTRL-A/CTRL-Bs, eight CTRL-C/CTRL-Ds, eight CTRL-E/CTRL-Fs, eight CTRL-G/CTRL-Hs, eight CTRL-I/CTRLJs and eight CTRL-K/CTRL-Ls. If you type it in after running the original program, it will make more sense (since these graphics characters will look like the invaders).

5420; Now we're ready to get things going on the screen. We first put the cursor at the top left of the screen. Then we print the first 240 characters of INV$, which is the first invader screen.

5430; Now we do the same thing, except this time we print the second 240 characters of INV$ or the second invader screen.

5440; Finally, we make it into a loop so that the invaders will walk in place forever (or until you press the BREAK key).

So, what do you think so far? Not bad for BASIC? The problem is, we haven't really done anything yet. Keeping that in mind, what you see on the screen is actually extremely slow. By the time the rest of the program is in place, these moving invaders would be moving in extremely slow motion. But is there any way to speed things up?

Before I answer, think about the fact that what's slowing us down is the PRINT statement. What we need to do is find another way to get the invaders from INV$ to the screen. Perhaps I should restate that as "moving the invaders from the INV$ to the screen." Does that suggest a solution to you? If you thought of MOVMEM, give yourself ten points. Since we can find out the address of the screen using locations 88 and 89, there is absolutely no reason why we can't use MOVMEM instead of PRINT. How much of a speed difference will it make? Make the following changes and find out for yourself:

```
5180 MEM1=PEEK(88)+PEEK(
89)*256
5420 X=USR(ADR(MOVMEM$),
ADR(INV$),MEM1,239)
5430 X=USR(ADR(MOVMEM$),
ADR(INV$)+240,MEM1,239)
5440 GOTO 5420
```

Wow, quite a difference, eh? Here's a brief explanation:

5180; MEM1 is the address of screen memory.

5420; This is the equivalent of the old Line 5420.

5430; This is the equivalent of the old Line 5430.

5440; Do it again (and again and again and).

Once you get over the shock of the extra speed, notice the invader's color. It may not be obvious, but it is a different color than when we were PRINTing them. The reason for this is the fact that when we move characters directly to screen memory, we're dealing with the internal values rather than the ATASCII ones. In graphics mode zero, this means that we don't get the proper character. In graphics modes one and two, we either get the wrong character or a change in color. Change line 5410 to the following:

For some reason, scrolling has become the BASIC programmer's dream and nightmare. Despite the great effects it creates, it's not easy to get scrolling working from BASIC.

```
5410 DATA ├│├│├│├│├│├│├│
├│,┘┤┘┤┘┤┘┤┘┤┘┤┘┤┘┤,EFEF
EFEFEFEFEFEF,GHGHGHGHGHG
HGHGH,IJIJIJIJIJIJIJ,K
LKLKLKLKLKLKLKL
```

If you RUN the program with this change you'll notice that, despite the fact that we've changed some of the CTRL characters to letters, we still get the invader characters on the screen, only in different colors. To figure out what's going on, look up the internal values of the characters in Line 5410 and compare them to the internal values of the CTRL characters.

In graphics modes one and two, only the values between zero and 63 are used to determine the character. By adding 64, 128 or 192 to these values, you simply specify a different color for the character. You might like to play around with the characters in Line 5410 to see what I mean. When you're done, leave them as they are in the above line. It's nice for each type of invader to have a color of its own.

Before we go on to the next stage of our game program, a few words about your own games. Even though we're using different colored invaders in this game, each invader only has one color. What if you want to use more than one color in each character? Is it possible? Yes, and there are a number of different ways to go about it.

In graphics mode zero, there is something called "artifacting" that will allow you to have more than one color per character. In this mode, the dots in even-numbered columns have a different color than the dots in odd-numbered columns. Two dots side-by-side make a third color (white, usually). Along with the background color, this gives you a total of four colors instead of the usual two. The only problem is you have to be careful of where you place your dots. As an exercise, try redefining a couple of graphics mode zero characters so that one is all even-column dots, one is all odd and the other is a mix.

The other way to get multi-color characters is through the use of a special graphics mode called ANTIC mode four. In this mode, each dot can have one of four colors also, but you don't have to worry about odd and even. Also, the dots get their colors from the color registers, whereas the colors in artifacting come from a trick, and you can't change one of them without changing the rest.

To get four colors in ANTIC mode four, each dot is two bits wide instead of one. These two bits can hold a value between zero and three, which specifies the color register. So, when designing characters in this mode, you must keep in mind that each dot is two bits wide instead of one and has three ways of being "on" instead of one. Otherwise, the procedure for redefining characters is exactly the same. As far as getting to the mode in the first place, that has to do with Display Lists, which we'll be getting to in the future.

So much for that. Our next step in programming our game is to get the invaders moving back and forth across the screen. Unfortunately, here we run into a problem. The only way to move things using character graphics alone is to move them a whole character at a time. This means that characters will appear to jump across the screen instead of moving smoothly. That's not good. We want our invaders to move one tiny step at a time. So how do we solve this dilemma?

Actually, there is a way to do it with character graphics. If we design eight versions of each character, with each version being shifted to the right one dot, then we can get smooth movement by switching between these different versions. Well, this may be a good solution in some instances, but not in ours. Don't forget that we're using fourteen different characters for the invaders and the invader explosion. Eight versions of each would mean a total of 112 characters. Not only is that most of the character set, but it's also more than the 64 we're allowed in graphics mode one.

Besides, even if we could have that many, we would also have to have eight versions of INV$, which would take up almost 4K of memory alone. Nope, this technique is just not going to work for us here. Instead, we're going to use something that I'm sure you know about but haven't been able to use much before: fine scrolling.

Scrolling

For some reason, scrolling has become the BASIC programmer's dream and nightmare at the same time. Despite the great effect it creates in even the simplest programs, it seems that it's not too easy to get scrolling working from BASIC. Well, by the end of this discussion, fine scrolling will be just as easy as a simple POKE or two.

Instead of spending time looking at exactly what fine scrolling is and how it's done, let's jump right into an example. After we've got something up and running, we'll come back and take a look at the details. For now, delete lines 5420, 5430 and 5440 from the program we just created, and then add the following lines:

```
90 GOTO 3010
1000 X=USR(ADR(MOVMEM$),
ADR(INV$)+SB,MEM1,239)
1080 SB=240*(SB=0)
1280 IF COARSE=4 OR COAR
SE=-2 THEN CHANGE=-CHANG
E:POKE 1791,129-PEEK(179
1)
1330 SCROLL=SCROLL+CHANG
E
1340 IF SCROLL>15 THEN S
CROLL=SCROLL-16:COARSE=C
OARSE+2:POKE 1790,2:GOTO
 1360
1350 IF SCROLL<0 THEN SC
ROLL=SCROLL+16:COARSE=CO
ARSE-2:POKE 1790,2
1360 POKE 1788,SCROLL:PO
KE 1787,1
1380 IF PEEK(1790)<>0 TH
EN 1380
1390 GOTO 1000
3010 POKE 559,0
3040 DIM VBLOFF$(20):GOS
UB 29000:VBLOFF$=MLANG$
3160 FOR SEC=0 TO 1:GOSU
B 31000+10*SEC
3170 X=USR(ADR(MOVMEM$),
ADR(MLANG$),CA-256+90*SE
C,LEN(MLANG$)-1):NEXT SE
C
4000 GRAPHICS 17:POKE 55
9,0:POKE 756,CB+2
5010 DLIST=PEEK(560)+PEE
K(561)*256
5020 POKE DLIST+3,86
5030 FOR LINE=2 TO 12:PO
KE DLIST+4+LINE,22:NEXT
LINE
```

Game Design Workshop

```
5270 SCROLL=0:CHANGE=1:S
B=0:COARSE=0
5200 POKE 559,34
5290 POKE 54276,0
5360 POKE 1789,0:POKE 17
90,0:POKE 1791,128
5460 GOSUB 31500:X=USR(A
DR(MLANG$),CA-256)
5490 GOTO 1000
29000 MLANG$="h"d _)/ \d
"d b)\ \d♦":RETURN
31000 MLANG$="-♠/Ph-|/┤
T-K/ ┐T-</p0-0 ┐L-1 ┐M
1LIAp=I|p4)pI@♣0IP♣*H-▶/
0-1L-Xm</┌LH1Li♥┌L♣●1L8X
m</┌LH1L":RETURN
31010 MLANG$="i♥┌L♣ HHHP
=)♥</♠/L_d":RETURN
31500 MLANG$="hhXh(♦)/ \d
♦":RETURN
```

Before I get into an explanation of what we just did, we need to take a look at exactly what scrolling is in the first place. "Easy stuff," you say, "it's just moving things around on the screen." Yes it is, but it's the way things are moved that counts. As you probably know already, you can move characters around the screen simply by printing them in different positions. Unfortunately, the kind of movement that results isn't exactly the smoothest thing in the world, and it's also very slow. There's another way to get the screen to move without actually moving the objects on the screen. This method involves something called the "display list," which you may have heard of, and is the heart of fine scrolling.

The Atari computers do something that no other home computer that I know of allows you to do: put the screen data anywhere in memory. You can put the whole screen in one place, or break it up into two or more pieces and put each piece in a totally different place. You can even put two or more of these pieces in the same place, thus making things appear in two or more places on the screen (think that one over).

All of this is because of the display list, which we'll cover in excruciating detail in the next column. For now, suffice it to say that you can specify a different screen memory location for each line on the screen. In a regular graphics mode screen, a location is specified for the first line, and the rest of the screen is assumed to come right after the date for that line.

Okay, so we've now specified where the screen memory can be found. Let's suppose that the number "1234567890" is on the screen, starting at the top left-hand corner. What would happen if we now added one to the screen memory address? Screen memory would then start at the location that the "2" is stored in, right? And what effect would that have on the screen? The "2" will now be in the top left-hand corner, and the whole screen would have appeared to shift to the left by one character (the "1" will have disappeared off the left-hand side). Similarly, if we had subtracted one from the address, the screen would have shifted to the right by one.

This relatively simple concept is the whole basis of scrolling. Each time you want to scroll by one character, you just add or subtract one to the screen memory addresses in the display list (if you don't want a particular line to scroll, you just leave the screen memory address for that line alone).

What if you want to scroll up or down instead of left or right? Let's suppose you were using graphics mode one, where there are 20 characters on a line. Scrolling up by one character would then be the same as scrolling left 20 characters, and scrolling down one the same as scrolling right 20. Can you see why? Adding or subtracting 20 from the screen memory addresses moves screen memory to the next or previous line respectively, since the lines are stored one after the other in memory. It's as simple as that.

Fine scrolling

Who cares about scrolling by one character anyway? Wasn't the whole point of this column to learn how to scroll by less than a character, a task called "fine scrolling"? Yes, and fine scrolling is actually easier than scrolling by one character (called "coarse scrolling"). The problem is, you can only fine scroll by up to two characters (graphics mode one size). To do more than that, you have to combine fine scrolling with coarse scrolling. Before we get into that, however, let's look at how to fine scroll.

There are actually only two relatively simple steps to fine scrolling. The first is to decide which lines you want to fine scroll and then make some changes to the display list so that the computer knows which ones too. The second step is nothing more than a simple POKE to tell the computer how much to scroll. That's it.

But like I said before, the most you can scroll a line is two characters horizontally and two characters vertically. To get continuous fine scrolling, the trick is to fine scroll by two characters (although some prefer only to scroll by one), then coarse scroll by two and reset the fine scrolling at the same time. This has no visible effect on the screen (since the coarse scrolling moves moves the screen two characters in one direction, and resetting the fine scrolling moves it two characters in the opposite direction), but you are no longer at your fine scrolling limit and can now go ahead and fine scroll two more characters. By repeating this process, you can fine scroll forever if you want to.

Are you suffering from information overload? Perhaps it's time to go ahead and explain the above program lines, since they give an example of everything we just talked about.

90; We're going to start by going off to the end of the program and getting everything all set up. The stuff that actually gets done during the game will come earlier on in the program listing, because Atari BASIC is set up to do the things that come early faster.

1000; This is the routine to move the invaders from INV$ to the screen. Apart from moving it to here, we've also changed it so that the variable SB will determine the screen version that will be printed. SB is equal to zero for the first screen (ADR(INV$)+0=INV$(1,1)) and, for now, 240 for the second screen (ADR(INV$)+240=INV$(241,241)).

1080; After we print a particular screen, we want to change SB so that the other screen will be printed next time around. This line simply switches SB between zero and 240. (SB=0) is equal to one if SB=0 and 0 if SB<>0.

When you fine scroll horizontally, the computer makes each scrolling line longer than 20 characters per line.

1280; COARSE is a variable that we use to keep track of how many characters we've coarse scrolled by so far. We are only worrying about moving the invaders from side to side at this point, and we want to make sure that they don't go off the edge of the screen. So, we change the direction that they're scrolling in when COARSE gets too high (they've reached the right-hand limit) or too low (they've reached the left-hand limit). CHANGE tells what direction they're going in and is equal to one for right and minus one for left. Location 1791 is used in the machine-language scroll routine (called SCROLL) and also keeps track of direction. It is equal to 128 for right and one for left. We'll be going into more detail on exactly how SCROLL works later.

1330; The variable SCROLL keeps track of the amount of fine scrolling that has been done. I mentioned before that there is a memory location that takes care of fine scrolling. Actually, there are two. HSCROL at location 54276 takes care of horizontal scrolling and VSCROL at location 54277 takes care of vertical scrolling. As it turns out, our machine-language SCROLL will take care of both of these locations, but regardless of this, you can only POKE them, you can't PEEK (well you can, but you won't get the same values you POKEd). That means that you have to keep track of their values in your own variables, which is what the variable SCROLL does.

1340; When we have fine scrolled 16 times, we will have gone past the fine scrolling limit and must do a coarse scroll. In this line, we reset the variable SCROLL (which we will later use to reset the fine scrolling) and update COARSE. Location 1790 tells our machine-language routine that we want to coarse scroll by two characters in the direction specified by location 1791 (see line 1280). Having taken care of all of these, we then skip ahead to line 1360.

1350; If we're scrolling in the other direction, then we want to do the exact opposite when we've fine scrolled down to zero. We set the variable SCROLL to 15, update COARSE in the opposite direction, and again set location 1790.

1360; Location 1789 tells the machine-language routine the value to store in HSCROL, and if the value in location 1788 is not zero, then the routine knows that something needs to be done.

1380; Now we wait until the routine is finished.

1390; And then we go back to line 1000 to do the whole thing all over again with the next invader screen.

3010; We're going to be making some changes to the display list, so we turn off the screen temporarily to avoid a mess.

3040; VBLOFF$ will hold a machine-language routine that is used to turn off our VBLANK routines. SCROLL is a VBLANK routine, so we may as well set up and introduce VBLOFF now.

3160-3170; Here we set up SCROLL itself. The way that SCROLL works, it has to be stored in memory, not in a string. But it has been designed so that it can go anywhere in memory, so we'll put it right below the character set, since we've already made sure that part of memory is reserved.

4000; Now we actually set up our initial graphics mode and tell the computer where the character set is.

5010; We find out where the display list is.

5020; And change the first line (the one that also tells where screen memory is) to a fine scrolling line.

5030; Now we change the next 11 lines as well.

5200; Our changes are complete, so we turn the screen back on.

5270; Here we initialize our variables (their uses are explained in the earlier part of the listing).

5290; This just makes sure that the scrolling register is initially zero.

5360; Now we initialize the machine-language routine SCROLL.

5460; This turns on SCROLL. The CA-256 in the USR command is to tell where SCROLL is located in memory.

5490; We're all done setting things up, so go and start the actual movement routine.

29000; This is VBLOFF.

31000; This is SCROLL.

31500; This is SCRLON, the routine to turn on SCROLL.

If you're curious about SCROLL, the machine-language routine that we're using here, you may wish to skip ahead and take a look at the complete explanation of it. I'm leaving it until the end because at this point it should be fairly straightforward as to how it is used in our program, and there are more important problems that I'm sure you've already run across that I feel should be explained first.

Assuming you've made the previous additions to the program and have gotten it running, you're probably wondering what's going on. After all, the invaders are no longer neatly lined up on the screen any more, are they? Don't worry, you didn't do anything wrong. The problem comes from the fact that we are now fine scrolling.

When you fine scroll horizontally, the computer makes each scrolling line longer than, in the case of graphics mode one and two, 20 characters per line. To be exact, it makes these lines 24 characters long (48 in graphics mode zero). Why?

Let's suppose you scroll a normal width line two characters to the right. What is the computer supposed to put on the left edge of that line? The same is true for scrolling to the left. The computer has to have two extra characters on either side of each scrolling line in order to have something to scroll onto the left or right side of the line. Thus the extra four characters. When we set up INV$, we thought there would only be 20 characters on a line, and that's why the invaders are now spread all across the screen.

Our solution? We change the program so that INV$ is set up with 24 characters per line instead of 20. We'll do this by adding two spaces at the beginning and end of each line:

```
1000 X=USR(ADR(MOVMEM$),
ADR(INV$)+SB,MEM1,283)
1080 SB=288*(SB=0)
```

Game Design Workshop

```
3030 DIM MLANG$(90),INV$
(576),DAT$(16)
5370 INV$="♥":INV$(576)=
"♥":INV$(2)=INV$
5390 FOR LP=0 TO 4 STEP
2:READ DAT$:INV$(LP*48+3
,LP*48+18)=DAT$:INV$(LP*
48+339,LP*48+354)=DAT$
5400 READ DAT$:INV$(LP*4
8+51,LP*48+66)=DAT$:INV$
(LP*48+291,LP*48+306)=DA
T$:NEXT LP
```

Here's the explanation of these changes:

1000-1080; We've added four characters to each of the 12 lines that make up each screen, so we have to add 48 (4*12) to the values in these lines.

3030-5370; An additional 48 characters per screen, times two screens, means an additional 96 characters altogether for INV$.

5390-5400; What have we done here? We're multiplying LP by 48 now (24*2), have shifted the beginning position of each row of invaders by two to give us the extra two spaces at the beginning of each line, and the second screen now begins an extra 48 characters into the string.

How do things look now, a little better? As you can see though, there's still a problem. As the invaders scroll to the right, a pair of characters that don't belong appear at the top left-hand corner of the screen. Before I tell you why and how to correct it, I'd like you to think about it. You should be able to come up with the answer yourself, based on everything we've talked about so far.

Give up, or did you figure it out? Either way, the problem comes from the fact that the computer is trying to scroll on information that doesn't exist. When you first set up a graphics mode, the computer sets aside an area of memory for use as screen memory. The memory right before this screen memory usually holds the display list. Anyway, when you coarse scroll far enough so that screen memory now begins before the point it originally began at, you start getting strange stuff appearing on the screen.

That's our problem now. How do we get rid of it? We can do one of two things. If we change things so that the first line on the screen doesn't scroll, and start printing our invaders on the second line, we'll be okay. This way, we would be backing up screen memory into the first line, which we know is full of spaces.

The other way is to change the initial screen memory address so that it points ahead in memory. That way we know the memory before it (which used to be screen memory) is also full of spaces. Which of these methods is best? In this case I tend to favor the second, because it's just a little easier to do and also because it will come in handy later in the program. Also, with the first method you don't get to use the first line for scrolling. As it turns out, we won't want to anyway, but it is something to keep in mind. So, without any further ado, here are the changes necessary to move the screen memory forward:

```
5030 L=PEEK(DLIST+4)+44:
POKE DLIST+5,PEEK(DLIST+
5)+(L>255):POKE DLIST+4,
L-256*(L>255)
5040 FOR LINE=2 TO 12:PO
KE DLIST+4+LINE,22:NEXT
LINE
5180 MEM1=PEEK(DLIST+4)+
PEEK(DLIST+5)*256
```

And, of course, the explanation.

5030; Here we make the changes to the screen memory address, which is stored at the beginning of the display list. See next month's column to have this make more sense to you.

5040; This is just the old line 5030.

5180; We now find out where screen memory is from the display list, not the operating system (which is what locations 88 and 89 are for).

Ta-da! We're now all cleaned up and looking good. This is all we're going to do with the invaders for now. It will make life easier for us later when we're figuring out some of the logic. As a matter of fact, we won't program them to move down the screen until we're almost finished with the program. For now, it's important that the invaders just go back and forth forever so that we can keep them under control.

Even though we're done with the programming part of this column, there's still more do discuss. First of all, I should explain SCROLL, as promised. SCROLL is a VBLANK routine to do fine and coarse scrolling for you. VBLANK stands for Vertical BLANK, which happens 60 times a second and is the time during which the electron beam is on its way from the bottom of the screen back to the top (see the column on Video Magic).

During VBLANK, there are no changes being made to the television screen, so it is a good time to make changes to the display list and screen memory. In this case, we're making changes to the display list. If we did not make these changes during VBLANK, the screen would "jump" while you were making them. Anyway, this is all just to satisfy your possible curiosity; you don't have to worry about the details since I've already taken care of them for you. All that you have to deal with is how the SCROLL routine is used.

You've seen in the above program listings how to set SCROLL up and turn it on. Once you turn it on, it will just sit and wait for you to tell it what to do. And how do you tell it what to do? By setting two or more of five memory locations. Here are those locations and their meanings:

1787; This location tells SCROLL when you want to do something. You set it to one after you've set the next four locations (emphasis on the "after").

1788; This location is used to set the horizontal fine scroll register. Just store the value you want in the register here (and then set location 1787 to one).

1789; This is the same as 1788, except for the vertical fine scrolling register.

1790; This location is used to specify the number of bytes you want to coarse scroll by. Set it when you're ready to coarse scroli.

1791; Finally, this location is used to tell SCROLL what direction to coarse scroll in. Set it to one for left and up, and 128 for right

Game Design Workshop

and down.

Every VBLANK, SCROLL will check location 1787 to see if it's set to one. If it isn't, then that's all SCROLL does. If it is, then it takes the values in locations 1788 and 1789 and stores them in the fine scroll registers. Then it checks to see if location 1790 is equal to zero, in which case it sets location 1787 back to zero and stops. If it isn't set to zero, then that means you want to coarse scroll, so it goes through the display list and updates all the fine scrolling screen memory addresses. Then it sets locations 1787 and 1790 back to zero and waits for your next command.

Let me answer a question that you may have: Why can't the fine scroll registers be changed directly? Why does SCROLL have to do it? Well, it doesn't. Try making this temporary change to our program:

```
    1355 POKE 54276,SCROLL
```

Now you're changing the horizontal fine scroll register directly. Do you notice the occasional flicker on the screen? That's why we use SCROLL for this.

screen. The solution is also similar to that for horizontal scrolling, and it is to add an extra line at the bottom of the screen, one that does not scroll.

This line then tends to act as a buffer and gives the screen a place to get the extra information from. So when you're setting up your display list for a screen that is to be fine scrolled vertically, remember to add an extra line at the bottom of the screen, one that isn't set up for scrolling.

The last thing we're going to cover here is how to set up screens for games like *Scramble* and *Eastern Front*, where they are much larger than the display screen. For example, let's suppose that you want to design a game in graphics mode one where the entire game screen is 40 characters wide and 48 characters high (two display screens wide by two high). First of all, this means that you will need four times as much screen memory as a regular graphics mode one screen. How do you get this memory? One way is to just reserve it at the top of memory, just like you reserve space for a character set. There is, however, an easier way, at least in this case.

Table 6 MODE	0	1	2	3	4	5	6	7	8	9-11
BYTES/ROW	40	20	20	10	10	20	20	40	40	40
NO. OF ROWS	24	24	12	24	48	48	96	96	192	192
TOTAL SCREEN BYTES	960	480	240	240	480	960	1920	3840	7680	7680
TOTAL MODE BYTES										
(NORMAL SCREEN)	992	672	420	432	696	1176	2184	4200	8138	8138
(SPLIT SCREEN)	—	674	424	434	694	1174	2174	4190	8112	

It's now time to take care of a few details about scrolling that BASIC invaders doesn't really get into. First of all, our demonstration game does not use vertical fine scrolling, and chances are that may not be the case with a game of your own. We've already seen that vertical scrolling is basically the same as horizontal, with the exception that we have to coarse scroll by a whole line instead of one character.

But there is also another difference between the two. You recall that we ran into the problem of needing a wider screen for our horizontal scrolling. The same type of problem exists with vertical scrolling, except it is only a problem at the bottom of the

In Table 6 you'll find a chart that lists a whole bunch of information about the various graphics modes. One of these pieces of information is the amount of screen memory that the graphics modes use, and you'll see that graphics mode one uses 480 bytes (20*24). We need enough memory to store 48 lines of 40 characters each, or a total of 1920 bytes.

Looking at the chart again, we see that graphics mode six happens to use 1920 bytes for screen memory (usually you won't get an exact match, so you'd pick whatever mode came closest without going under). So if we set up a graphics mode six screen, we'll have the right number of bytes already set up for us. Of course we will have to change the display list but, as you'll see in next month's column, that's a piece of cake.

Once screen memory has been set up, the next step is set up the display list so that every line (except the last one, remember) is set to scroll vertically and horizontally, and also specifies a section of screen memory. Each of these screen memory addresses will be 48 bytes past the previous one, since that's how long our new lines are.

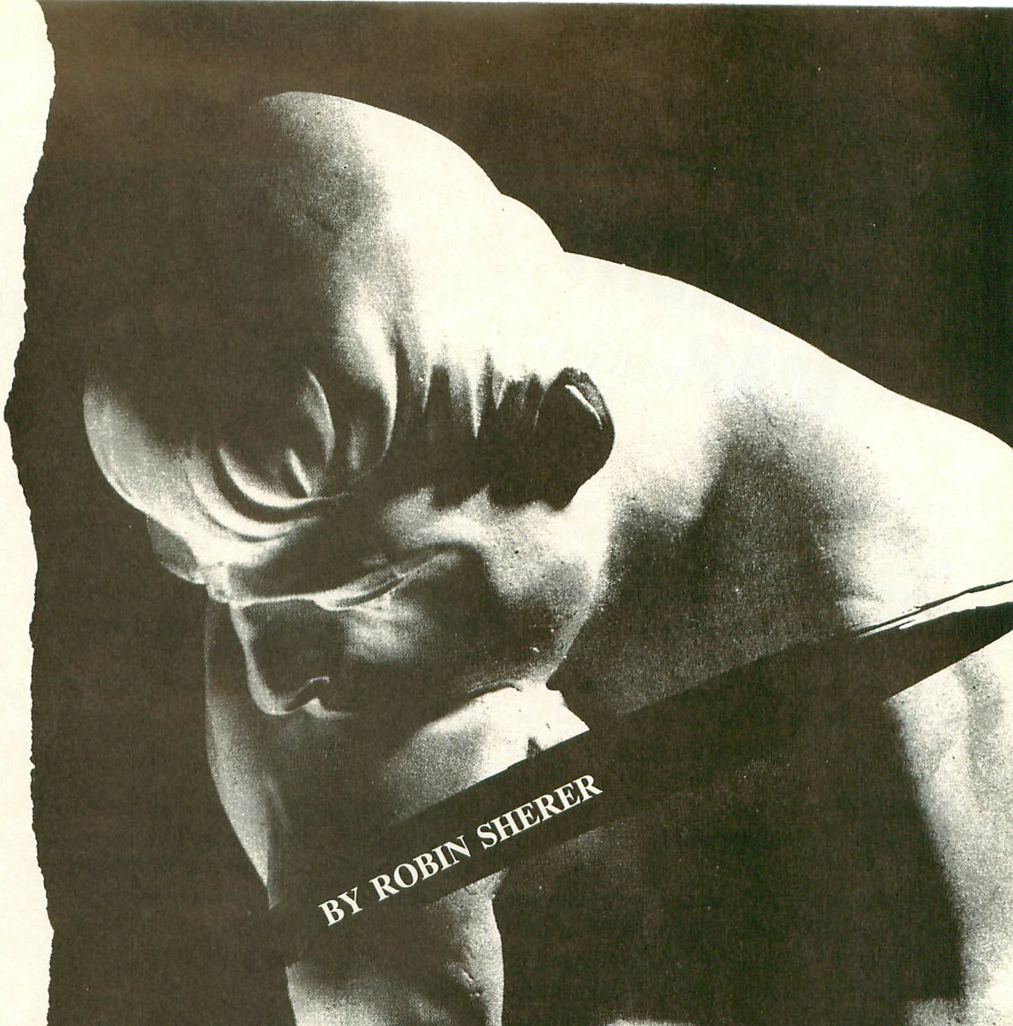
The final step before we actually begin to scroll is to set up the screen data in screen memory. This is just a matter of figuring out how you want the screen to look on paper (or design it a display screen at a time on the computer). From there, transfer the data into a string (remembering that the first 48 bytes of the string will be the first line of the screen), and then use MOVMEM to move the string data into the screen memory.

With all this taken care of, you're now ready to scroll around your new giant screen. This is the easy part, since SCROLL takes care of all the work for you. All you have to do is tell SCROLL when to scroll, and also keep track of how far you've scrolled, both vertically and horizontally, so you don't run off the edge of your screen. You can do this very simply by keeping two variables, say VCOARSE and HCOARSE. At any given time, these variables should show how many bytes you've scrolled down and how many you've scrolled right, respectively (you should update them every time you coarse scroll, just like we updated COARSE in our previous program).

In the case of our example, you don't want VCOARSE to get greater than 24, or HCOARSE to get greater than 16. Why not 48 and 40? When VCOARSE gets to 24, 24 lines will have already scrolled off the screen and there will be 24 lines on the screen, for our total of 48. Similarly, when HCOARSE gets to 16, 16 lines will have scrolled off the screen, and 24 will be on, for our total of 40.

Well, that about does it for fine scrolling. Of course, I still haven't explained the display list. We'll cover that topic next month.

MASTER MEMORY MAP PART IV



How to read the Memory Map

Beginning users: Read the text that is printed in bold type only. These memory locations will be the easiest for you to use and usually don't involve assembly language.

Advanced users: Read everything! Many areas of memory are not of any practical use, but you can learn a lot about how a computer works by reading the boring parts.

RUNSTK
142,143 008E,008F

This one is a pointer to the runtime stack. What is a "runtime stack"? Let's start off with a quick explanation of a stack.

Ever seen a stack of trays in a cafeteria? Customers take trays off the top; cafeteria people put trays on the top. If you're not lucky, there'll be a mad rush of people, and by the time you get to the stack there will be none left, and the cafeteria people will be nowhere to be seen. Well, a computer stack is the same thing, except it uses memory locations instead of trays, and there are no cafeteria people. A special memory location is used to point to the current top of the stack.

Now you know what a stack is, so let's talk about the runtime stack. *Runtime* just means that it's used while the program is running. When you use a GOSUB or a FOR/NEXT loop, BASIC has to be able to remember certain things, so it puts them on the stack until it needs to refresh its memory. Now you need to know what exactly gets put on the stack.

For each GOSUB encountered, four bytes are put on the stack (they are taken off when BASIC RETURNs from the subroutine). The first byte is a zero and tells BASIC that this is a GOSUB. The second and third give the line number that the GOSUB was on, and the last one is an offset into the line so that BASIC knows where to continue from after the RETURN.

FOR/NEXT loops are a little more complicated; they require 16 bytes to be put on the stack. The first six bytes give the number (in BCD) that the counter in the loop can go up to. The second six give the STEP value (also in BCD). The 13th byte gives the variable number plus 128 of the counter variable. The next two give the line number that the FOR statement was on, and the last one gives the offset within that line of the FOR. These values remain on the stack until the FOR/NEXT loop is complete.

There is one exception to the preceding two paragraphs. A BASIC POP statement will take the top entry off the stack, be it a GOSUB for a FOR/NEXT. You should

MASTER MEMORY MAP PART IV

make sure you POP the stack if you have to leave a FOR/NEXT loop before it's finished or a GOSUB before the RETURN.

Don't forget that the stack is constantly changing, so its size will vary.

Lastly, since the beginning of the runtime stack is also the end of the string/array area, BASIC also calls it ENDSTAR. Okay?

MEMTOP
144,145 0090,0091

Two uses for this one. First, relevant to the last location, MEMTOP is also called TOPSTK and points to the end of the runtime stack. Since the runtime stack is the last section of memory used by your BASIC program, MEMTOP is a pointer to the end of your BASIC program (which makes sense, right?). The memory locations from the address in MEMTOP plus one, all the way up to the display list (see SDLSTL [560,561]), are free for your use (but don't forget that the value in MEMTOP will change during program execution, since the runtime stack will be growing and shrinking).

For those of you who are still alert, don't confuse this MEMTOP with the MEMTOP at 741 and 742. This is the BASIC MEMTOP; the other is the OS MEMTOP.

The BASIC cartridge uses locations 146 to 202 for various uses, not all of which are worthwhile reporting on—with the following exceptions, of course:

FORLN
160,161 00A0,00A1

FORLN holds the line number of the current FOR statement encountered. For example:

```
100 FOR X=1 TO 25
110 NEXT X
120 PRINT PEEK(160)+PEEK(161)*256
```

LSTPNT
173,174 00AD,00AE

List pointer. Contains the location of the line being LISTed. When you just type LIST, you find 32767 here.

DATLN
182 00B6

Points to the number of the item within the DATA statement. This means we are currently reading the first number, the second, etc. Try this program:

```
10 FOR I=1 TO 8
20 READ A
30 ? PEEK(182)
40 NEXT I
50 DATA 1,2,3,4,5,6,7
```

DATALN
183,184 00B7,00B8

DATALN holds the line number of the DATA statement that was last READ. For example:

```
100 READ A
110 PRINT PEEK(183)+PEEK(184)*256
1000 DATA 10
```

You can use DATALN in an error-trapping routine to find out where a READ error occurred.

STOPLN
186,187 00BA,00BB

STOPLN holds the line number that the program was on when the program stopped, the BREAK key was pressed or an error was trapped. It is also useful in error-trapping routines. Now for our example:

```
100 TRAP 30000
110 NEXT Y
30000 PRINT PEEK(186)+PEEK(187)*256
```

ERRSAV
195 00C3

This location holds the number of the error that was trapped or caused the program to stop.

PTABW
201 00C9

When you print a whole bunch of items and separate them by commas in the PRINT statement (like PRINT A,B,C$), they get printed on the screen with a bunch of spaces in between them, right? Well, PTABW tells how many spaces to separate them by. In technical terms, that means it tells how many spaces there are between each tab stop on the screen (see TABMAP [675 to 689] if you want to set tabs for the TAB key). It can be set to any value from three to 255 but is initialized to ten. Let's look at an example:

```
100 PRINT 1,2,3
110 POKE 201,5
120 PRINT 1,2,3
```

SYSTEM RESET doesn't restore PTABW to its original value; GRAPHICS doesn't; nothing does. This is a very durable location.

Pokeing a zero here will cause the Atari to lock up shop when it encounters a comma in a PRINT statement.

BININT
202 00CA

If you put anything other than a zero here, then going into the immediate mode (i.e., SYSTEM RESET, BREAK or the program ending) causes the program currently in memory to erase itself—yet another fun way to prevent people from looking at your program (I personally like this one; it's devious).

Noname

A floating point register is just a place used to hold floating point numbers while operations are performed on them.

203-209 00CB-00D1

These locations are free, free, free for your use if you're programming in BASIC. If you're using a different language, check the accompanying documentation to find out which page zero locations *it* leaves free.

Noname
210,211 00D2,00D3

These two locations are reserved for BASIC, which means they have no specific use but you should stay away from them.

The floating point package

The remaining page zero locations from 212 to 255 are used by the OS's floating point package, a whole bunch of subroutines that BASIC uses when doing math and that kind of stuff. The routines themselves are stored in the OS ROM, so if you don't use them at all in your program, these locations will be free. Don't count on it though, even if you think you're not using the routines. They can sneak up on you when you least expect it.

Floating point math uses six-byte BCD, which was explained briefly under location VVTP (134,135). See the section in "De Re Atari" on the floating point package for more information.

Unfortunately, the listing for the floating point package is mighty hard to come by, so some of these locations are going to have real short explanations. My apologies to you, and my thanks to the *OS Manual* and *Mapping the Atari* for the information I couldn't find anywhere else.

FR0
212-217 00D4-00D9

Floating point register zero. A floating point register is just a place used to hold floating point numbers while operations are performed on them (it may also hold a partial result of an operation). They are all, including FR0 of course, six bytes long, since they must hold a six-byte BCD representation of the number.

FR0 is also used by the USR command. Remember that USR has the format X=USR (address [,argument][,...]) where X can be any variable and the arguments are optional. If you want your machine-language routine to give a value to X, you should store that value in the first two bytes of FR0 (212,213 —low byte and high byte respectively) before your RTS statement. BASIC will automatically convert these bytes into a floating point number and store it in X (or whatever variable you used for the call). If you're not using BASIC, you can use FRO yourself to convert binary values to floating point and vice versa. Put the binary number in locations $D4 and $D5 and then JSR $D9AA to convert to floating point (the result will be stored in FR0). To convert back, JSR $D9D2. Note that you can't use these routines from BASIC since BASIC is constantly using FR0 and will mess up your values.

FRE
218-223 00DA-00DF

This isn't very well documented, but it appears to be an extra floating point register.

FR1
224-229 00E0-00E5

Floating point Register 1. FR1 has the same format as FR0 and is often used in conjunction with it, especially for two-number arithmetic.

FR2
230-235 00E6-00EB

Floating point Register 2.

FRX
236 00EC

A single-byte register used for single-byte calculations.

EEXP
237 00ED

The value of the exponent (E). I suspect this is the exponent of the floating point number currently being processed, but this is only a suspicion.

NSIGN
238 00EE

The sign of the floating point number (same suspicion as above).

ESIGN
239 00EF

The sign of the exponent in EEXP (237).

FCHRFL
240 00F0

The first character flag. Your guess is as good as mine.

DIGRT
241 00F1

The number of digits to the right of the decimal point (zero to eight).

CIX
242 00F2

An offset into the text buffer pointed to by INBUFF.

INBUFF
243,244 00F3,00F4

Finally something that can be understood! There are times when BASIC has to convert an ATASCII representation of a number to the corresponding floating point value (like when you type in X=1000). INBUFF points to a buffer used to hold the ATASCII representation. The result gets stored in FR0. See LBUFF (1408 to 1535) for the buffer itself.

MASTER MEMORY MAP PART IV

ZTEMP1
245,246 00F5,00F6

A temporary register.

ZTEMP4
247,248 00F7,00F8

Another temporary register.

ZTEMP3
249,250 00F9,00FA

Still another temporary register (will it never end?).

RADFLG
251 00FB

RADFLG determines whether the trigonometric functions (SIN, COS, etc.) are performed in radians or degrees. If it's zero, then radians are used. If it's six, then degrees are in fashion. SYSTEM RESET and NEW both restore RADFLG to radians (zero).

BASIC also calls this location DEGFLG.

FLPTR
252,253 00FC,00FD

FLPTR holds the address of the floating point number that the package is now operating on. FLPTR and FPTR2 (to follow) point to the addresses where the results of the current operation will be stored. The documentation is sketchy though, so I'm just making an educated guess.

FPTR2
254,255 00FE,00FF

FPTR2 holds the address of the second floating point number that the package is operating on.

Page one

Locations 256 to 511 are called page one and have a very important use. They make up *the* stack for the OS, BASIC and DOS (see RUNSTK at locations 142 and 143 for an explanation of what a stack is). On powerup (and on SYSTEM RESET), the stack pointer is set to 511. Each time a machine-language JSR or PHA (PusH Accumulator on stack) instruction is executed, data is put on the stack and the pointer moved downward accordingly. When an RTS or PLA (PuLl Accumulator from stack) is executed, the corresponding data is pulled off the stack and the pointer moved back up. Since the stack pointer (which is a special location built into the main part of the computer) is just one byte, if you try and move it below location 256, it will wrap back around to Location 511 and vice versa.

Pages two through four

Locations 512 to 1151, as you will see, are used by the OS as a workspace. Some are used for variables, some for tables, some for vectors, some for buffers and some just for miscellaneous stuff. Now, a few words on using these locations. *Don't,* unless the description says you can! A lot of them are very important to the OS, and if you mess with them, they may cause the computer to crash, which you don't want to happen. Keep in mind, though, that no matter what you do, you can't hurt the computer (unless you throw it at a wall in frustration). You'll just hurt your program.

Also, be careful of locations that don't appear to be used. Atari has warned that these locations may be used in future versions of the OS, so stay away if you want to make sure your programs will work on all machines.

Let's jump right into page two. The first 42 bytes are used for interrupt vectors, so we'd better take a quick look at interrupts. As you remember, we first saw interrupts at location POKMSK (16). If you don't remember, go back and reread that section. I'll wait for you here. . . .

Back again? Okay, so now we have the basic idea of what an interrupt is. The *type* of interrupt we saw at POKMSK is called an Interrupt ReQuest (IRQ). There's another kind of interrupt called a Non-Maskable Interrupt (NMI). What's the difference? Well, there's an assembly-language command called SEI (SEt Interrupt disable). It tells the 6502 (the *main* chip) to ignore IRQ-type interrupts. Unfortunately, it can't tell the 6502 to ignore the NMIs. They are taken care of by another chip, called ANTIC, and so ANTIC is where you must go if you want to ignore NMIs.

The NMIs consist of the Vertical Blank Interrupt (VBI), the Display List Interrupt (DLI) and the SYSTEM RESET interrupt. We'll be seeing the interrupt vectors for both IRQs and NMIs in the next few locations, along with how to use them. An interrupt vector tells the OS where to go when the corresponding interrupt occurs (assuming it hasn't been disabled).

You might also want to look at IRQEN (53774), NMIEN (54286) and NMIST (54287) for more information on interrupts.

VDSLST
512,513 0200,0201

This is the vector for the Display List Interrupt (DLI) ,which is an NMI, as we discussed in the last location. DLIs interrupt the screen drawing process so you can do things like change the screen color halfway down. They exist entirely for your benefit; the OS doesn't use them at all.

To get a DLI going, there are a couple of things you have to do. First, and most important, you have to decide what you want the interrupt to do! Write the routine to do it, making sure it ends with an RTI (ReTurn from Interrupt) instruction. Next, decide which row on the screen you want it to occur at (it will actually occur at the end of this row). Go into the display list and set the leftmost bit (bit seven) of the instruction for that row. That tells the display list that there is to be a DLI on this row. Now tell the OS where the DLI routine is by setting VDSLST (low byte and high byte of the routine address). Finally, you have to enable the DLIs. Do this by setting NMIEN (54286) to 192.

DLIs are powerful. They can be used to change colors, to change character sets, even to change player/missile positions and the fine scrolling registers.

Here's a quick example from BASIC, simply reversing the playfield colors halfway down the screen:

```
100 GRAPHICS 0
110 DLIST=PEEK(560)+PEEK(561)*256
120 POKE DLIST+16,130
130 FOR MEM=1536 TO 1553
140 READ INSTR
150 POKE MEM,INSTR
160 NEXT MEM
170 POKE 512,0:POKE 513,6:POKE 54286
,192
180 LIST
190 DATA 72,173,198,2,141,10,212,141
,23,208
200 DATA 173,197,2,141,24,208,104,64
```

Make sure that the DATA is correct before you run the program. If it isn't, the computer might lock up. Here's an assembly listing of what those DATA statements represent:

```
0600  48      PHA
0601  ADC602  LDA  COLOR2
0604  8D0AD4  STA  WSYNC
0607  8D17D0  STA  COLPF1
060A  ADC502  LDA  COLOR1
060D  8D18D0  STA  COLPF2
0610  68      PLA
0611  40      RTI
```

Now that you know the basics, let me tell you a few limitations. First of all, there is very little time available during a DLI before the next row starts to get drawn. Make your routine short. Second, because an interrupt often occurs while something else is going on (like your BASIC program running), you have to make sure that you restore the accumulator and the X and Y registers if you use them. Do this by pushing their values onto the stack before you use them and then pulling the values back off before you RTI. Finally, as should be painfully obvious to you BASIC programmers by now, this is most definitely machine-language country. It's not very difficult machine language, but it is machine language.

A few notes now for the machine-language programmers. Change the hardware registers, not the shadow registers. The shadow registers are used to update the hardware registers during VBLANK. Changing them halfway down the screen won't have any effect until VBLANK kicks in.

If you're going to have more than one DLI, then each DLI routine will have to reload VDSLST to point to the next one. The last one will have to point back to the first one. Make sure in this case that you enable the DLIs during VBLANK, or else they may not execute in the right order.

Use WSYNC (54282) if you're changing screen colors. When *any* value is stored in WSYNC, the next command won't be executed until the TV has finished drawing the current scan line. If you *don't* use it, your colors will change in the middle of a line and will flicker back and forth. Try it and see for yourself (get rid of "141,10,212" in Line 190 and change "1553" in Line 130 to "1550").

One other problem with DLIs is that pressing a key on the keyboard can cause DLI colors to "jump" down a scan line (try it). The solution? Well, the easiest is just not to use the keyboard. For more complex ways around it, you should consult "De Re Atari."

DLIs are *extremely* powerful. They can be used to change colors, to change character sets, even to change player/missile positions and the fine scrolling registers; so be creative. Proper use of DLIs can produce a program that will do things you never thought the Atari was capable of.

VPRCED
514,515 0202,0203

This one's an IRQ vector, for an interrupt called the "serial proceed line interrupt," where the word "serial" indicates I/O to a peripheral such as the disk drive. It is initialized to 59314, which just holds a PLA and an RTI (i.e., the interrupt is used).

VINTER
516,517 0204,0205

Another IRQ, this time for the "serial bus I/O interrupt." Initialized to 59314 again because it isn't normally used. Both VINTER and VPRCED's interrupts are processed by the PIA (Peripheral Interface Adapter) chip.

VBREAK
518,519 0206,0207

IRQ again, for the machine-language BRK command [which is *not* the same as the BREAK key; see POKMSK (16) and BRKKEY (17)]. It's also initialized to 59314.

VKEYBD
520,521 0208,0209

From now on, if I don't tell you what kind of interrupt it is, it's an IRQ, okay? There's a whole bunch of these suckers and only so many ways to say "here's another IRQ."

So here's another IRQ. This one occurs whenever a key other than BREAK is pressed (START, OPTION and SELECT don't count because they're buttons, not keys). It's initialized to 65470, which is the OS keyboard IRQ routine (it makes sure that only one character gets printed when you press a key, and resets ATRACT [77]). If you want to put your own routine in, this is the place to do it. Keep in mind, however, that your routine will be executed *before* the key code gets converted to ATASCII (see the OS manual for a list of key codes).

The following three vectors are used to control communication between the serial bus and the serial bus devices (serial refers to the fact that bits are sent or received one after the other in succession). A much simplified explanation of this process follows. You should consult "De Re Atari" if you need more details.

The data being sent or received is stored

MASTER MEMORY MAP PART IV

in a buffer. If we're doing output, then a byte gets transferred from the buffer over to the serial output register (an interrupt routine does this). SIO takes it from there and puts it in POKEY's serial output shift register. POKEY then picks it up and sends it out one bit at a time. An interrupt is then generated, and the whole process starts over. This goes on until the checksum byte has been sent, at which time a "transmit done" interrupt is generated and SIO hands control back to the main program, which has been waiting patiently all this time.

The process is pretty much the same if we're receiving data, except in reverse.

VSERIN
522,523 020A,020B

This is a good one. The "POKEY serial I/O bus receive data ready" interrupt vector. It means that this vector is used when the I/O bus indicates that it has received a byte that is now waiting in the serial input register, ready to be moved to a buffer. The routine in the OS to do this is at 60177, and that's what VSERIN is initialized to.

VSERIN is also called INTRVEC by DOS, which changes its value to 6691, a routine in DOS that does pretty much the same thing as the one in the OS, except in a different place.

VSEROR
524,525 020C,020D

The opposite of VSERIN, VSEROR is used when the I/O bus is ready to send a byte. Its official name is the "POKEY serial I/O bus transmit data ready" interrupt vector, which should make more sense this time. It is initialized to 60048, the address of an OS routine that, logically, moves the next byte in the buffer to the serial output register (from whence it gets sent). DOS messes with this one too, changing it to 6691, the address of *its* routine to do the same thing.

VSEROC
526,527 020E,020F

Another long-winded name: the "POKEY serial I/O bus transmit complete" interrupt vector. Since I'm sure you're all becoming experts at interpreting these names, it should come as no surprise that this vector is used when all the data has been sent. It is initialized to 60113, a routine that, when the checksum byte is sent (see CHKSUM [49]), sets the "transmission" done flag at XMTDON (58) and disables this kind of interrupt.

The following three locations are the interrupt vectors for the POKEY timers, all of which are initially unused and therefore set to the PLA/RTI combination at location 59314. The timer interrupt occurs when the associated timer counts down to zero.

For more information on the POKEY timers, see the section on timers right before location 53760.

VTIMR1
528,529 0210,0211

Interrupt vector for POKEY Timer 1 (see AUDF1 [53760,53761]).

VTIMR2
530,531 0212,0213

Interrupt vector for POKEY Timer 2 (see AUDF2 [53762,53763]).

VTIMR4
532,533 0214,0215

Interrupt vector for POKEY Timer 4 (see AUDF4 [53766,53767]). This vector only exists in the "B" version of the OS.

VIMIRQ
534,535 0216,0217

Every IRQ vectors through this location on its way to the individual interrupt routines. It is initialized to 59126, the address of an OS routine that looks at IRQST (53774) to determine what kind of interrupt occurred and then jumps through the appropriate vector.

Attention B OS owners!

Since a lot of addresses in the new "B" version of the OS got shifted around, some of the initialization addresses given aren't the same in that version (which is now in a majority of the Ataris out there). Here are the changes (Figure 9).

Software timers

There are two types of timers in the Atari: software and hardware. We've already come across the hardware timers (see VTIMR1-4 [528-533]), and we're about to learn everything we never wanted to know about the software timers, which use Locations 536 to 558. But first, a few words from our author.

There are, of course, differences between software and hardware timers, and you'll probably want to know them before you go running off into timer land. The biggest difference comes from the names.

VECTOR	INITIAL VALUE
VDSLST	59280
VPRCED	59279
VINTER	59279
VBREAK	59279
VKEYBD	same as before
VSERIN	60175
VSEROR	same as before
VSEROC	60111
VTIMR1-4	59279
VIMIRQ	59142
VVBLKI	59310
VVBLKD	59653

Hardware timers are built into the POKEY chip; software timers are part of RAM. The big difference comes in the way they keep time. You recall from location RTCLOK (18-20) that a jiffy is 1/60 of a second, the amount of time it takes the television set to fill the screen. Well, the software timers count down by one every jiffy. The hardware timers, on the other hand, count down by an amount less than

A jiffy is $\frac{1}{60}$ of a second: that's the time it takes the TV set to fill the entire screen with a picture.

a jiffy, which *you* can specify (see Locations 53760 through 53769). So, if you want to time things that take longer than a jiffy, use the software timers. Otherwise, go for the hardware.

CDTMV1
536,537 0218,0219

This is the first software timer (affectionately known as "System Timer 1"). Every VBLANK, the value in CDTMV1 gets decremented by one. When it reaches zero, a flag gets set so the OS knows to JSR through CDTMA1 (550,551). An important thing to note here is that the decrementing for this timer (and *only* this timer) is done during Stage 1 VBLANK. This means that CDTMV1 (along with RTCLOK [18-20] and ATRACT [77]) is updated every VBLANK, no matter what's going on elsewhere in the computer. The rest of the software timers, on the other hand, are updated during Stage 2, which means that during time-critical I/O (like disk and cassette I/O; see CRITIC [66]), the other times are *not* updated. Unfortunately, the OS knows this too, so *it* uses CDTMV1 for I/O routines. So, you see, we have a catch-22 situation here. Oh, well! If you're doing your own time-critical routines though, you know which timer to use.

CDTMV2
538,539 021A,021B

This is System Timer 2, of course. When it reaches zero, it JSR's through CDTMA2 (552,553). And, unless you slept through the last paragraph, you should already know that it will *not* be updated during time-critical I/O.

CDTMV3
540,541 021C,021D

The third system timer, again hampered by time-critical I/O. This one has problems of its own through. First of all, the cassette handler uses it. Secondly, instead of JSRing through a vector when it gets down to zero, it just clears a flag at CDTMF3 (554). So don't use it during cassette operations and don't expect it to go anywhere after it's done.

CDTMV4
542,543 021E,021F

Let's see. You've already figured out that this is System Timer 4, that it doesn't work during time-critical I/O and you may have guessed that it clears a flag at CDTMF4 (556) when it's done instead of vectoring. What's left for me to say?

CDTMV5
544,545 0220,0221

The last of the timers. This one is no different than the last one except that the flag it clears is at CDTMF5 (558). But since you're getting to know these things so well, I shouldn't have to tell you that.

VVBLKI
546,547 0222,0223

Since this is the vector for the VBLANK Interrupt (VBI), I suppose this is probably a good time to explain exactly what vertical blank is. With all the previous mentions of jiffies in this book, you should know by now that a jiffy is $\frac{1}{60}$ of a second. It is important because that's the time it takes the television set to fill the whole screen with a picture. Since the screen can't hold on to that picture for very long, the TV keeps drawing the picture over and over again, even if it doesn't change. It draws it one line at a time, from top to bottom. When it gets to the bottom, it stops drawing and goes back to the top, where it starts all over again. Now, the important part for us is when it stops drawing. At that time it tells the computer, "Hey, I'm not drawing to the screen anymore," thus generating a vertical blank interrupt. You should be able to see where the name comes from now. Incidentally, there is also a horizontal blank, which occurs while the TV has finished drawing one line and is on its way to the beginning of the next. Store any value in WSYMC (54282) and the computer won't do anything until the next HBLANK occurs.

Back to VBLANK. There are a few reasons why the TV isn't drawing to the screen. First of all, it gives us a way to time things, since VBLANK occurs precisely every $\frac{1}{60}$ of a second. Secondly, nothing is being drawn to the screen during this time, so any graphics changes made during VBLANK will result in smooth, instantaneous changes on the screen. But, perhaps most importantly, VBI code runs independently of mainline code. What does that mean? It means that VBI code is essentially a separate program, running at the same time as your regular program! I wrote one VBI program, for example, that allowed the computer to play music at the same time I was typing in programs. Chris Crawford, in his classic *Eastern Front 1941* game, used VBI to separate the thinking process of the game from the tedious stuff like graphics and user input. That allowed the computer to think about its next move at the same time the player was thinking about his or hers, thus simulating a true one-on-one situation. As you can see, VBLANK is an extremely powerful tool.

Let's take a closer look at what normally goes on during VBI. First of all, there are two stages. The first stage is always executed, while the second gets ignored if the time-vertical I/O flag at CRITIC (66) is set. The first is called "immediate" vertical blank, the second is "deferred."

VVBLKI is the vector for the immediate stage, so the OS goes through VVBLKI when the VBLANK interrupt first occurs. During this stage the real-time clock (RTCLOK [18-20]), the attract mode (ATRACT [77], DRKMSK [78] and COLRSH [79]), and system timer one (CDTMV1 [536, 537]) get updated, processed and so forth. Then CRITIC is checked. If it's set, indicating that the interrupt occurred in the middle of a time-

MASTER MEMORY MAP PART IV

critical I/O operation, the OS returns from the interrupt. If it's not, then it's okay to go on to Stage 2, so we do. When the OS is done with Stage 2, it vectors through VVBLKD (548,549) to the user's deferred VBI routine, and then finally returns from the interrupt when it's done there.

VVBLKI is initialized to point to SYSVBV (58463), which contains a JMP instruction to the OS Stage 1 code (located at 59345 in the old OS, 59310 in the new one). If you change VVBLKI to point to your own routine, and you still want the OS code to be executed, you should end your routine with a JMP SYSVBV statement.

Whew, what a lot of mumbo jumbo! If you managed to plod through all of that, take a well-deserved rest. When you're done, we'll take a look at how you can use vertical blank for your own routines.

VVBLKD
548,549 0224,0225

Don't worry, there's still more to come on VBIs! This just seemed like a good time to formally introduce VVBLKD, the vector for the user's deferred VBI routine. The OS initializes VVBLKD to its "exit vertical blank" routine (at 59710 in the old OS, 59653 in the new one). If you use VVBLKD to point to your own routine, make sure to end that routine with a JMP XITVBL (XITVBL contains a JMP instruction to the exit vertical blank routine, which means you don't have to worry about which OS is being used since XITVBL is at 58466 in both). Note that you can also avoid the whole entire OS VBI code by writing your own immediate VBLANK routine and ending it with a JMP XITVBL instead of a JMP SYSVBV. Remember that none of the timers or color registers or anything will be updated if you do this (unless you update them in your routine).

By now you're probably either real excited over the prospect of using VBIs yourself, or you're asleep. If it's the latter, then you're not even reading this because your eyes are closed, so I'm only going to deal with those of you who are excited, okay? Let's look at how to write our own VBLANK routines.

The first step is to decide whether you want your routine to be immediate or deferred. Most of the time it doesn't matter. There are, however, the following conditions which will require one over the other.

1. If you want to change locations that the OS deferred routine also changes, you obviously want to do so *after* the OS does. Use deferred.

2. The maximum amount of time you can spend in immediate VBI is 2,000 machine cycles (see a book on 6502 assembly language for information on the number of machine cycles per instruction). If your routine is going to be long, you should therefore put it in deferred VBI, which has 20,000 cycles available. If you don't, things are going to look mighty funny on the screen. If you do use deferred, do your graphics first, since some of those 20,000 cycles occur while the screen is being drawn.

3. If you need your routine to be executed every VBLANK, regardless of whether time-critical I/O is occurring, use immediate. Be careful, however, that your routine will not cause problems with the I/O.

Now that you've decided what it should be (and you've presumably written it and put it in memory somewhere), all you need to do is change VVBLKI or VVBLKD to point to it. A simple task, right? Not quite. What happens if a VBI occurs while you're changing the vector? Crash city!

To make sure this doesn't happen, you have to change the vectors during VBLANK. But that itself presents a small problem. How do we get into VBLANK to change the vectors if we have to change the vectors to get to VBLANK (good old catch-22 again)? Luckily, Atari has thoughtfully provided a VBI routine that makes the change for you. It's called SETVBV and is at 58460. To use it, load the 6502 Y register (LDY) with the low byte of the address for your routine, and load the X register (LDX) with the high byte. Then load the accumulator (LDA) with a six if you want immediate VBI, seven if you want deferred, and JSR SETVBV. Now your VBI will be up and running.

Here's a simple example that uses location Chact (755) to make inverse text blink:

```
100 FOR MEM=1536 TO 1575
110 READ CODE
120 POKE MEM,CODE
130 NEXT MEM
140 X=USR(1536)
150 DATA 104,169,0,141,29,2,160,16,1
62,6,169,6,141,29,2,32
160 DATA 92,228,96,173,28,2,208,13,1
69,30,141,28,2,173
170 DATA 243,2,73,2,141,243,2,76,95,
228
```

Make sure that the DATA values are correct before you run the program. If they aren't, the computer will probably crash and you'll lose the program.

Here's the assembly-language listing of the machine code (which is stored in the DATA statements):

```
0600 68               PLA
0601 A900             LDA #$00
0603 8D1D02           STA CDTMV3+1
0606 A010             LDY #VBLANK&255
0608 A206             LDX #VBLANK/256
060A A906             LDA #$06
060C 8D1D02           STA CDTMV3
060F 205CE4           JSR SETVBV
0612 60               RTS
0613 AD2C02   VBLANK  LDA CDTMV3
0616 D00D             BNE VBLXIT
0618 A91E             LDA #$1E
061A 8D1E02           STA CDTMV3
061D ADF302           LDA CHACT
0620 4901             EOR #$02
0622 8DF302           STA CHACT
0625 4C5FE4   VBLXIT  JMP SYSVBV
```

The "LDA #$1E" in the preceding listing is used to specify a half-second interval ($1E hex equals 30 decimal equals 30 jiffies equals half a second) for use in blinking. Make it

What happens if a VBI occurs while you're changing the vector? Crash City!

larger or smaller to make the interval longer or shorter, respectively.

CDTMA1
550,551 0226,0227

CDTMA1 is the vector for System Timer 1 (CDTMV1 [536,537]). It's initialized to 60400, which is the address of a routine to set the time-out flag TIMFLG (791). This is because the OS uses CDTMV1 for I/O routines, which is a very good reason why you probably should use Timer 2 instead.

The OS vectors through CDTMA1 when CDTMV1 counts down to zero. If you *do* use CDTMV1 and are setting it for a value greater than 255 (i.e., setting both the low *and* high byte), this presents a potential problem. Since CDTMV1 is updated during VBLANK, and there is a chance that a VBLANK might occur while you're setting CDTMV1, you should set the low byte first. You can also use the SETVBV routine mentioned in the VBLANK description preceding. Just LDY with the low byte, LDX with the high, LDA with the timer number (1-5), and JSR SETVBV. This will assure that the timer gets set during VBLANK.

Since the OS JSRs through this vector, you should end your routine with an RTS instruction.

Incidentally, CDTMV1 reaching zero generates an NMI, which then does the vector.

CDTMA2
552,553 0228,0229

Same as CDTMA1, except this one is *not* used by the OS and is therefore initialized to zero. Oh, and of course CDTMV2 (538,539) reaching zero causes the vector through here, not CDTMV1. But then we already knew that, didn't we?

CDTMF3
554 022A

Unlike system Timers 1 and 2, Timers 3 through 5 merely clear a flag when they count down to zero. This is the flag for CDTMV3 (540,541) and is also used by DOS as a time-out flag, so beware of possible conflicts if you use it.

As with the other two flags, *you* must set CDTMF3 when you set CDTMV3. Any non-zero value is okay.

SRTIMR
555 022B

Well, here in the middle of all the timer stuff is a different kind of timer. As *everybody* knows, if you hold down a key on the Atari, it will start repeating, right? And something has to tell the OS how long to wait before starting that repeat and before repeating it again, right? And can you guess what location does that? Sure, I knew you could. SRTIMR is set to 48 every time a key is pressed. Every Stage 2 VBLANK that the key is still held down, SRTIMR gets decremented by one. When it reaches zero, the repeat process starts. It gets set to six, decremented again, the key repeats, it gets reset to six, and so forth until the key is released. Unfortunately, there are no locations that store the two delay times, so you can't speed up or slow down the process just by changing a couple of locations. There is, however, another way to do it.

As you recall, the initial delay time of 48 is set whenever a key is pressed. As you may or may not recall, we came across a vector a few locations ago (VKEYBD [520,521]) that pointed to the IRQ routine for a key being pressed. It is in this routine that the delay is set. So in order to change the delay, you must essentially take the OS routine, change the delay value, store your revised version in memory and update the vector. You'll find the OS routine at location $FFBE on page 130 of the OS listing.

How about the other delay, the six jiffy one, once the repeat is started? If you were paying attention (and *I know* you were), you already know that it gets set in Stage 2 VBLANK. Can you guess what you're going to have to do to be able to set it yourself? If you guessed "take the OS Stage 2 VBLANK interrupt routine and put it in my own deferred VBI routine with the delay value changed," then give yourself a pat on the back.

"But wait! The OS Stage 2 VBI routine gets executed whether I have my own deferred VBI routine or not," you say, taking me completely by surprise. You're right, though (or would have been if you had said it). Your deferred routine, however, happens *after* the OSs, so you can just repeat the part that sets the delay and, since you'll set it *after* the OS does, yours will be the one that counts. The part you want is at locations $E87C through $E897 on page 36 of the OS listing, and locations $E8E8 through $E8EE on page 37 (these locations will be different in the new OS, but that's irrelevant here). Be aware that the OS will now be executing this routine twice and will therefore be decrementing by *two* every VBLANK. You should set SRTIMR to *double* the delay you want, and also change your deferred routine so that it resets SRTIMR if it's equal to zero *or* one. That makes sure that the OS routine doesn't reset it before you get a chance to.

CDTMF4
556 022C

And now, back to our timers. This is the flag for CDTMV4 (542,543). See CDTMF3 for more information.

INTEMP
557 022D

INTEMP is used for temporary storage during the SETVBL routine. As you recall, SETVBL is at the address stored in 58460. Heaven only knows what INTEMP is doing here in the middle of the system timers.

CDTMF5
558 022E

This is the flag for CDTMV5 (558,559). See CDTMF4 for more information (ha, ha).

BITS 'N' PIECES: POPS

by Lee S. Brilliant

In the never-ending battle between AMIGA and ST partisans we hear rhetoric like:

"Well, *I've* got 4,000 colors to choose from!"

"Yeah, but how many can you put up at the same time?"

"But I've got a blitter chip!"

"So what, I will have one soon too (ha-ha!). Besides, my faster clock speed makes it unnecessary!"

"Okay. Top this: I've got stereo sound!"

Uh-oh. How do you respond to that one? You could mention the MIDI interface, but try this retort instead: "Atari gave up that one with the 8-bit machines because no one ever used the feature!"

"*Say what*? The old-fashioned, outdated, inferior, 8-bit Atari computers had four-channel stereo sound?"

"Yes! In fact, they had three-channel sound!"

"So, how come I never heard about it?"

"Because until this article, no one ever knew about it!"

So now you know the subject we will cover this month. Along the way we will rehash some old material about POKEY, cover some new stuff and build the POly Phonic Sound (POPS) device. So let's rehash!

POKEY revisited

In the last three installments of "Bits 'n' Pieces," we spent a long time discussing how POKEY performs serial output. Simply stated, POKEY contains a serial input and a serial output shift register whose baud rate is controlled by the sound frequency counters. This is similar in concept to a USART (Universal Synchronous/Asynchronous Receiver/Transmitter). Such devices form the heart of all serial I/O and modem devices. Certain aspects of the POKEY control registers were only glossed over before, and so now is the time to go into slightly greater detail. Let's look at POKEY register number 15 or SKSTAT/SKCTL ($D20F,53775). (See Figure 1.)

You see that bits 4, 5 and 6 control the USART parameters, determining what input and output modes will be used in various combinations of synchronous/asynchronous I/O. It is not a very straightforward proposition here, but I will try to simplify it. The key to understanding is to look to the clock lines on the serial bus. There are two: Clock Out (Pin 2) and Clock In (Pin 1).

Clock Out is relatively straightforward; whatever frequency is driving the Data-Out shift register is also present on the Clock-Out line. But the Clock-In line is bi-directional and may accept an external clock signal or act like Clock Out and transmit a POKEY-generated clock signal. This allows the input shift register or the output shift register to either accept an outside clock signal or use an internal clock.

Bits 4, 5 and 6 control the sources of clock pulses for both the In and Out shift registers and the directionality of the bi-directional clock, according to Figure 2. Since POKEY is the internal clock source, the frequencies generated by its dividers also appear on the clock lines.

Atari chose to use only asynchronous communication and clock the shift registers internally. The OS ignores any external clocks and does not support synchronous I/O even though the hardware can. You can actually disconnect the clock lines in your serial cable and still use your 1050 disk-drive and printer. I cannot say whether all peripherals will work this way but these two will. See the diagrams in Figure 3 for examples of how POKEY can be configured for synchronous or asynchronous communications.

BIT	FUNCTION
7	Forces Serial output to 0
6*	Serial port parameter selections
5*	" " " "
4*	" " " "
3	Changes serial out from 1/0 logic to two-tone
2	Changes from normal to fast POT scan
1	Activates keyboard scanning
0	Enables keyboard debounce circuits

FIGURE 1: SKCTL

The important thing to learn from Figure 2 is that when only bit 6 is set, POKEY Channel 4 will appear on the Clock-Out line. Here is the potential for four-voice sound in "stereo." Place an amplifier on the Clock-Out line and "play" one voice through the serial port and the others through the TV in "stereo." Moreover, you can have trinaural sound by playing Channels 2 and 4 on separate clock lines through separate amplifiers while playing 1 and 3 together on the regular audio output line via the TV (or the audio output at the 5-pin plug). You need only set bits 5 and 6 of SKCTL. This should make AMIGA fans stand up and take notice. The following information will allow you to produce two- or three-channel sound from your Atari.

BIT			BIDIRECT CLOCK	SERIN CLOCK	SEROUT CLOCK	
6	5	4	IN/OUT	SOURCE	SOURCE*	FUNCTION
0	0	0	IN	EXT	EXT	XMIT & RECV RATES SET BY EXTERNAL SOURCE AT CLOCK IN
0	0	1	IN	CH 4	EXT	XMIT CLOCKED BY EXT, RECV BY CH4. ASYNC RECV, SYNC XMIT
0	1	0	CH4 OUT	CH4	CH4	XMIT & RECV SET BY CH4. CH4 ON CLOCK IN
0	1	1	IN	CH4	CH4	XMIT & RECV SET BY CH4. **
1	0	0	IN	EXT	CH4	XMIT SET BY CH4 RECV BY EXT CLOCK
1	0	1	IN	CH4	CH4	XMIT & RECV SET BY CH4. **
1	1	0	CH4 OUT	CH4	CH2	RECV SET BY CH4. XMIT SET BY CH2. CH4 OUT ON CLOCK IN. ***
1	1	1	IN	CH4	CH2	RECV SET BY CH4. XMIT SET BY CH2. ***

NOTES:

EXT MEANS AN EXTERNAL CLOCK CONTROLS THESE SHIFT REGISTERS AND SETS THEIR BAUD RATES.
16-BIT RESOLUTION CAN BE USED FOR FINE TUNING.
* SAME SIGNAL ALWAYS APPEARS ON CLOCK OUT LINE.
** LABLED AS NOT USEFUL BY HARDWARE MANUAL.
*** TWO-TONE MODE CANNOT BE USED BECAUSE CHANNELS 1 & 2 ARE USED TO MAKE AUDIO TONES AND WILL CONFLICT.

FIGURE 2: POKEY USART Parameters

```
                              EXT
                  |----->------------------->--CLOCK OUT
                  |     DATA          DATA
                  ^      |             ^
                  |      v             |
CLOCK IN/OUT----->SERIN-->---SEROUT
  (EXT)                  ^             |
                         |             |
                         |             v
DATA IN-->-----------                  ------->----DATA OUT

        Synchronous, external clock both directions.
           Bits 4,5, & 6=0

                              POKEY
CLOCK IN/OUT----<-----CHANNEL 4-------->----CLOCK OUT
                               |
                         DATA  |      DATA
                         ^     v       |
                         |     |       v
                         SERIN<---->SEROUT
                         ^             |
                         |             |
                         |             v
DATA IN---->---------                  ------->----DATA OUT

        Asynchronous I/O
          Bit 5=1, Bits 4&6=0
```

FIGURE 3: Examples of USART configuration.

The POPS device

The POly Phonic Sound (POPS) adapter is really nothing more than three sets of resistors to reduce, balance and control volume for the three audio amplifiers; the rest is done with programming. The two serial port amplifiers incorporate electronic volume controls that are regulated through the joystick ports (PORTA) similar to the POKEY control registers.

Remember that these sound channels played on the clock lines are logic level signals or five volts. This level of audio will blow away most amps so you need the resistors to drop the voltage to a more reasonable 0.5 volts. The 4066 IC is a bank of electronic switches each of which is connected to a resistor, thus the higher the binary code applied, the more resistors are switched in and the more signal passes through. The desired volume value is placed in PORTA, which is configured for output through the joysticks and controls the volume of POPS left and center channels. Right channel volume and distortion are controlled as usual through AUDC registers.

I have demonstrated this system to users' groups and it really turns heads, even among people as jaded as Atari users. Electronic volume control allows fade-outs from all voices and can give rise to some interesting spatial sound effects. Listings 3 and 4 give simple demonstrations of the abilities of POPS. I have included a schematic and circuit board for POPS: note part values, size and polarity of capacitors and IC orientation.

Power is obtained from a nine-volt DC. 500-milliamp wall supply like the one used by Atari video games. They are available in abundance. You may even have one of these at home! Be sure to note the polarity of the power plug: + at the tip and — at the sleeve. POPS is compatible with all 8-bit Atari computers, but the 400 does not have an audio output line, so you can only play the right channel through your TV. Each channel of the amplifier puts out about half a watt, not enough to shatter glass but enough to get your wife upset! If you need more power you can get larger amplifiers and power supplies from Radio Shack.

Schematic

To produce stereo sound you must do the following things:

1) Plug POPS into the serial bus and the audio output, and then connect the amplifiers to three speakers and turn on the power. The right channel will come out the "normal" audio line so you can use your TV for one of the amplifiers on a 400.

2) Set AUDCTL for the proper values for the kind of sound you want to produce (ie., 16-bit or different master clock). Try 0 for starters (standard setting).

3) Set the bits of SCKTL; 67 (3+[bit 6]64) for two-channel. Without the +3, the keyboard won't work, and you will need RESET to recover from POKE 53775 64.

4) Set your AUDC and AUDF values for volume and distortion on all audio channels. Keep in mind that the signals on the clock lines are taken off before the control circuits, so the audio control registers have no effect. You cannot control the noise con-

tent of the audio, but for most music this is just fine. The actual value poked into AUDC2/4 is not important, except you want the volume component to be 0, so you won't hear these channels play on the audio line or TV: A 0 works great. You control volume by poking the volume level (0-15) into PORTA. Remember that the value to POKE into 54016 is the left volume +16*center value. The channels which come out the normal audio line do use the AUDC registers and can be used with noises for percussion sounds or special effects. You cannot use the BASIC SOUND statement because it resets the special values POKEd into AUDCTL. Build your POPS, connect it and try this simple program:

```
1 REM PLAYS 2 POKEY CHANNELS IN STER
EO THROUGH THE AUDIO CHANNEL (OR TV)
 AND SERIAL PORT PIN 2.
2 P=PEEK(54018):POKE 54018,P-4:POKE
54016,255:POKE 54018,P:REM INIT POR
TA
5 DELAY=100:POKE 53761,174: POKE 53
767,0:REM INITIALIZE AUDC1&4
10 POKE 53768,0:POKE 53775,67:REM I
NITIALIZE AUDCTL AND SKCTL FOR STER
EO
20 POKE 53760,60:REM PLAY A NOTE ON
 R CHANNEL
25 GOSUB DELAY:POKE 53760,0:REM TUR
N OFF
30 POKE 53766,91:PLAY A NOTE ON L C
HANNEL
35 GOSUB DELAY:POKE 53766,0
40 POKE 53760,72:POKE 53766,121:REM
 STEREO!
50 GOSUB DELAY:GOSUB DELAY:END
100 FOR S=1 TO 300:NEXT S:RETURN
```

When you POKE SKCTL with 67, you get Voice 4 from the left channel and 1, 2 and 3 from the right. You can combine values for 16-bit resolution with half on right and three-quarters on left. If you follow the same steps for two-channel sound, except you POKE SKCTL with 99 ([Bits 5+6=96]+3), you will get three-channel sound. Now Voice 2 will play through the left channel, Voice 4 through the center channel and Voices 1 and 3 on the right channel. You can combine 3 and 4 to good advantage here. Figure 4 is a block diagram of POKEY configuration with both two- and three-channel modes.

```
                             CH 1&2----------->---------AUDIO RIGHT
                                                       CH 3
                                                        |
                                                       CH 4
                                                        |
                          DATA              DATA        |
                           ^                  |         |
                           |                  v         v    AUDIO LEFT
   CLOCK IN/OUT--->SERIN                SEROUT<----->CLOCK OUT
                           ^                  |
                           |                  v
   DATA IN----->-------                  ----->----DATA OUT

                              STEREO



                  CH 3                  CH 1--->---AUDIO RIGHT
                   |
                  CH 4                       CH 2
                   |                          |
                   |    DATA          DATA    |
                   v     ^              |     v
   AUDIO CENTER    |     |              v     |      AUDIO LEFT
   CLOCK IN/OUT--->SERIN          SEROUT<------CLOCK OUT
                         ^              |
                         |              v
   DATA IN----->-------              ----->----DATA OUT

                          CHANNEL MODE
```

FIGURE 4: POKEY Multi-mode.

Putting it all together

To use the POPS device you need a good player program. I will make no bones about it, my programs are only modifications to Enhanced POKEY Player. I chose to modify this program because the player is in BASIC and machine language; so I could easily disassemble it and modify it. (Why reinvent the wheel?) Enhanced POKEY Player has been around a long time and was obtainable in the past from the ANTIC catalog (AP 0147), but has not been listed lately. You may have to ask for it specially. The following programs are published with the author's permission.

After you obtain your Enhanced POKEY Player, make a duplicate to work from and put the original away. *Never use the original disk with POPS!* You will also need a separate work disk. Listing 1 is the main program and consists of a BASIC routine to load the music files and display titles. Then there is the Player itself, which is contained in a large string array, PP$, and some fixed locations in page 6. One section of the player string is essentially blank which allows us to insert our different routines for mono, stereo or trinaural sound generation (CH1$, CH2 and CH3$).

Listing 2 creates these complex strings. Type in and save both Listings 1 and 2 on the separate work disk, and then run Listing 2 and save as LIST "D:TEMP," 2000,2300. Load back Listing 1 and ENTER "D:TEMP." Now resave to your working POKEY Player disk as "D:PLAYER." This new program replaces the original Player and will list all available music files and flag them as three-channel (*), two-channel (+) or one-channel. Music files on the original POKEY Player disk are all labeled *.V, but this new program requires files with extenders .V1C, .V2C, or .V3C to denote one-, two- or three-channel music files. You will need to work through DOS to change the extenders according to Table 1. The player is now able to tell which mode to use and automatically adjust itself according to the filename extenders.

FSONA	.V2C	MINUET2	.V2C	SLP3	.V3C
ANVIL	.V1C	GLAD	.V2C	MUSETTE	.V2C
SLP4	.V3C	BC5	.V2C	GMARCH	.V2C
NELLIE	.V1C	SLP5	.V3C	BLUES	.V2C
GOLDV1	.V2C	PRELUDE	.V3C	SLP6	.V2C
BMINOR	.V3C	GOTHOS	.V3C	PRESTO	.V2C
SOLDIER	.V3C	CALLIOPE	.V2C	GSONA	.V2C
RIGAUDON	.V3C	SWISS	.V3C	CANTATA	.V3C
HBDAY	.V2C	SABRE	.V2C	THEMEVAR	.V3C
CAPRICIO	.V3C	HOLST	.V1C	SCAR	.V3C
TPIAM	.V2C	CLAV	.V3C	HUMORESK	.V3C
SCHERZAD	.V3C	TPIBF	.V2C	COURANTE	.V2C
ITALIAN	.V2C	SCIPIO	.V3C	TPIF	.V2C
CSONATA	.V3C	LITTLE	.V3C	SLP1	.V3C
WILLTELL	.V3C	DMARCH	.V2C	LONDON	.V3C
SLP2	.V3C	WSOLDIER	.V2C	FEAST	.V3C
MINUET1	.V1C				

TABLE 1: Extender List

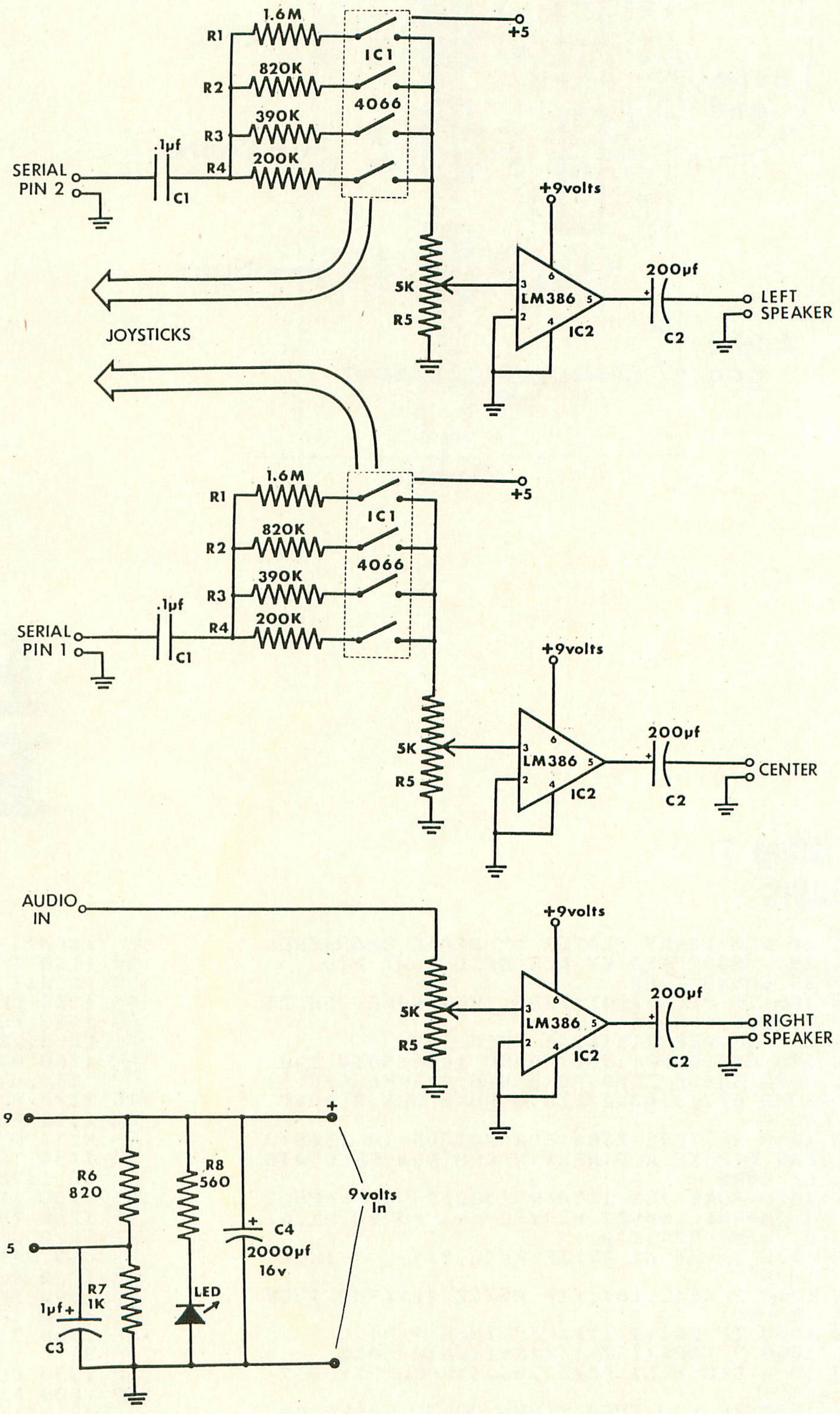
Notes about Enhanced POKEY Player

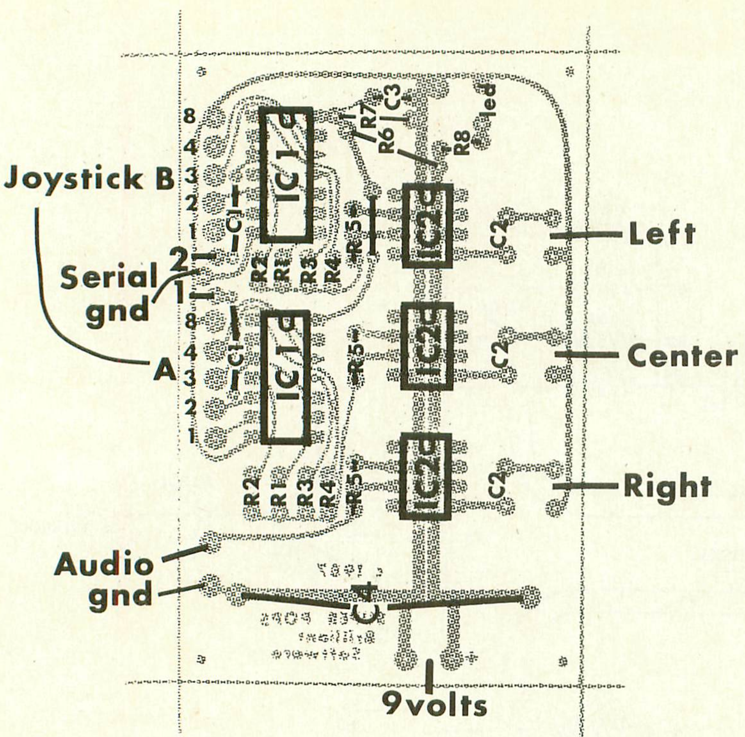
Writing music with the POKEY Player editor requires a couple of notes. POKEY Player has only three voices, naturally labeled 1, 2 and 3. POKEY Player Voice 1 uses POKEY chip's Voice 1 and will be the right channel in POPS. POKEY Player Voice 2 similarly is POKEY Voice 2 and is the left channel in three-channel mode, but in stereo it will come out the right channel along with Voice 1. POKEY Player Voice 3 is POKEY Voices 3 and 4 combined in 16-bit fashion and comes out from the center channel in three-channel mode and the left side in stereo.

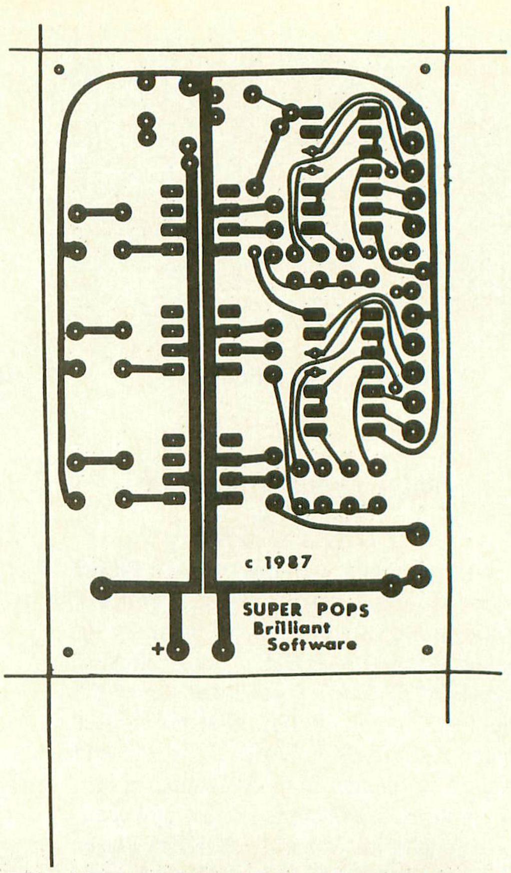
Only voices from the right channel use the distortion abilities of POKEY, so you can always use Voice 1 for percussion or special effects, but in stereo you can also use Voice 2. Voice 3 (3/4) usually carries the melody line because it has the widest range of note values and can also be used for really deep bass lines. Not all features of the POKEY Player editor can be used by POPS, and for some reason some three-voice music will not play properly on POPS' three-channel mode, but will do okay in stereo. Some two-voice programs can only be played in mono. The list of files on Enhanced POKEY Player are in Table 1 with their proper extenders.

Parts list

The 13-pin Atari Serial Plug (#83-360) is available from MCM Electronics, 2582 East River Road, Moraine, Ohio 45439; (800) 858-4330. The resistors, volume controls, capacitors, joystick cables and amplifiers are available at Radio Shack. Not all resistor values are available at Radio Shack, however. You may need to combine resistors such as 220K plus 470K to approximate 800K. Actual values are completely non-critical. The RCA phono jacks, speakers and cables are also available at Radio Shack. The 4066 ICs may be ordered specially through Radio Shack. Ask them to special order SK4066B. Finally, the power supply is Radio Shack's # 273-1455.







BITS 'N' PIECES: POPS

Listing 1: BASIC

```
SI 10 REM POKEY PLAYER BY CRAIG CHAMBERLA
   IN.  MODIFIED BY LEE BRILLIANT M.D.
MF 30 GOSUB 1000
FX 100 IF PEEK(764)<>255 THEN POKE 764,25
   5:POKE 1536,0
KL 110 IF PEEK(1536)=1 THEN 100
IV 120 U=USR(PP+156):GOSUB 1090:GOTO 100
NS 1000 GOSUB 2000:POKE 675,0:POKE 676,1:
   POKE 677,0:POKE 678,8:POKE 679,0:POKE
   65,0
HQ 1010 RESTORE 2300:FOR N=1606 TO 1648:R
   EAD D:POKE N,D:NEXT N:REM NON-RELOCATB
   LE CODE
CW 1020 POKE 752,1:TRAP 1900:? "K----MULT
   I CHANNEL MUSIC PLAYER----":OPEN #1,6,
   0,"D:*.V?C":K=0
TF 1030 INPUT #1,R$:IF R$(2,2)<>" " THEN
   1070
YS 1040 ? R$(3,10);:IF R$(12,12)="3" THEN
    ? "*";
KH 1050 IF R$(12,12)="2" THEN ? "+";
MS 1060 ? CHR$(127);:K=K+1:GOTO 1030
OJ 1070 CLOSE #1:POKE 703,4:POKE 752,0:?
   "K";
KG 1080 IF K=0 THEN ? "NO MUSIC FILES ON
   THIS DISK":FOR DE=1 TO 500:NEXT DE
TF 1090 TRAP 1950:? "KYOUR REQUEST";:INPU
   T R$:IF R$="" THEN POKE 703,24:GOTO 10
   20
CU 1100 F$="D:":F$(3)=R$:F$(LEN(F$)+1)="*
   .V?C":OPEN #1,6,0,F$
PF 1110 INPUT #1,R$:IF R$(12,12)="1" THEN
    PP$(181,254)=CH1$
YD 1120 IF R$(12,12)="2" THEN PP$(181,254
   )=CH2$
AR 1130 IF R$(12,12)="3" THEN PP$(181,254
   )=CH3$
SS 1140 CLOSE #1:OPEN #1,4,0,F$:A=BUFF
FG 1150 FOR K=0 TO 2:GET #1,LO:GET #1,HI:
   L=LO+256*HI
WY 1160 U=USR(CIO,A,L):IF U>127 THEN POKE
    195,U:CLOSE #1:GOTO 1910
TG 1170 U=USR(PP+94,K,A):A=A+L:NEXT K
BJ 1180 FOR N=1601 TO 1604:POKE N,0:NEXT
   N:? "K";
IZ 1190 TRAP 1200:INPUT #1,R$:? :? R$;:GO
   TO 1190
WT 1200 IF PEEK(195)<>136 THEN 1950
TK 1210 TRAP 1950:CLOSE #1
CZ 1220 U=USR(PP,PP+180,PP+267,PP+473,PP+
   819,PP+625):POKE 1536,1
AM 1230 RETURN
CJ 1900 IF PEEK(195)=170 THEN ? "I DON'T
   KNOW THAT SONG.":CLOSE #1:GOTO 1090
QW 1950 ? "KERROR ";PEEK(195):A=USR(PP+15
   6)
NN 1999 CLR :END
QP 2000 DIM PP$(892),CH1$(74),CH2$(74),CH
   3$(74),CIO$(34),R$(40),F$(16),BUFF$(FR
   E(0)-500)
NS 2005 PP=ADR(PP$):BUFF=ADR(BUFF$):CIO=A
   DR(CIO$)
OS 2240 PP$(876,876)=CHR$(155):RETURN
MI 2300 DATA 12,24,36,48,244,232,220,208,
   1,2,3,4,5,6,7,0,255,254,253,252,251,25
   0,249,108,2,6,108,4,6,108,6,6,160,0
GE 2310 DATA 177,203,230,203,208,2,230,20
   4,96
```

Listing 2: BASIC

```
BR 90 OPEN #1,4,0,"K:"
PZ 100 ? "K":LINE=2000:FOR N=1 TO 948:REA
   D D
DU 110 IF D<0 THEN GOSUB 400:? "K":LINE=L
   INE+10:POSITION 2,3:? LINE;" PP$(";(AB
   S(D));")=";CHR$(34);:READ D
IL 120 ? CHR$(D);:NEXT N:GOSUB 400
UT 130 RESTORE 900:? CHR$(125):POSITION 2
   ,3:? "2200 CH1$=";CHR$(34);:FOR N=1 TO
    74:READ D:? CHR$(D);:NEXT N:GOSUB 400
ID 140 ? CHR$(125):POSITION 2,3:? "2210 C
   H2$=";CHR$(34);:FOR N=1 TO 75:READ D:?
    CHR$(D);:NEXT N:GOSUB 400
LC 150 ? CHR$(125):POSITION 2,3:? "2220 C
   H3$=";CHR$(34);:FOR N=1 TO 75:READ D:?
    CHR$(D);:NEXT N:GOSUB 400
FG 160 ? CHR$(125):POSITION 2,3:? "2230 C
   IO$=";CHR$(34);:FOR N=1 TO 39:READ D:?
    CHR$(D);:NEXT N:GOSUB 400
ZZ 170 ? "KPRESS RETURN TO:":? "LIST 'D:T
   EMP',2000,2300"
WO 180 GET #1,K:IF K<>155 THEN 180
EX 190 LIST "D:TEMP",2000,2300:END
NY 400 ? :? :? "CONT":POSITION 2,0:POKE 8
   42,13:STOP
SH 410 POKE 842,12:RETURN
BR 500 DATA -1,104,104,141,3,6,104,141,2,
   6,104,141,5,6,104,141,4,6,104,141,7,6,
   104,141,6,6,104
KS 510 DATA 141,14,6,104,141,10,6,104,141
   ,15,6,104,141,11,6,169,0,141,0,6,141,1
   ,6,141,54,6
XV 520 DATA 169,144,162,5,27,157,54,6,74,
   202,208,249,169,12,141,60,6,169,40,141
   ,69,6,173,36,2,141,8
AJ 530 DATA 6,173,37,2,141,9,6,169,7,162,
   6,160,-91,93,76,92,228,104,104,104,170
   ,104,27,157,21,6,27,157,27,27
CN 540 DATA 6,104,27,157,18,6,27,157,24,6
   ,169,0,27,157,39,6,27,157,42,6,27,157,
   51,6,168,169,1,27,157,33,6,169
SK 550 DATA 36,224,2,208,3,169,48,200,27,
   157,36,6,152,27,157,27,30,6,169,166,27
   ,157,45,6,169,7,27,157,48,6,96
VX 560 DATA 104,169,7,174,9,6,172,8,6,32,
   92,228,169,0,162,7,27,157,0,210,234,20
   2,16,249,96,-181,0
JH 570 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   ,0,0,0,0,0,0,0,0,0,0
JJ 580 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   ,0,0,0,0,0,0,0,0,0,0
IF 590 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   ,0,0,0,0,0,0,162,0,32,96,6,232
MO 600 DATA 224,3,208,248,108,8,6,189,18,
   6,-271,133,203,189,21,6,133,204,222,33
   ,6,240,27,27,189,42,6,208
SF 610 DATA 18,189,33,6,221,48,6,176,10,1
   89,65,6,41,15,240,3,222,65,6,96,32,99,
   6,32,102,6
IO 620 DATA 133,207,41,248,201,128,208,18
   ,165,207,41,7,168,185,70,6,24,27,125,3
   6,6,27,157,36,6,24,144,227
LP 630 DATA 165,207,41,7,240,218,168,136,
   185,54,6,27,157,33,6,188,27,30,6,185,1
   0,6,133,205,185,-362,14,6,133
PK 640 DATA 206,165,207,74,74,74,41,15,20
   8,10,27,157,65,6,169,1,27,157,42,6,208
   ,78,168,136,185,78,6,24
DC 650 DATA 27,125,36,6,27,157,36,6,168,1
   89,45,6,27,157,65,6,177,205,24,27,125,
   39,6,27,157,61,6,224,2,208,13
OO 660 DATA 152,24,105,97,168,177,205,141
   ,64,6,24,144,26,189,45,6,41,240,201,16
   0,208,17,189,36,6,201
KG 670 DATA 50,144,10,189,45,6,41,15,9,-4
   52,192,27,157,65,6,165,207,41,128,27,1
   57,42,6,165,203,27,157,18,6,165
LB 680 DATA 204,27,157,21,6,96,160,0,165,
   207,41,120,208,21,189,51,6,240,5,222,5
   1,6,240,10,189,24,6
GD 690 DATA 133,203,189,27,27,6,133,204,9
   6,201,8,208,13,32,102,6,27,157,45,6,32
   ,102,6,27,157,48,6,96,201
GD 700 DATA 16,208,17,32,102,6,27,157,51,
   6,165,203,27,157,24,6,165,204,27,157,2
   7,27,6,96,201,24,208,-544,7,32,102
HX 710 DATA 6,27,157,39,6,96,201,32,208,2
   2,160,6,177,203,153,54,6,136,16,248,16
   9,7,24,101,203,133,203
DD 720 DATA 144,2,230,204,96,201,40,208,7
   ,32,102,6,141,1,6,96,201,48,208,7,32,1
   02,6,141,69,6
MH 730 DATA 96,201,56,208,7,32,102,6,27,1
   57,27,30,6,96,201,64,208,7,32,102,6,27
   ,157,36,6,96,140,0,6
AQ 740 DATA 96,209,223,237,251,9,27,30,44
   ,65,-634,79,100,121,149,165,189,217,24
   5,17,59,87,129,27,157,199,241,41,75
JF 750 DATA 121,177,233,33,117,173,1,57,1
   41,225,81,165,21,133,245,101,241,97,9,
   149,61,229,141,81,21,245
NC 760 DATA 213,209,205,1,27,253,49,101,2
   09,61,165,42,234,170,162,154,2,250,98,
   202,162,122,75,84,212,84,68
IQ 770 DATA 52,4,244,196,148,68,244,150,1
   68,168,168,136,104,8,232,136,40,136,23
   2,72,0,-724,0,0,0,1,1
XF 780 DATA 1,1,1,1,1,1,1,1,1,1,2,2,2,2,2
   ,2,2,3,3,3,3,3,4,4,4,5
RL 790 DATA 5,5,5,6,6,7,7,7,8,8,9,10,10,1
   1,11,12,13,14,14,15,16,17,19,19,21,22
JR 800 DATA 23,25,26,27,28,27,29,27,31,33
   ,35,38,39,42,44,47,50,53,56,59,63,67,7
   1,76,79,84,89,95,100
UC 810 DATA 106,112,119,27,126,134,142,15
   2,-814,27,159,169,179,190,201,213,1,3,
   6,9,12,15,18,22,23,25,26,27,28,27,29
SD 820 DATA 27,31,33,35,37,40,42,45,47,50
   ,53,57,60,64,68,72,76,81,85,91,96,102,
   108,114,121,128,136
OC 830 DATA 144,153,162,173,182,193,204,2
   17,230,243,27,255,110,116,122,131,137,
   146,27,0,167,173,185,197,206,221,233
BW 840 DATA 245,87,91,97,102,108,115,121,
   130
FE 900 DATA 234,234,234,234,234,234,173,0
   ,6,240,73,216,169,3,141,50,2,141,15,21
   0,173,61,6,141,0
WH 910 DATA 210,173,65,6,141,1,210,173,62
   ,6,141,2,210,173,66,6,141,3,210,173,63
   ,6,141,4,210
AC 920 DATA 173,67,6,141,7,210,173,64,6,1
   41,6,210,173,68,6,141,5,210,173,69,6,1
   41,8,210
VK 930 DATA 234,234,234,234,234,234,234,2
   34,234,234,234,234,234,234,173,0,6,240
   ,65,216,169,67,141,15,210
NG 940 DATA 162,6,160,3,185,61,6,27,157,0
   ,210,202,202,136,16,245,173,65,6,141,1
   ,210,173,66,6,141
MV 950 DATA 3,210,173,67,6,41,15,141,0,21
   1,169,0,141,5,210,141,7,210,173,69,6,1
   41,8,210
VB 960 DATA 234,234,234,234,234,234,173,0
   ,6,240,73,216,169,99,141,15,210,162,6,
   160,3,185,61,6,27,157
GS 970 DATA 0,210,202,202,136,16,245,173,
   65,6,141,1,210,173,66,6,41,15,133,0,17
   3,67,6,10,10
PA 980 DATA 10,10,5,0,141,0,211,169,0,141
   ,3,210,141,5,210,141,7,210,173,69,6,14
   1,8,210
HX 990 DATA 104,162,16,169,7,27,157,66,3,
   104,27,157,69,3,104,27,157,68,3,104,27
   ,157,73,3,104,27,157,72
PK 1000 DATA 3,32,86,228,132,212,169,0,13
   3,213,96
```

Listing 3: BASIC

```
OM 5 GOSUB 300
GJ 30 FOR S=0 TO 81:READ A,B:POKE 54016,A
   :POKE 53761,B:NEXT S
ZP 99 END
RV 100 DATA 15,160,15,160,15,160,31,160,4
   7,160,63,160,79,160,95,160,111,160,127
   ,160,142,160,157,160,172,160,187,160
HJ 110 DATA 202,160,217,160,232,160,247,1
   60,246,160,245,160,244,160,243,160,242
   ,160,241,160,240,161,224,162,208,163
TJ 120 DATA 192,164,176,165,160,166,144,1
   67,128,168,112,169,96,170,80,171,64,17
```

80 COL. SCREEN!—NEW PRICES!

Turbobase.™ **Winner ANTIC awards '88**

"IBM power without the price... I really can't think of any feature associated with running a business that has been left out—except for the **huge** prices charged for comparable software on MS-DOS computers."
—**ANTIC**, Dec. '87

"... the most time consuming review I have ever done, due to the number of features... Turbobase finally gives what 8-bit owners have been clamoring for for years; true, powerful business software... set up a fully capable business system for less than $1,000... customer support is superb... Practicality-excellent. Documentation-excellent."
—**COMPUTER SHOPPER**, Aug. '87

"... one of the most powerful and versatile database programs available..."
—**COMPUTER SHOPPER**, Aug. '88

COMPARE TO IBM CLONES:

• Capability	• Complete Documentation	• Speed among thousands of records
• Capacity	• $20-$50 Customizations	• Ease of learning (per feature)
• Remote Terminals	• One package/all modules	• Number of English error messages
• Exhaustive Support	• All Hardware Upgrades	• Adaptability to Existing Application
• No Disk Switching	• Brand Name Hardware	• Hardware/DOS easier than Clone/MS DOS™
• *Tiny* Footprint	• True Integration	• Faster Back-up to inexpensive floppy
• Not Copy Protected	• Free Application Set-up	• Complete Invoice/Payments Error Checking

Turbobase takes $20,000 video store sale from IBM... S.V., Plainfield, NJ
Turbobase takes $20,000 IBM sale for waterbed store... A.J., Phoenix, AZ
Turbobase replaces $37,000 air conditioning application... A.B., Alton, NH
Until you have Turbobase you don't have a database!... Acorn Users Group

Micromiser is looking for resellers. If you have 2 DD drives, or an MIO™, or hard disk, **You qualify** for free training, dealer prices, marketing/direct mail help, and myriad customer references who express extreme satisfaction with Turbobase. Compare the Turbobase™/MIO™ configuration at $830 (all hardware **& software** except printer) with the IBM AT™: Immediate **RAM access** to 6,000 invoices, or 15,000 inventory items, or 50,000 G/L records, or 20,000 payroll records, or any combination of above! With a hard drive (add only $100) the figures go up! 4,000 addresses too! An unbeatable selling point: replace any component for the cost of a typical IBM™/Apple™ repair bill! The small business market is yours! ***Just ask, "Is IBM™ compatibility worth $20,000 to you?"***

TURBOBASE™ – the *all in one* database/business system: 3 databases + word processor includes file manager/spread sheet/relational features/accounting/report generator, G/L, P/S, AR, AP, open invoicing/statements, inventory, payroll, mailing, utilities, all truly integrated in one program/manual so simplified that we can present complete *detailed* instructions in only 700+ pages of superb documentation (third re-write) includes separate Quick Course and Cookbook + 8 disk sides. Runs on any 48K 8-bit Atari, **only 1 drive req. Call today!**

Turbobase $159—Turbo Jr $99
For XEP—80 col. screen:
Turbobase 80 $179—Jr 80 $119 w/80 col word processor add $24 80 col word processor alone $49

ST owners! Ask about Ultrabase ST (B/W monitor only) all Turbobase features + much more + Ultimate SIMPLICITY and speed

(407) 857-6014

MICROMISER SOFTWARE, 1635-A HOLDEN AVE., ORLANDO, FL 32809

CIRCLE #105 ON READER SERVICE CARD.

```
   2,48,173,32,174,16,175,0,175,0,175
AZ 130 DATA 0,175,0,175,16,175,32,174,48,
   173,64,172,80,171,96,170,112,169,128,1
   68,144,167,160,166,176,165,192,164
RP 140 DATA 208,163,224,162,240,161,241,1
   60,242,160,243,160,244,160,245,160,246
   ,160,247,160,232,160,217,160,202,160
TN 150 DATA 187,160,172,160,157,160,142,1
   60,127,160,111,160,95,160,79,160,63,16
   0,47,160,31,160,15,160,15,160,15,160
FR 300 P=PEEK(54018):POKE 54018,P-4:POKE
   54016,255:POKE 54018,P
KD 310 SOUND 0,200,10,0:SOUND 1,200,10,0:
   SOUND 3,200,10,0
GZ 320 POKE 53775,99:RETURN
```

Listing 4: BASIC

```
OM 5 GOSUB 300
FJ 10 PITCH=INT(RND(0)*100)+15:CH=INT(RND
   (0)*3)+1
FE 20 ON CH GOSUB 100,120,140:FOR DE=1 TO
    50:NEXT DE:POKE 54016,0:POKE 53761,0
SM 30 FOR DE=1 TO 50:NEXT DE:GOTO 10
JL 100 POKE 53760,PITCH:POKE 53761,174:RE
   TURN
YG 120 POKE 53766,PITCH:POKE 54016,240:RE
   TURN
OZ 140 POKE 53762,PITCH:POKE 54016,15:RET
   URN
FW 300 P=PEEK(54018):POKE 54018,P-4:POKE
   54016,255:POKE 54018,P:REM JOYSTICKS F
   OR OUTPUT
UJ 310 POKE 53775,99:POKE 53768,0:RETURN
```

No Frills Software 800 E.23rd St. Kearney, Ne. 68847 (308) 234-6250 M-Sat. 11-7

THE CONVERTER
It's Here! Only $19.95(+$2 shipping) 8-bit

Now you can convert your PrintShop icons to Awardware, Printpower or Newsroom format. The Converter also converts Awardware seals to graphics or vice versa or to Printpower or Newsroom form. It also has a graphics editor to allow touchup of the icons or to make your own icons from scratch.

Also available at $19.95(+ shipping)

P.S. Users Utility Disk

Almost a dozen features! A icon viewer for your PrintShop icons, an icon cataloger, border cataloger, font cataloger, a transfer program for moving icons from disk to disk, a multi-size label maker(includes sizes for video tapes-inside & spine), a bookmark maker, envelope cachet maker, coupon maker & more. Currently supports Epson & fully Epson compatible printers, send SASE for details & all printers supported). $2 shipping covers both programs. PrintShop is a trademark of Broderbund Software, Awardware & Printpower trademarks of Hi-Tech Expressions, Newsroom is a trademark of Springboard Software, not affiliated with NO FRILLS SOFTWARE.

For 8bit PrintShop and Printmaster (ST)

Nearly 6000 icons, fonts & borders from $2 to $23 a disk(group specials available)
Fonts & Borders 1-4 now $14.95.
Christian Scenes & Symbols 1 or 2 $19.95 each.
DavkaGraphics 1,2,3(Hebrew icons/fonts) $23 each.
PS GRAPHICS 1 to 7 now $9.95 each.
JACS PS Graphics 1 to 4 $9.95 each (the ONLY authorized source for the JACS disks).
Budget Graphics 1 to 16 now $4.95 each
Budget Graphics 17-24 $2 each
NERDS MAP disk set $15 or $9 each
NERDS Biology set $15 or $9 each
NERDS Chem(periodic table) disk $9
COAT OF ARMS Const. Set now $13
NEW! Beagle Bros. MINIPIX disks 1,2,3 (200 icons per disk) $19.95 each. $49/95 for all 3.
NEW! Fonts & Borders 5 and 6 $14.95 each (#6 ready late November). Each font disk has more than 15 fonts and many borders(#3 has fewest borders).

Add $1.50 shipping for first PS/PM disk ordered, add $.50 each additional disk to max of $5.(US shipping). Add $1 additional to Canada/Mexico, $4 other countries. Prices on Ps/Pm disks good thru 1988. Add 4% VISA/MC. checks-2 week clearance/ Prices above for 8-bit disks. ST disks add $1 per title. All disks not available for ST. Call or write (SASE $.45 postage) for details/printouts. Discounts available for large orders (call/write), ST icon disks available in package groups (call/write).

Printmaster is a trademark of Unison World not affiliated with NO FRILLS SOFTWARE.
ST icon disks (individual disks only, not package disks) also contain DEGAS formats of icons.

We also carry a variety of other ATARI hard/software.
Send SASE for list (only 1 SASE needed for all areas here, please specify areas of interest-PS, pgms,etc).

8bit

ACE OF ACES- $9.25	BRIDGE 4.0-$14.99
BRIDGE 5.0-$17.99	COMPUBRIDGE-$14.99
LINKWORD(Ital,French,German,Spanish) $14.99 ea.	
Strip Poker-$17.99	DATA DISKS 1,2,3 $11.99 ea.
PEGGAMON-$9.25	GUDERIAN-$17.99
GULF STRIKE-$17.99	SPITFIRE '40-$20.99
BLAZING PADDLES-$20.99	TYCOON-$11.99
Video Vegas or Rainy Day Games-$17.99ea.	
221 Baker St-$17.99	221 Baker case disk-$10.99
Alt.Reality(Dungeon)-$25.75	Alt.Reality(city)-$18.75
BISMARK-$19.99	VIDEO TITLE SHOP-$19.99
TOMAHAWK-$19.99	CHESSMASTER 2000-$25.79
EPYX 500XJ Joystick-$11.97	Spy vs Spy 3-$11.97
PrintPower-$9.99	Astro Grover-$6.49
Sesame St. Print Kit-$9.49	Celeb Cookbook-$20.99
Gauntlet-$20.99	MIO BOARD 1 MEG-$254.99
NEWSROOM-$29.99	TRAILBLAZER-$17.99

ST titles

Pinball Wizard-20.99	221 Baker St. $25.75
DEGAS ELITE-$38.99	MARK WILLIAMS "C"-$119.99
WORD PERFECT-$195	MONITOR MASTER-$41.95
HUNT For Red October-$31.95	OIDS-$21.99
SUNDOG-$24.99	IB 40 Track drive-$189.99

SHIPPING: add $2 first title ordered. add $1 each added title. Hardware-call for shipping. Add 4% VISA/MC. Call for details on credit card guarentee ordering. Prices of outside software subject to change.

CIRCLE #106 ON READER SERVICE CARD.

Attention Programmers!

ANALOG Computing is interested in programs, articles, and software review submissions dealing with the Atari home computers. If you feel that you can write as well as you can program, then submit those articles and reviews that have been floating around in your head, awaiting publication. This is your opportunity to share your knowledge with the growing family of Atari computer owners.

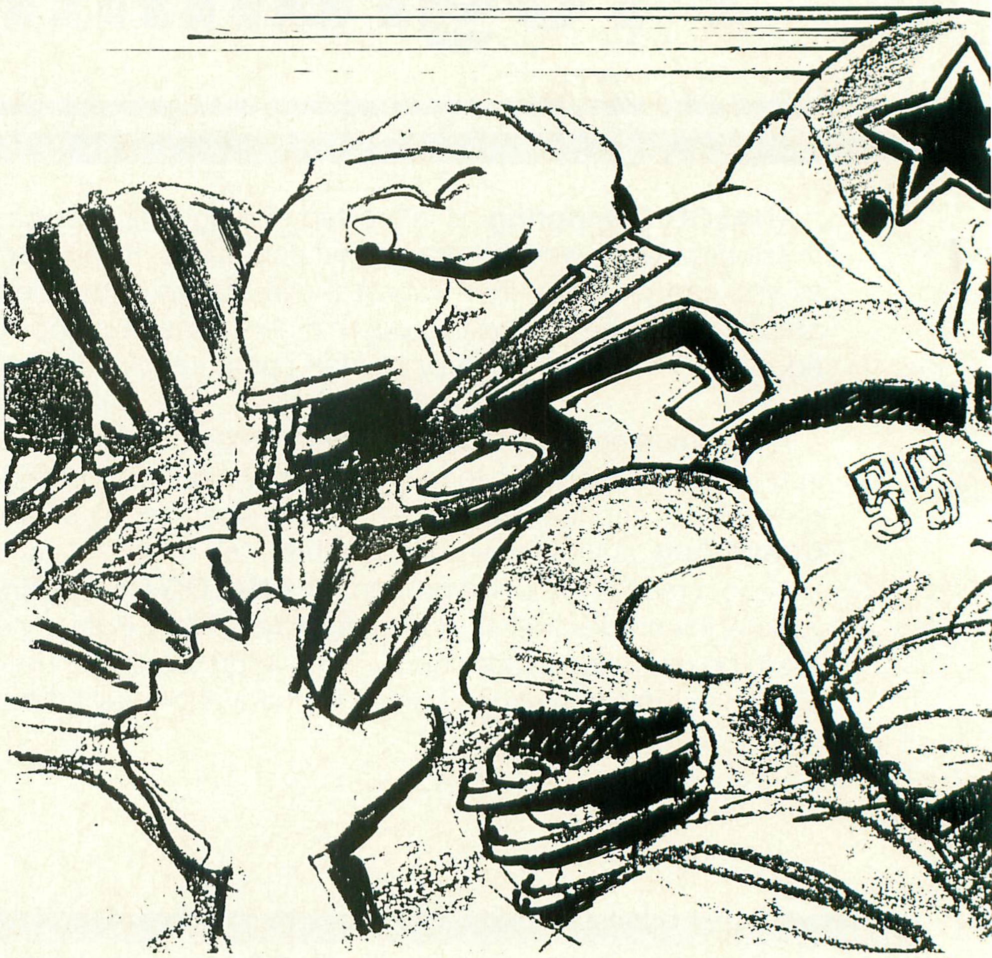
All submissions for publication, both program listings and text, should be provided in printed and magnetic form. Typed or printed copy of text is mandatory and should be in upper and lower case with double spacing. By submitting articles to **ANALOG Computing**, authors acknowledge that such materials, upon acceptance for publication, become the exclusive property of **ANALOG Computing**. If not accepted for publication, the articles and/or programs will remain the property of the author. If submissions are to be returned, please supply a self-addressed, stamped envelope. All submissions of any kind must be accompanied by the author's full address and telephone number.

Send your programs and articles to:
ANALOG Computing
P.O. Box 1413-M.O.
Manchester, CT 06040-1413

3 in 1 Football

Lance Haffner Games
P.O. Box 100594
Nashville, TN 37210
(615) 242-2617
48K disk
$39.95

reviewed by Dave Arlington

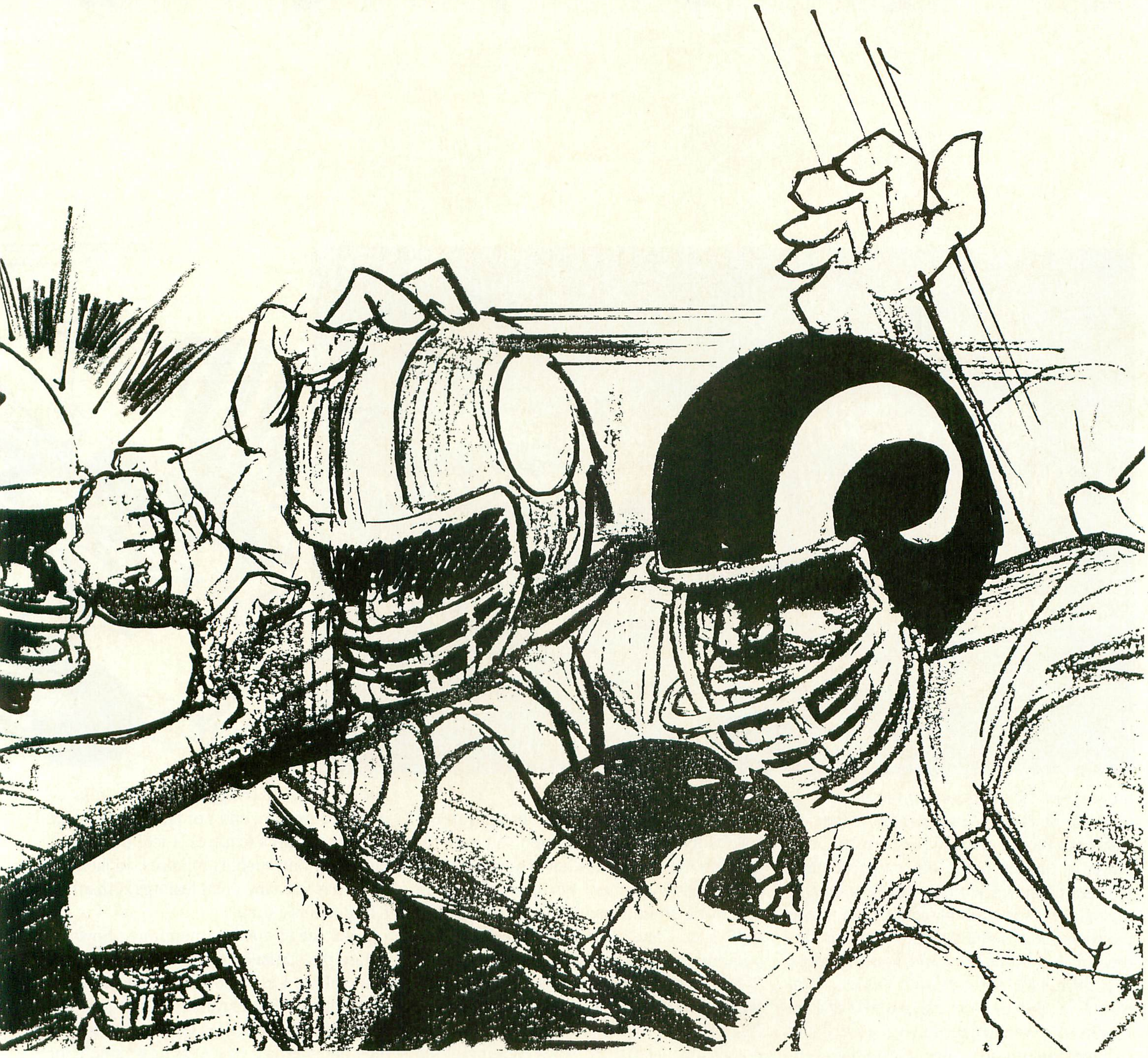


What with the player's strike early in the 1987 season, I once again began my long quest last fall to find a decent football game for my Atari computer. I was not looking for an arcade-type football game like *Touchdown Football* or *Gamestar Football*. I was looking for a strategic football game that would allow me to coach real NFL football teams and their players like *Microleague Baseball* lets me manage real baseball players. Unfortunately, up until late last year, there were no such football games for the Atari.

There were some attempts before this year. *Gridiron Glory*, late of APX and now from Main ST. Software, was the first to allow you to use real football teams. I'm not sure exactly what statistics they used, but the game was very unrealistic as the quarters were only eight minutes long and the computer coach called plays randomly. Next was *Football Strategy* from Avalon Hill. That game featured 16 historical NFL teams, mostly Super Bowl participants. Again the team values used were hard to judge and did not feature individual players. They also didn't seem to have too much effect on game play, as it was usually easy for me to beat the computer, with me taking teams like the (9-7) N.Y. Giants and giving the computer teams like the (14-0) Miami Dolphins.

The most recent attempt is *Computer Quarterback* from SSI. It is sold as a strategy football game using generic football teams. In this respect, it is the best football game available for Atari, since on offense you can choose from 36 different types of plays and 20+ different types of defenses. The computer coach can also learn to play better against you if you call the same types of plays over and over again. Computer Quarterback also includes a draft feature where general managers are given a certain amount of money to spend on a team to improve certain player positions. However, neither option uses real NFL teams or players. SSI began to make available seasons disks with the NFL teams for an entire year. Currently 1984, 1985 and 1986 seasons disks are available. Again, however, teams are rated abstractly by position and individual players are not available.

Which brings us to the star of this review: *3 in 1 Football* from Lance Haffner Games. This statistical-based football game allows you to play with college, NFL, USFL and WFL football teams with the rules for college; NFL and USFL being available to play under. Teams are rated in many offensive and defensive categories in-



cluding the types of plays usually called by the teams. Individual players are rated at the following positions: quarterback, running back (both running and receiving), receivers, kickers, punters and return specialists. Everyone else is rated on a team basis.

Let's talk about the bad parts of the game first to get them out of the way. The game is written in BASIC, as it was translated from many other computers. Therefore, it can be very slow at times when loading the game or the data for the teams. The actual game itself is very fast. There are absolutely no graphics in this game, so anyone expecting a fancy display should be forewarned. It is simple Graphics 0 text, white on a black background. I can understand this as the graphics were foresaken to use as much memory as possible for the many options this game has.

You may play against a friend, against the computer or watch the computer play itself. On defense, you may choose from six different formations, each having different types of effectiveness on certain types of plays. On offense, you may choose from 14 different offensive plays, or get a quickie scouting report. As I mentioned above, the actual game play itself is fast, with an average game against the computer lasting maybe 15 to 20 minutes. When the computer takes on itself, games last about five minutes. This fast game is nice, as it actually makes the possibility of replaying an entire season a reality. This would be impossible with any of the other longer football games mentioned above.

The game uses very realistic statistics for

Players perform as they do in real life. Quarterbacks will pick alternate receivers or scramble for first downs; so your long pass attempt might result in a two- or three-yard gain or loss.

the players. However, since the game features no injuries, the program picks the ball carrier or passer on each play for you. The player is chosen based on the percentage of times he actually ran or threw or caught the ball. This is good in one respect, since the statistics generated by this game are incredibly accurate. The bad point is when it is third and long, all of a sudden, Don Strock is throwing the ball instead of Dan Marino. Or you are faced with a third and one and you do not know whom the computer will choose to run the ball on the play.

There are other things that take getting used to as well. First, all yardage is measured in tenths during the game. For example, Payton carries on the inside run for 3.7 yards, making it three and 13.3 to go. This bothered me until I realized that this was probably more realistic than the other football games, since rarely does a player in real life gain exactly three yards or exactly five yards. This sets up situations in 3 in 1 Football where you really do have fourth and inches to go. And besides, when calculating stats at the end of the game, all yardage is rounded off as in real life to the nearest yard.

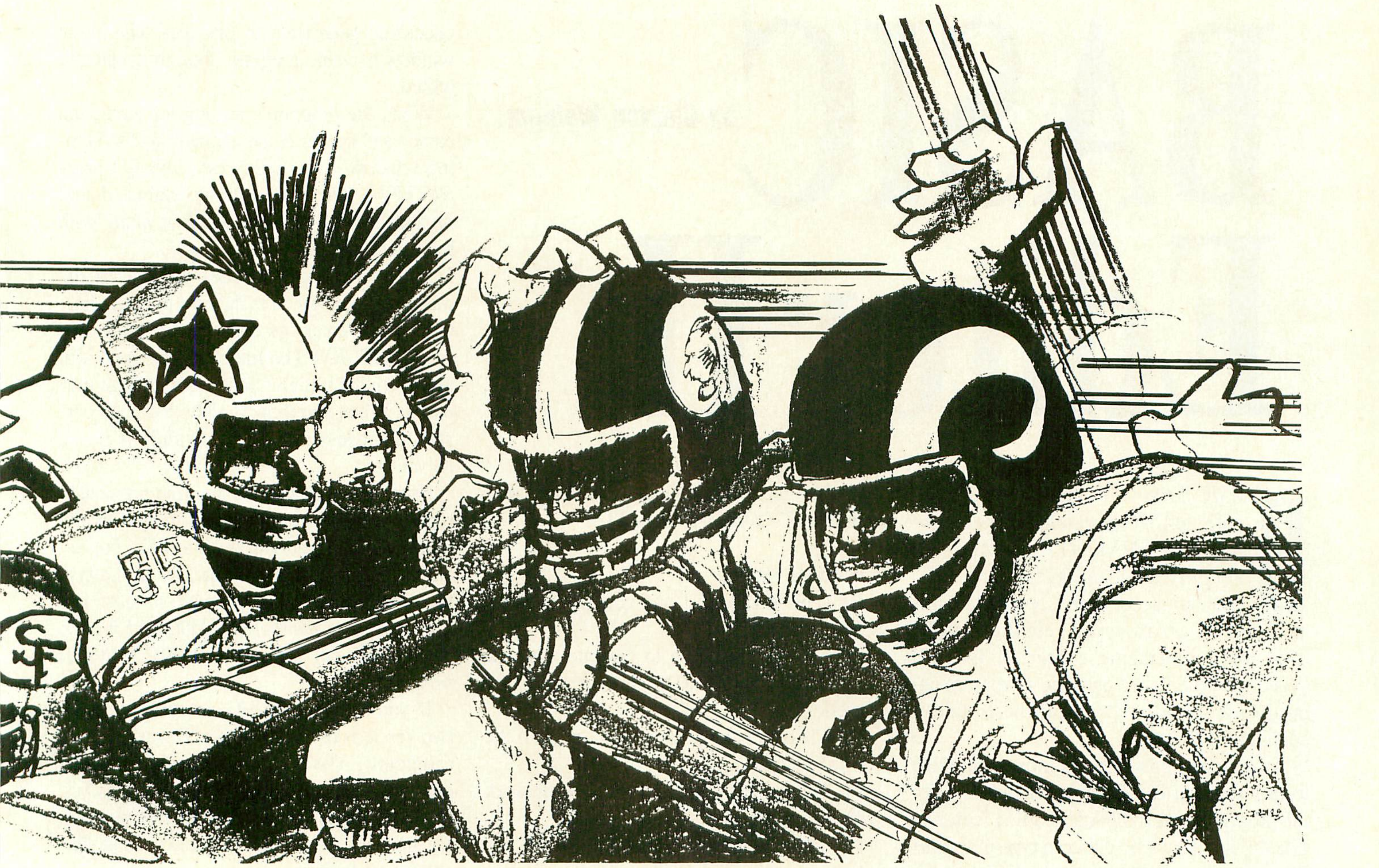
Due to an oversight in programming (probably lost in a conversion somewhere) time on the clock does not always show all the necessary zeroes. For example, when the clock should say 2:08 left in the first half, it will say 2:8. This can be confusing sometimes as you might go from 2:43 to 2:5 on a play, and it looks like the clock went backwards. Another programming bug crops up when displaying accumulated seasons' statistics. After looking at the first team, the program will not allow you to look at another team without first quitting the stat-viewing program and reloading it again.

Also, printing statistics to your printer results in a tremendous waste of paper, as each screen is dumped in total to your printer even if it might contain 20 blank lines. Since there are about five screens of information, the results of one game can cross several pages.

After all this, you're probably saying that this game has some real problems. *Au contraire, mon ami*. Sure, it has what I consider small problems, but most of these are due in part to being translated from so many computers and not taking advantage of all the Atari's features. I feel the game's pluses far outweigh its minuses. In fact, the game has so many great features, I'm afraid I might miss some!

Let's see, I already mentioned how accurate the statistics were and how fast the game plays. Let's get back to the accuracy of the game first. Players really do perform as they do in real life. Quarterbacks will pick out alternate receivers or scramble for first downs, so your long pass attempt might only result in a two- or three-yard gain or loss. When playing against the computer, teams play as they do in real life. You can expect the Rams to keep it on the ground with their boring philosophy, while Miami will be sure to air it out quite a bit. Playing Tampa Bay against Chicago is usually the mismatch it is in real life.

The computer coach is good. It will run a lot more if leading in the last quarter, and throw more if behind. The game features a two-minute offense in the NFL version, and you can work the clock by throwing sideline passes, calling time-outs and run-



ning a "hurry up" offense. Games can be played in either team's stadium or at a neutral site as is done for the Super Bowl.

The game keeps a complete record of all game statistics, and they may be shown on the screen or on your printer at the end of the game. The game also includes a statistics compiler that will keep track of a team's record and the accumulated statistics for all its games. So not only does the game play fast enough to complete an entire season, but at the end you will have complete statistics for every team. All this is included at no extra cost.

I have saved the best for last. How many teams do you get with the game? 50? 100? Try almost 600! That's right, almost 600 completely rated and accurate teams with all the player types and team ratings I've mentioned above. You get all 28 NFL teams from 1986, 180 college football teams from 1986 and 12 WFL teams from 1974. (Normally the USFL had been included, but since the USFL did not play in 1986, the WFL teams were included instead. Don't ask me why.) You also get 174 of the best college football teams of all time, including the 1967 USC team with O.J. Simpson.

And last but not least, you get 96 NFL teams from past seasons. Or do you? Here was where I got a big surprise. I just happened to be poking around on the disk with the old-timer NFL teams when I noticed a team that was not on the list that was enclosed with the game. Looking a little closer at the list that came with the game, I noticed some team numbers were missing. It turns out that you actually get not 96, but 189 past NFL pro football teams on the disk! With all the pro teams available on this disk, you can replay every Super Bowl from 1966 to 1982. Taking my hometown Buffalo Bills as an example of the wide range of teams that are offered, you can play with the Bills' teams that are included from 1986, 1981, 1975, 1974, 1973, 1971, the 1-12-1 1968 team with Eddie Rutkowski, 1966, 1965, 1964 and even the 1948 Buffalo Bills of the AAFC! Imagine, 11 different Buffalo Bills are yours to coach.

If you are a fan of some other team like the Miami Dolphins, Oakland Raiders or Dallas Cowboys, you will find an equally large range of your favorite teams to choose from. It is really quite interesting to play 1972 Miami with Bob Griese, Larry Csonka and Jim Kiick against Vince Lombardi's 1966 Green Bay Packers with Bark Starr, Boyd Dowler and Jim Hornung. All the greats are here to coach in their prime: Joe Namath, O.J. Simpson, Jim Brown, Otto Graham, Walter Payton, Fran Tarkenton and many, many more.

All in all, you get a total of 583 past and present, college and pro teams. That alone makes this game a great value. Add the great statistical accuracy and your ability to replay some "dream" football games, and you have a real winner that is a definite bargain and a great enjoyment.

Dave Arlington, an Atari devotee since 1983, has recently graduated with an A.S. in computer science and math. He is rediscovering his Atari while looking for "real" employment. He enjoys programming in Action! and computer simulations of all types.

BASIC Editor II

by Clayton Walnum

BASIC Editor II is a utility to help you enter BASIC program listings published in ANALOG Computing. To simplify the identification of errors, each program line is evaluated immediately after it's typed, eliminating the need for cumbersome checksum listings. When you've finished entering a program using BASIC Editor II, you can be certain it contains no typos.

An option is provided for those who wish to use standard BASIC abbreviations. Also, the program retains all Atari editing features. Finally, for those who prefer to type programs the conventional way, using the built-in editor, a post-processing mode is available. It allows you to check typing after the entire listing has been entered.

Typing in the Editor

To create your copy of BASIC Editor II, follow the instructions below— exactly.

Disk version:

(1) Type in Listing 1, then verify your work with Unicheck (see Issue 39).

(2) Save the program to disk with the command *SAVE "D:EDITORL1.BAS".*

(3) Clear the computer's memory with the command *NEW.*

(4) Type in Listing 2, then verify your work with Unicheck.

(5) Run the program (after saving a backup copy) and follow all the on-screen prompts. A data file will be written to your disk.

(6) Load Listing 1 with the command *LOAD "EDITORL1.BAS".*

(7) Merge the file created by Listing 2 with the command *ENTER "D:ML.DAT".*

(8) Save the resultant program with the command *LIST "D:EDITORII.LST".*

Cassette version:

(1) Type in Listing 1 and verify your work with Unicheck.

(2) Save the program to cassette with the command *CSAVE.* (Do not rewind the cassette.)

(3) Clear the computer's memory with the command *NEW.*

(4) Type in Listing 2 and verify your work with Unicheck.

(5) Run the program and follow the on-screen prompts. A data file will be written to your cassette.

(6) Rewind the cassette.

(7) Load Listing 1 with the command *CLOAD.*

(8) Merge the file created by Listing 2 with the command *ENTER "C:".*

(9) On a new cassette, save the resultant program with the command *LIST "C:".*

Using the Editor

Take a look at one of the BASIC program listings in this issue. Notice that each program line is preceded by a two-letter code. This code is the checksum for that line; it's not a part of the program.

To enter a program listing from the magazine, load BASIC Editor II with the *ENTER* command, and run it. You'll be asked if you wish to allow abbreviations (see your BASIC manual). If you do, type *Y* and press *RETURN*. Otherwise, type *N*.

Note: If you set BASIC Editor II to allow abbreviations, the program will run slightly slower.

Your screen will now be divided into two "windows." The upper window will display each line after it's processed, as well as the checksum generated for that line. The lower window is where program lines are typed and edited.

When the program's waiting for input, the cursor will appear at the left margin of the typing window. Type a program line and press *RETURN*. The line will be evaluated and reprinted in the message window, along with the checksum generated.

If the checksum matches the one in the magazine, then go on to the next program line. Otherwise, enter the command *E* (edit) and press *RETURN*. The line you just typed will appear in the typing window, where you may edit it. When you think the line has been corrected, press *RETURN*, and it'll be reevaluated.

Note: You may call up any line previously typed, with the command *E* followed by the number of the line you wish to edit. For example, *E230* will print Line 230 in the typing window. *Do not attempt to edit any program lines numbered 32600 and higher.* These lines fall within the BASIC Editor II program.

If you're using BASIC abbreviations, the two versions of the command *E* work slightly differently. The command *E*, without a line number, will call up the line exactly as you typed it. When you append the line number, the line will be printed in its expanded (unabbreviated) form.

Leaving the Editor

You may leave BASIC Editor II at any time, by entering either *B* (BASIC) or *Q* (quit). If you type *B*, the Editor will return you to BASIC. Enter *LIST* to review your work, if you wish. Note that lines 32600 and above are the Editor program. Your work will appear before these lines. To return to the Editor, type *GOTO 32600.*

Type *Q*, and you'll be asked if you really want to quit. If you type *Y*, the Editor program will be erased from memory, and you may then save your work in any manner you like. If you type *N*, the *Q* command will be aborted.

Large listings

If the program you're entering is particularly long, you may need to take a break. When you want to stop, type *Q* and press *RETURN*, then save your work to disk or cassette. When you're ready to start again, load the program you were working on, then load BASIC Editor II with the *ENTER* command. Type *GOTO 32600*, and you're back in business.

The post-processor

Many people may not want to use BASIC Editor II when entering a program listing, preferring, instead, the Atari's built-in editor. For that reason, BASIC Editor II will allow you to check and edit your programs after they've been typed.

To take advantage of this option, type any magazine program in the conventional manner, then save a copy to disk or cassette (just in case). With your typed-in program still in memory, load BASIC Editor II with the *ENTER* command, then type *GOTO 32600.*

Respond with *N* to the "abbreviations" prompt. When the Editor appears on your screen, enter the command *P* (post-process), and the first program line will appear in the typing window. Press *RETURN* to enter it into the Editor.

The line will be processed, and the checksum, along with the program line, will be printed in the upper window. If the checksum matches the one in the magazine, press *RETURN* twice, and the next line will be processed.

If you find you must edit a line, enter the command *E*, and the line will be moved back to the typing window for editing.

When the entire listing has been checked, you'll be asked if you wish to quit. Type *Y* and press *RETURN*. The Editor program will be removed from memory, and you may then save the edited program in any manner you wish.

Murphy's Law

Anyone who's been associated with computing knows this is the industry Murphy had in mind. You may find that, after typing a program with BASIC Editor II, it still won't run properly. There are two likely causes for this.

First, it may be that you're not following the program's instructions properly. Always read the article accompanying a program *before* attempting to run it. Failure to do so may present you with upsetting results.

Finally, though you can trust BASIC Editor II to catch your typos, it can't tell you if you've skipped some lines entirely. If your program won't run, make sure you've typed all of it. Missing program lines are guaranteed trouble.

One last word: Some people find it an unnecessary and nasty chore to type REM lines. I don't condone the omission of these lines, since they may be referenced within the program (a bad practice, but not unheard of). If you want to take chances, BASIC Editor II is willing to comply.

When you've finished entering a program using BASIC Editor II, you can be certain it contains no typos.

Listing 1.
BASIC listing.

```
32600 IF FL THEN 32616
32602 DIM L$(115),SV$(115),C2$(2),B$(1
15),M$(119),S$(98),E$(69),A$(1):FL=1:S
TMTAB=PEEK(136)+PEEK(137)*256
32604 GRAPHICS 0:POKE 710,0:P=0:ABR=0:
? "ALLOW ABBREVIATIONS";:INPUT A$:IF A
$="Y" OR A$="y" THEN ABR=1
32606 B$(1)=" ":B$(115)=" ":B$(2)=B$
32616 OPEN #17,4,0,"E:":L$=" ":GOSUB 3
2662:START=0
32618 POKE 766,1:POKE 83,39:POSITION 1
,3:IF LEN(L$)<39 THEN ? L$:GOTO 32624
32620 IF LEN(L$)<77 THEN ? L$(1,38):?
L$(39,LEN(L$)):GOTO 32624
32622 ? L$(1,38):? L$(39,76):? L$(77,L
EN(L$))
32624 POKE 752,0:POKE 766,0:POKE 559,3
4:POKE 82,1:POKE 83,38:POSITION 0,10:?
 " ";:INPUT #17;L$:POKE 766,1
32626 IF (L$="P" OR L$="p") AND START=
0 THEN P=1:L$=""
32628 IF L$="E" OR L$="e" THEN E=1:POS
ITION 1,10:? SV$ GOTO 32624
32630 IF L$="Q" OR L$="q" THEN 32690
32632 IF L$="" AND P=1 THEN 32686
32634 IF L$="" THEN 32624
32636 IF L$="B" OR L$="b" THEN GRAPHIC
S 0:? "TYPE 'GOTO 32600' TO CONTINUE":
END
32638 IF L$(1,1)="E" OR L$(1,1)="e" TH
EN E=1:TRAP 32624:EL=VAL(L$(2)):POSITI
ON 1,9:LIST EL:GOTO 32624
32640 SV$=L$:TRAP 32624:X=VAL(L$)
32642 START=1:IF P AND  NOT E THEN 326
52
32644 GOSUB 32674:IF  NOT ABR OR P THE
N 32652
32646 POKE 766,0:? CHR$(125):POSITION
0,3:L=VAL(L$):LIST L:? :? :? "CONT":L$
=B$
32648 POSITION 0,0:POKE 842,13:STOP
32650 POKE 842,12:A=USR(ADR(S$),ADR(L$
),4):L$=L$(1,A)
32652 CHKSUM=USR(ADR(M$),ADR(L$),LEN(L
$)):CHKSUM=CHKSUM+PEEK(1542)*65536
32654 CHK=CHKSUM-(INT(CHKSUM/676)*676)
:HI=INT(CHK/26):LO=CHK-(HI*26):C2$(1)=
CHR$(HI+65):C2$(2)=CHR$(LO+65)
32656 IF  NOT P OR E THEN E=0:GOSUB 32
662:IF  NOT P THEN 32660
32658 POKE 83,39:POKE 752,1:FOR X=3 TO
 5:POSITION 1,X:? B$(1,38):POSITION 1,
X+7:? B$(1,38):NEXT X:POKE 83,38
32660 POKE 766,1:POKE 83,38:POSITION 6
,7:? C2$:POKE 752,0:GOTO 32618
32662 GOSUB 32702:POKE 766,0:POKE 752,
1:? "K":POKE 82,1:DL=PEEK(560)+256*PEE
K(561)+4
32664 POKE DL-1,70:POKE DL+2,6:POKE DL
+3,112:POKE DL+4,112:POKE DL+5,112:POK
E DL+13,112:POKE DL+14,112
32666 POKE DL+22,112:POKE DL+23,112:PO
KE DL+24,65:POKE DL+25,PEEK(560):POKE
DL+26,PEEK(561):POKE 83,39
32668 POSITION 20,0:? "  basic editor
ii  ":POSITION 0,7:? "             TYP
ING WINDOW                    "
32670 POSITION 0,1:? "            MESS
AGE WINDOW                ":POSITION 1,7
:? "CODE:";
32672 POKE 559,34:RETURN
32674 GRAPHICS 0:POKE 559,0:POKE 766,1
:POKE 82,0:POKE 83,39:POSITION 0,3:? L
$:? :? :? :? "CONT":POSITION 0,0
32676 POKE 842,13:STOP
32678 POKE 842,12:TRAP 32682:A=USR(ADR
(E$),VAL(L$)):IF A=4 THEN POP :GOTO 32
682
32680 RETURN
32682 GOSUB 32662:SOUND 0,75,10,8:FOR
X=1 TO 20:NEXT X:SOUND 0,0,0,0:POSITIO
N 1,3:? "SYNTAX ERROR!":POKE 766,1
32684 POKE 83,38:POSITION 1,10:? SV$:G
OTO 32624
32686 LINE=PEEK(STMTAB)+PEEK(STMTAB+1)
*256:IF LINE>32599 THEN 32690
32688 OFS=PEEK(STMTAB+2):STMTAB=STMTAB
+OFS:POSITION 1,9:LIST LINE:GOTO 32624
32690 POKE 766,0:POSITION 1,10:? "READ
Y TO QUIT";:INPUT A$:IF A$<>"Y" THEN P
OSITION 1,10:? B$(1,38):GOTO 32624
32692 GRAPHICS 0:? :? :? :FOR X=32600
TO 32636 STEP 2:? X:NEXT X:? "CONT":PO
SITION 0,0:POKE 842,13:STOP
32694 POKE 842,12:GRAPHICS 0:? :? :? :
FOR X=32638 TO 32674 STEP 2:? X:NEXT X
:? :? "CONT":POSITION 0,0
32696 POKE 842,13:STOP
32698 POKE 842,12:GRAPHICS 0:? :? :? :
FOR X=32676 TO 32702 STEP 2:? X:NEXT X
:? :? "POKE 842,12":POSITION 0,0
32700 POKE 842,13:STOP
32702 POKE 16,112:POKE 53774,112:RETUR
N
```

CHECKSUM DATA.
(see issue 39's ***Unicheck***)

```
32600 DATA 6,665,923,757,809,171,225,8
98,532,499,910,267,912,144,735,8453
32638 DATA 97,358,230,693,706,878,317,
127,36,597,238,258,182,430,168,5315
32668 DATA 864,953,472,385,887,724,72,
687,908,736,625,612,672,184,891,9672
32698 DATA 8,856,85,949
```

Listing 2.
BASIC listing.

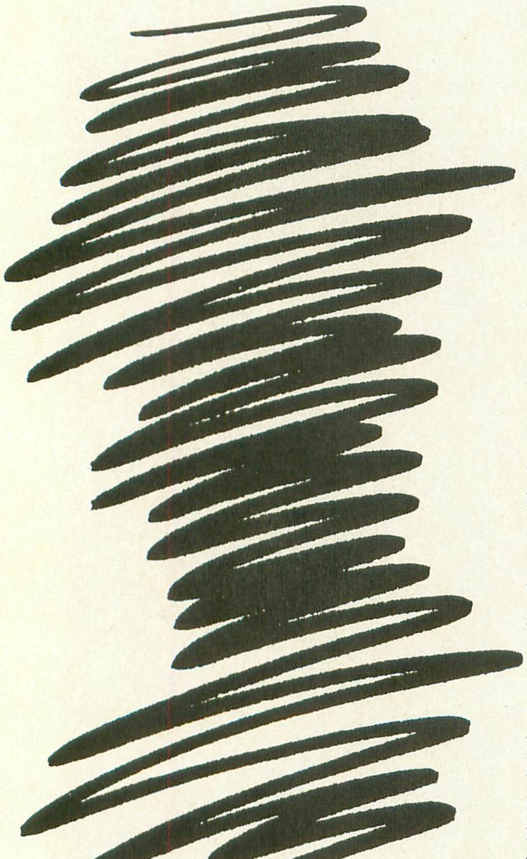
```
10 DIM L$(120),ML$(119),A$(1)
20 GRAPHICS 0:POKE 710,0:? "DISK OR CA
SSETTE";:INPUT A$:IF A$<>"C" AND A$<>"
D" THEN 20
30 IF A$="C" THEN 50
40 ? "PLACE FORMATTED DISK IN DRIVE":?
 "THEN PRESS RETURN":INPUT L$:OPEN #1,
8,0,"D:ML.DAT":GOTO 60
50 ? :? "READY CASSETTE, PRESS RETURN"
;:INPUT L$:OPEN #1,8,0,"C:"
60 L$="32608 M$(1)=":L$(13)=CHR$(34)
70 N=119:GOSUB 130:L$(14)=ML$(1,58):L$
(LEN(L$)+1)=CHR$(34):? #1;L$
80 L$(1)="32610 M$(59)=":L$(14)=CHR$(3
4):L$(15)=ML$(59):L$(LEN(L$)+1)=CHR$(3
4):? #1;L$
90 L$(1)="32612 S$=":L$(10)=CHR$(34)
100 ML$="":N=98:GOSUB 130:L$(11)=ML$:L
$(LEN(L$)+1)=CHR$(34):? #1;L$
110 L$(1)="32614 E$=":L$(10)=CHR$(34)
120 ML$="":N=69:GOSUB 130:L$(11)=ML$:L
$(LEN(L$)+1)=CHR$(34):? #1;L$:END
130 FOR X=1 TO N:READ A:ML$(X)=CHR$(A)
:NEXT X:RETURN
140 DATA 104,104,133,204,104,133,203,1
04,104,133,205,169,0,141,3,6,141,2,6,1
41,4,6,141,5,6
150 DATA 141,6,6,238,3,6,32,68,218,172
,2,6,177,203,133,212,32,170,217,32,182
,221,32,68,218
160 DATA 173,3,6,133,212,32,170,217,32
,219,218,32,210,217,165,212,141,0,6,16
5,213,141,1,6,24
170 DATA 173,0,6,109,4,6,141,4,6,173,1
,6,109,5,6,141,5,6,144,3,238,6,6,238,2
180 DATA 6,172,2,6,196,205,208,176,173
,4,6,133,212,173,5,6,133,213,96
190 DATA 104,104,133,204,104,133,203,1
04,104,141,255,6,169,0,133,213,216,165
,88,133,205,165,89,133,206
200 DATA 174,255,6,24,165,205,105,40,1
33,205,144,2,230,206,202,208,242,160,0
,177,205,201,64,144,18
210 DATA 201,96,144,19,201,128,144,18,
201,192,144,6,201,224,144,7,176,8,24,1
05,32,144,3,56,233
220 DATA 64,145,203,200,192,114,240,2,
208,215,177,203,201,32,208,3,136,208,2
47,200,132,212,96
230 DATA 104,104,141,254,6,104,141,253
,6,169,0,133,213,216,165,136,133,205,1
65,137,133,206,160,0,177
240 DATA 205,205,253,6,208,8,200,177,2
05,205,254,6,240,15,160,2,177,205,24,1
01,205,133,205,144,228
250 DATA 230,206,176,224,160,4,177,205
,201,55,240,4,160,0,240,0,132,212,96
```

CHECKSUM DATA.
(see issue 39's ***Unicheck***)

```
10 DATA 203,265,465,844,294,973,652,27
0,978,797,278,275,835,209,301,7639
50 DATA 355,94,254,420,935,840,580,41
,974,564,5435
```

AUTORUN.SYS Secrets

by LeRoy Baxter



Ever wondered how the AUTORUN.SYS file worked with BASIC? Ever wanted to automatically run a BASIC program that wasn't named MENU? Ever had a conflict between the autorun loader and a machine-language routine that the program needed? How about a two-stage LOAD and ENTER situation? Or an AUTORUN.SYS for a language other than BASIC?

I've seen a lot of AUTORUN.SYS makers, but none have resolved all the questions and few that will let you autorun any program but MENU.

I disassembled one of my AUTORUN.SYS files and delved into its secrets. I made some changes and eliminated some potential problems. The result is Listing 1, an AUTORUN.SYS maker that is more flexible than any you've ever seen. It works with any Atari-type DOS (DOS 2.5, MyDOS, etc.) and with any language.

The secrets revealed

Normally, when BASIC comes up, it prints the READY prompt on the screen and then calls the editor to accept a line from the keyboard. While all the editor routines are in OS ROM, the designers of the Atari have allowed us unlimited flexibility by putting the address of the editor routines in a RAM table called HATABS. By replacing the editor address in HATABS, we can supply new routines that make the computer do what we want.

The program is divided into three parts. The first part finds the vector for the editor and replaces it with our own. The handler table is searched from the bottom up just in case a new E: handler has been loaded. The second part is our new (and temporary) E: handler routine. It passes back a command line to BASIC without waiting for keyboard input, and then resets the E: handler vector to its original value. The last part is the BASIC command line itself. This can be anything that you could type on one line from the keyboard. You can change screen margins, change screen colors, play music, generate a graphics display, load one program and enter another (and then run them), set up password security, call DOS—the list is endless and limited only by your imagination.

Getting started

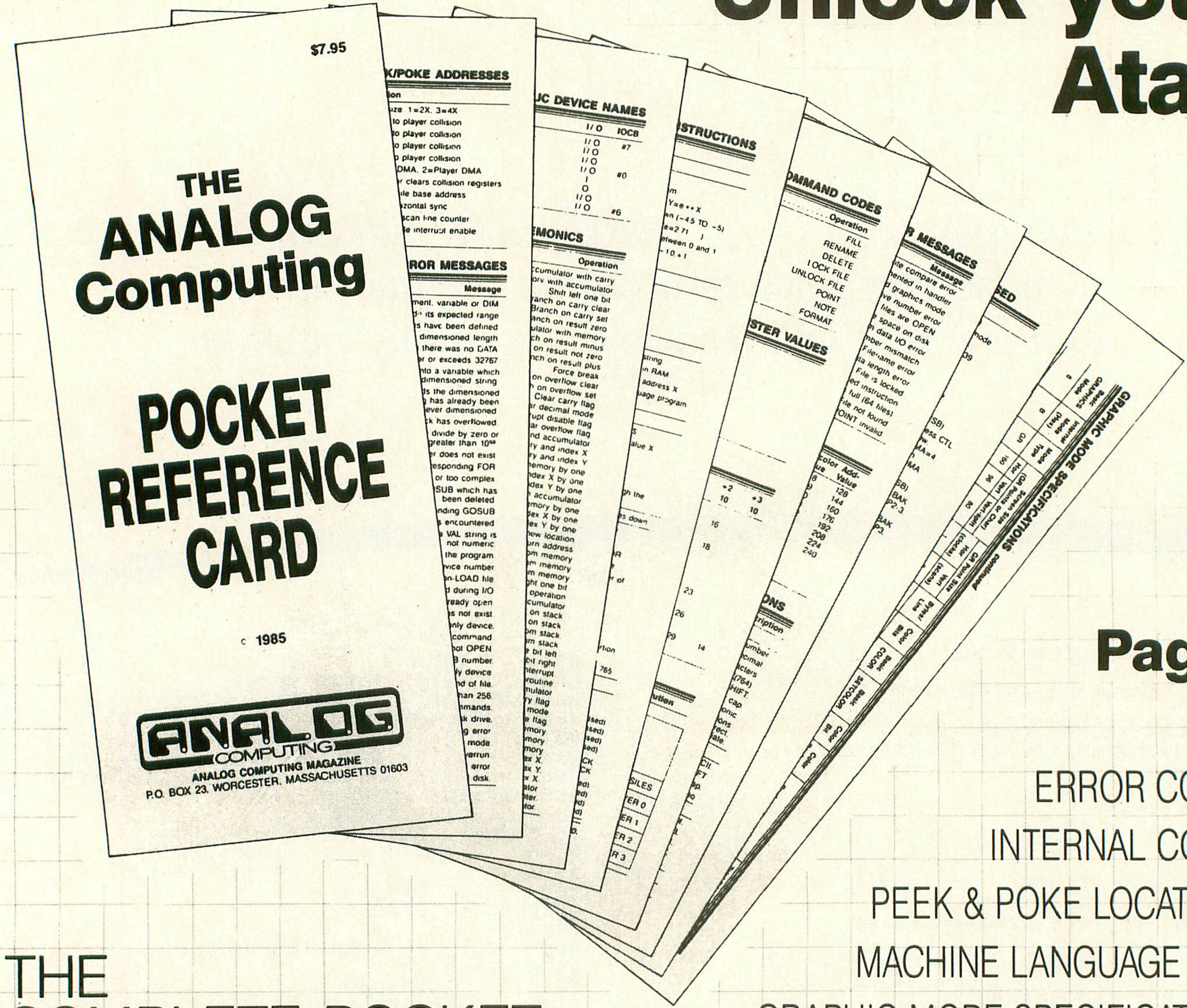
Using MAC/65 or the Atari Assembler Editor, type in Listing 1 and save it. (You'll want to use it again many times.) Note that you can just list Line 10 and edit off the line number and semicolon to cause the program to be listed to disk. Next, change CMDLIN to reflect the BASIC command line you want executed. In Listing 1, the command line is: ? "Loading. . .MYPROG":RUN D:MY PROG.BAS." To insert the quote mark into the BASIC command line, it must be specified by its ATASCII value ($22) as a separate byte. Your command line can be a maximum of 119 characters.

To write the AUTORUN.SYS file, load the destination disk into your drive and assemble Listing 1 to disk with the command: ASM,,#D:AUTORUN.SYS.

I've seen a lot of AUTORUN.SYS makers, but none that have resolved all the questions and few that will let you autorun any program but MENU.

```
10 ;LIST #D:AUTOBAS.SRC
20 ;
30 ;for creating AUTORUN.SYS
40 ;
50 ;MAC65 source code with
60 ;conversions to Atari Assembler
65 ;Editor
70 ;
80 ;Equates:
90 HATABS =   $031A
0100 TEMP =   $CB
0110 ;
0120      *=  $4000     ;or anywhere
0130 ;
0140 ;Modify the Handler table
0150 MAIN
0160      LDX #36       ;search from END
0165 ;                   of table
0170 ELOOP LDA HATABS,X
0180      CMP #'E       ;for 'E:' handler
0190      BEQ CHANGE
0200      DEX
0210      DEX
0220      DEX
0230      BPL ELOOP
0240 CHANGE ;           the table address
0250      INX
0260      STX EDEX     ;save HATABS loc
0270      LDA HATABS,X ;and E: vector
0280      STA TEMP
0290      LDA # <NEWTAB ;or NEWTAB&$FF
0300      STA HATABS,X
0310      INX
0320      LDA HATABS,X
0330      STA TEMP+1
0340      LDA # >NEWTAB ;or NEWTAB/256
0350      STA HATABS,X
0360 ;now transfer ROM table to RAM
0370      LDY #$00
0380      STY YSAV
0390 XLOOP LDA (TEMP),Y
0400      STA NEWTAB,Y
0410      INY
0420      CPY #$10      ;16 BYTES
0430      BCC XLOOP     ;branch if <16
0440 ;now setup new getbyte routine
0450      LDA # <NEWGET-1
0455      ;             or (NEWGET-1)&$FF
0460      STA GETBYTE
0470      LDA # >NEWGET-1
0475      ;             or (NEWGET-1)/256
0480      STA GETBYTE+1
0490      RTS
0500 ;Handler table space
0510 NEWTAB
0520 OPEN .WORD 0       ;see Atari OS
0530 CLOSE .WORD 0      ;Manual,
0540 GETBYTE .WORD 0 ;DeRe Atari, or
0550 PUTBYTE .WORD 0 ;Mappng the Atari
0560 STATUS .WORD 0
0570 SPECIAL .WORD 0
0580 JUMP .BYTE 0,0,0
0590      .BYTE 0,0     ;16th byte
0595      ;             & insurance
0600 YSAV .BYTE 0
0610 EDEX .BYTE 0
0620 ;
0629 ;Our new GETBYTE routine
0630 NEWGET
0640      LDY YSAV
0650      LDA CMDLIN,Y ;get 1 char
0660      CMP #$9B      ;if C/R then done
0670      BEQ DONE
0680      INC YSAV      ;indx next char
0690      LDY #$01      ;tell O.S. OK
0700      RTS
0710 DONE
0720      PHA           ;save C/R
0730      TXA           ;save X register
0740      PHA
0750      LDX EDEX      ;find 'E:' entry
0760      LDA TEMP      ;in HATABS
0770      STA HATABS,X ;replace our
0775      ;             routine
0780      INX           ;with the real
0785      ;             vector
0790      LDA TEMP+1
0800      STA HATABS,X
0810      PLA           ;restore X reg
0820      TAX
0830      PLA           ;restore C/R to A
0840      LDY #$01      ;set status OK
0850      RTS
0860 ;
0870 CMDLIN ;           passed to BASIC
0880      .BYTE "? ",$22,"Loading... MY
PROG",$22
0890      .BYTE ":RUN ",$22,"D:myprog.b
as",$22
0900      .BYTE $9B     ; C/R!!
0910 ;
0920 ;set to execute when loaded
0930      *=  $02E2
0940      .WORD MAIN
```

Unlock your Atari


$7.95
THE ANALOG Computing POCKET REFERENCE CARD
© 1985
ANALOG COMPUTING
ANALOG COMPUTING MAGAZINE
P.O. BOX 23, WORCESTER, MASSACHUSETTS 01603
GRAPHIC MODE SPECIFICATIONS continued

16 Pages

ERROR CODES
INTERNAL CODES
PEEK & POKE LOCATIONS
MACHINE LANGUAGE AIDS
GRAPHIC MODE SPECIFICATIONS
BASIC COMMANDS WITH ABBREVIATIONS

ANALOG COMPUTING

THE COMPLETE POCKET PROGRAMMING AID

ONLY $7.95 ea.

THE #1 MAGAZINE FOR ATARI COMPUTER OWNERS
ANALOG COMPUTING™

P.O. Box 16927
N. Hollywood, CA 91615

YES! Please send me ________ **ANALOG Computing Pocket Reference Cards**.
I am enclosing $7.95 per copy.

☐ CASH ☐ CHECK ☐ CHARGE

Name ______________________________

Address ______________________________

City ________________ State ____ Zip ______

Card # ______________________________

Exp. date ______________________________

Signature ______________________________

BOOT UP TO BIG SAVINGS!

1 YEAR FOR ONLY $28

SAVE $14 OFF THE COVER PRICE

1 YEAR *WITH DISK* ONLY $105

SAVE TIME AND MONEY SUBSCRIBE TO ST-LOG

SAVE $14 OFF THE COVER PRICE WITH THE CONVENIENCE OF HAVING ST-LOG DELIVERED DIRECTLY TO YOUR DOOR *BEFORE* IT EVEN HITS THE NEWSSTANDS! GET THE MOST OUT OF YOUR COMPUTER

SUBSCRIBE TO ST-LOG TODAY!

☐ 1 YEAR @ $28 — SAVE $14! MCKYW
FOREIGN — ADD $7 PER YEAR
☐ 1 YEAR WITH DISK @ $79 DCKYW
FOREIGN — ADD $15 PER YEAR

ST LOG The Atari ST Monthly Magazine

☐ PAYMENT ENCLOSED ☐ BILL ME

CHARGE MY: ☐ VISA ☐ MC #____________________

EXPIRATION DATE __________ SIGNATURE __________

MONEY BACK ON ALL UNUSED PORTIONS OF SUBSCRIPTIONS IF NOT SATISFIED.

NAME ____________________

ADDRESS ____________________

CITY __________ STATE __________ ZIP __________

MAKE CHECK PAYABLE TO L.F.P., INC., P.O. Box 16928, N. Hollywood, CA 91615. Offer expires January 25, 1989. Your first issue will arrive in 6 to 8 weeks.

WATCH FOR IT!

B&C ComputerVisions

3257 Kifer Road
Santa Clara, CA 95051
(408) 749-1003

STORE HOURS
TUE - FRI 10am - 6pm
SAT - 10am - 5pm
CLOSED SUN - MON

NEW!! *1050 SUPER ARCHIVER CHIP* $69.95

8-BIT INTEGRATED CIRCUITS

$3.50 EACH OR 3.00 IN QTY OF 10			$12.00 EACH
ASSEN REV A	800 CPU 6502	RAM 6810	ASSEM REV B
BASIC REV A	810 ROM C	POKEY	XE GATE ARRAY
MPU 6507	800 ANTIC	PIA 6502	XL ANTIC
PIA 6532	OS ROMS (499B-599B)		VCS TIA 444
			BASIC REV C
$4.50 EACH OR 4.00 IN QTY OF 10			850 ROM B
1771 FDC	XL CPU 14806	GTIA	FREDDIE
1050 ROM	XL/XE MMU	XL DELAY	XL/XE OS

NEW PRINTED CIRCUIT BOARDS WITH PARTS

800 MAIN WITH CHIPS	10.00	800 10K O/S	10.00
800 16K RAM	10.00	800 POWER	5.00
810 SIDE WITH D/S	15.00	800 CPU W/GTIA	10.00
810 ANALOG	10.00	810 POWER	15.00

JOYSTICKS

ATARI SPACE AGE	14.95	WICO 3-WAY	29.95
ATARI STANDARD (2)	12.00	WICO BAT HNDL	22.00
ATARI TRAK BALL	25.00	NUMERIC KEYPAD	19.95
EPYX 500 JOYSTICK	19.95	ST MOUSE	45.00

MISCELLANEOUS

400/800 POWER PACK	10.00	800 KEYBOARD	40.00
800/810 POWER PACK	15.00	800XL KEYBOARD	25.00
KLM 400/800 POWER PACK	5.00	130XE KEYBOARD	35.00
1030 POWER PACK	10.00	520ST KEYBOARD	75.00
800XL/XE POWER PACK	20.00	1040ST KEYBORD	85.00
2600 POWER PACK	5.00	314/354 POWER	35.00
520ST POWER PACK	50.00	1040 POWER PCB	75.00

DISK DRIVES

ATARI 810	140.00	(130.00)	**ATARI XF551**
ATARI 810 on Plate	100.00	(90.00)	199.95
ATARI 810 W/HAP	220.00	(210.00)	**INDUS GT**
B&C 810	140.00	(120.00)	225.00

PRICES IN BRACKETS DO NOT INCLUDE I/O OR POWER PACK

DE RE ATARI $10.00

FIELD SERVICE MANUALS

ATARI	400/800, 810	25.00 EACH
ATARI	800XL, 850, 1025, 1050	20.00 EACH
SAMS	800, 800XL, 130XE, 1050	19.95 EACH
SAMS	520ST	35.00

RUN MAC SOFTWARE ON YOUR ST

MAGIC SAC	119.95
TRANSLATOR	279.95
MAC ROMS	39.95
PRT DRIVERS/FINDER	44.95

MAC ROMS

COMPUTERS & INTERFACE

ATARI 800	100.00	MPP-1150 INTERFACE	54.95
ATARI 800XL	119.95	MPP-1151 INTERFACE	74.95
ATARI 130XE	149.95	ATARI 850	125.00

DIAGNOSTICS

1050 DIAG. DISK	20.00	810/1050 DIAG. CART	25.00
SALT 800XL CART	25.00	SALT 400/800 CART	25.00
5-1/4" ALIGNMENT	40.00	3-1/2" SS ALIGNMENT	50.00

HAPPY ENHANCEMENT VER. 7.1
810 OR 1050 - $99.95

ST INTEGRATED CIRCUITS

DISK CONTROLLER WD1772	25.00	DMA CONTROLLER	26.00
PHOTO COUPLER PC900	2.95	MMU	28.00
YAMAHA SOUND CHIP	10.00	VIDEO SHIFTER	26.00
6850 ACIA	3.95	GLUE CHIP	28.00
68000-8 CPU	27.00	68901 MPF	16.00
KEYBOARD CHIP	15.00		

ATARI XE GAME MACHINE $139.95
INCLUDES MISSILE COMMAND, FLIGHT SIMULATOR II, BUG HUNT, LIGHT GUN, JOYSTICK, BASIC, AND 64K OF MEMORY WITH A REMOVEABLE KEYBOARD. ADD A DISK DRIVE AND PRINTER FOR A COMPLETE HOME COMPUTER SYSTEM!

WE CARRY A FULL LINE OF CARTRIDGES FOR THE XE GAME MACHINE.

SUPER SPECIALS

RECONDITIONED ATARI MERCHANDISE

All merchandise has been tested and reconditioned and is in like-new condition except where noted. 30 day warranty.

ATARI TRAK BALL $9.95
SPICE UP THE ACTION IN YOUR ARCADE GAMES!

ATARI SPACE AGE JOYSTICK $5.00

1020 COLOR PLOTTER/PRINTER $29.95
40 Columns wide
Includes paper and color pen set

600XL (64K) $59.95
Upgraded to run newer 64K software - includes Basic & power supply

ATARI BOOKKEEPER
$14.95 - NO BOX
($19.95 WITH RECON KEYPAD)
$24.95 - IN BOX
(29.95 WITH RECON KEYPAD)

400 (16K) COMPUTER $29.95
48K UPGRADE KIT
$25.00

1030 MODEM WITH EXPRESS! $29.95
GET ONLINE TODAY

800 (48K) COMPUTER $79.95
INCL. BASIC CART & MANUAL

NUMERIC KEYPAD $7.95
INCL HANDLER DISK USE WITH BASIC & BOOKKEEPER

DISKETTES
AS LOW AS 20 CENTS
10 FOR $4.00
100 FOR $29.95
1000 FOR $200
MOST ARE UNNOTCHED WITH OLD SOFTWARE

SHIPPING INFORMATION - Prices do not include shipping and handling. Add $5.00 for small items ($8.00 Min. for Canada). Add $8.00 for disk drive. Calif. res. include 7% sales tax. Mastercard and Visa accepted if your telephone is listed in your local phone directory. Orders may be pre-paid with money order, cashier check, or personal check. Personal checks are held for three weeks before order is processed. C.O.D orders are shipped via UPS and must be paid with cash, cashier check or money order. International and APO orders must be pre-paid with cashier check or money order. $20.00 minimum on all orders. All sales are final - no refunds - prices are subject to change. Phone orders accepted TUESDAY THROUGH FRIDAY from 10:00 am to 6:00 pm PST.

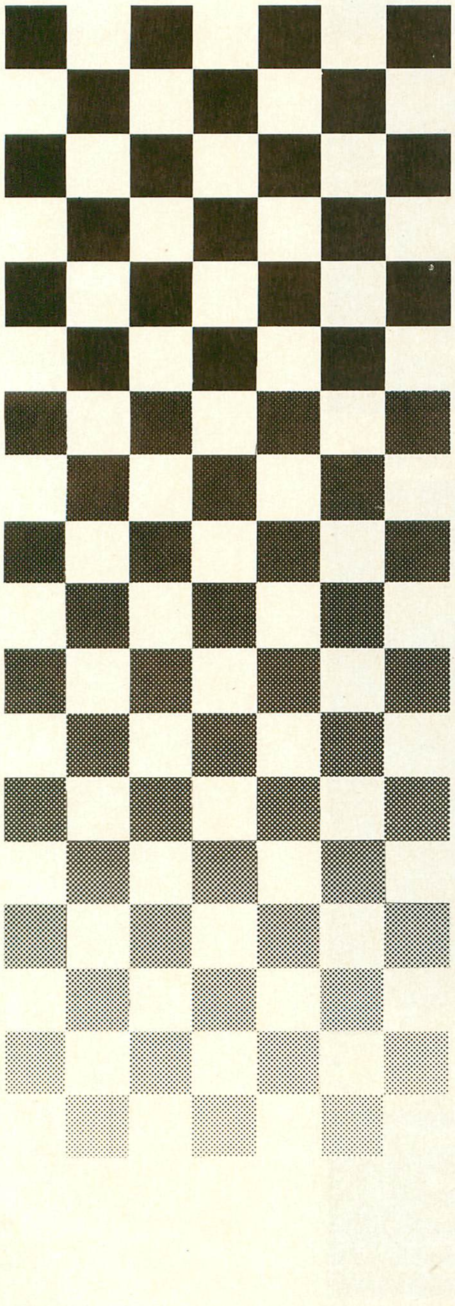
We carry a complete line of ATARI products and have a large public domain library. Write or call for free catalogue. (408) 749-1003 TUE - FRI 10AM - 6 PM

PRICES SUBJECT TO CHANGE WITHOUT NOTICE - ALL SALES ARE FINAL

CIRCLE #104 ON READER SERVICE CARD.

PANAK STRIKES

by Steve Panak



Again we are witness to yet another software market cycle. As you might have noticed by now, an alarming number of software publishers are weaning out Atari 8-bit support. Some new titles, which I'm certain would be good sellers, and most of which probably don't require more memory than the XL/XE series provides, just aren't in our language. It's getting hard to find two games a month to examine for you, much less the four or five I used to be able to play. And though this sounds like the beginning of the end and is extremely distressing to Atari newcomers, there has been an unexpected but welcome side effect.

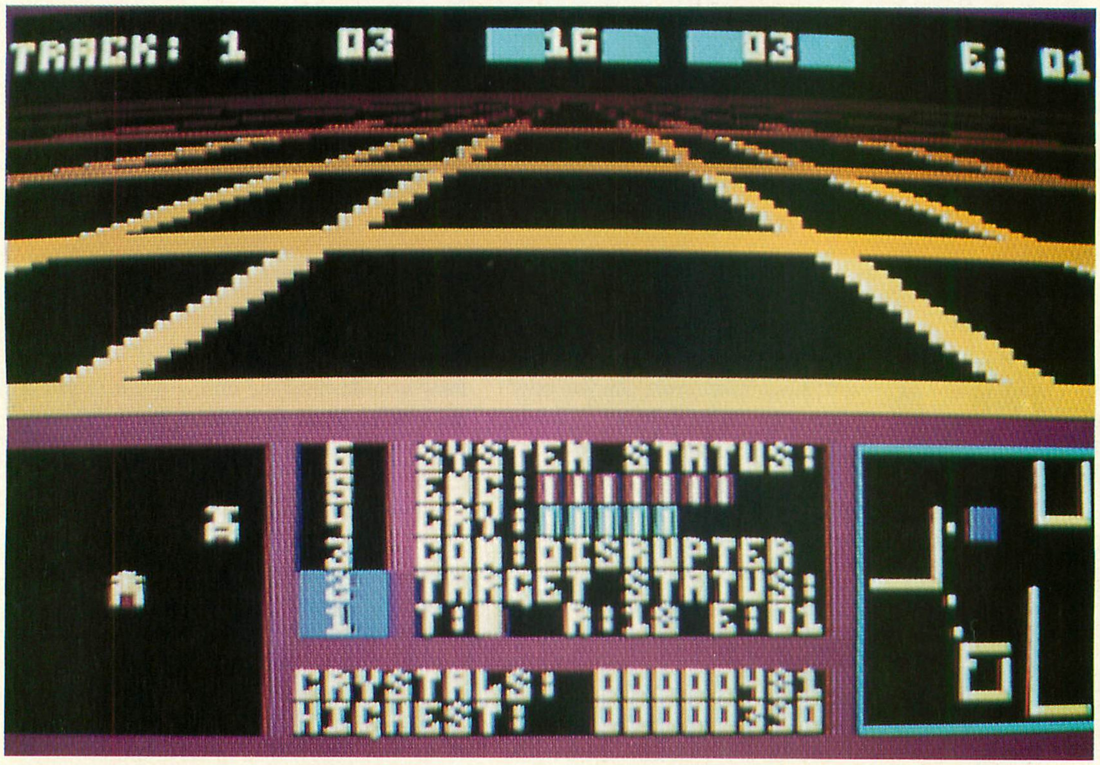
I'm starting to see a lot of independent producers and developers jumping into the water. And while some of their stuff is good, some bad and some mediocre, all of it is sensibly priced. The lack of high marketing, research and development and packaging costs is passed right on to us, the end users. This makes it a lot easier for a novice to stock his library with a lot of games of different types, where a few years ago he might only have been able to afford one or two. To make things even better, the big boys have seen this market niche as well, making us even bigger winners as they discount some of their older games, attempting to milk the few remaining dollars from their cash cows. If you pick and choose wisely, you can't help but be satisfied.

So keeping these two facts in mind, I think what we'll do to kick off the holiday season is to look at some oldies, and a couple of new games from a new company. I'm going to start by bringing you all up to date on a few of the best thinking-man's games available for the 8-bit. Games which require not fast reflexes nor superb hand-eye coordination, but instead ask you to think. And because they demand more of you, I like them the best. For you action addicts, if you'll just hang around until the end, I promise you a photon fix. And if you're not careful, you might even learn something before we're done.

When I think of thinking games, the first to come to mind is chess. Chess has been around for hundreds of years and is still complex enough to challenge the world's greatest minds. And when you feel up to the challenge, your Atari is ready for you, thanks to the *Chessmaster 2000,* from ElectronicArts, which I consider to be the best chess game available. In addition to having one of the most devastating decision-making algorithms available the program is also the most attractive, featuring a three-dimensional playing field which can be modified to your heart's content. Chock full of features, including multiple levels of difficulty, the ability to study the decision-making process and to print a log of moves, as well as all the standard options like move take-back and board setup, this grand master is likely to keep you engaged for months. And if you don't know how to play, Chessmaster is more than happy to teach you. Experts revel in its library of classic games and the brief history of computer chess contained in the generous manual. For all these reasons, and many more, Chessmaster 2000 is the best chess simulation on the market and a must for all game libraries.

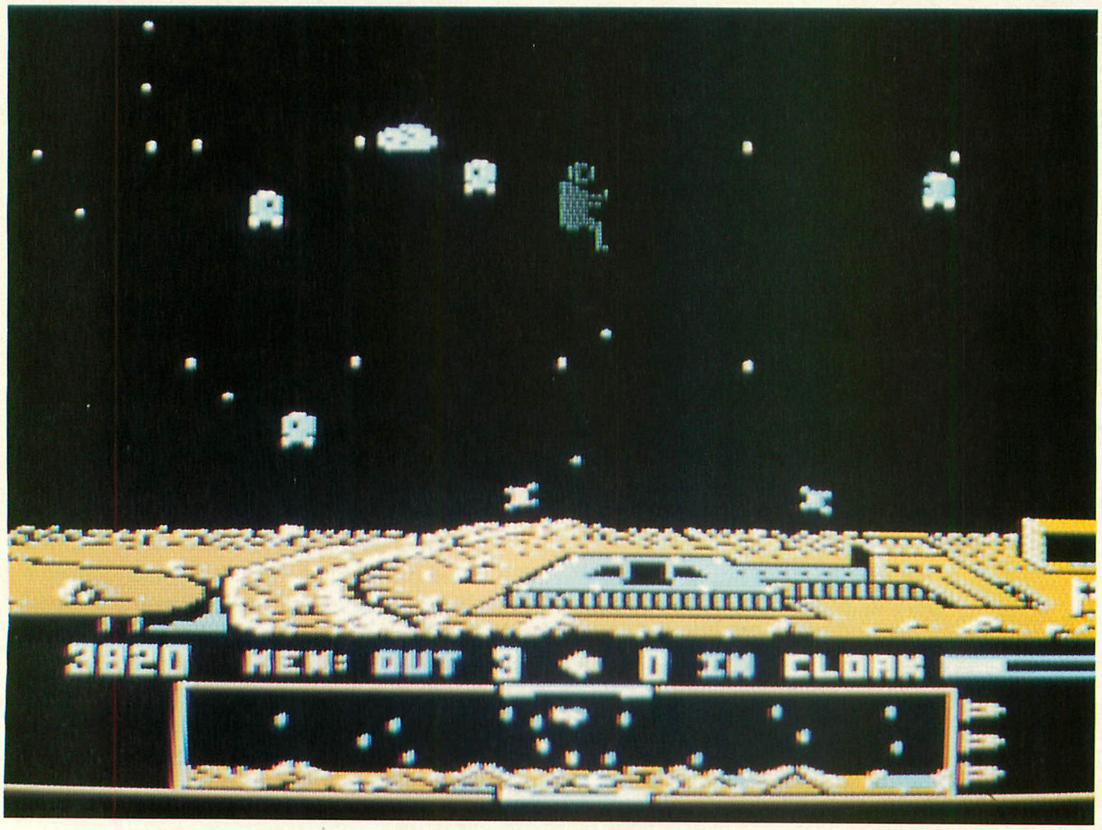
Of course if you're a real masochist, you will probably love any of the many Infocom titles. These text adventures feature the most sophisticated grammatical parser available, and are able to understand complex and compound sentences. In fact, they often demand them. My favorite titles are the *Zork* trilogy, *Stationfall* and *Planetfall*, *Suspended*, *The Hitchhiker's Guide to the Galaxy* and *Leather Goddesses of Phobos*. And although this list indicates that I am partial to their comedy and

PANAK STRIKES



SOLAR STAR ▲

DROP ZONE ▼



This grand master is likely to keep you engaged for months. And if you don't know how to play, Chessmaster is more than happy to teach you.

science-fiction titles, rest assured that, like a bookstore, Infocom has titles to suit everyone's taste. From mysteries to romances to adventures, nearly every computer user who loves a good story will find at least one title to keep them up at night—all night. And to make it even easier on the pocketbook, many of their 8-bit prices have been lowered, some to a mere $9.95. At that price it's pretty hard to go wrong.

If you're into word games, there are a couple of titles out there to confound you. The first which comes to mind is *Buzzword*. This game is very similar to the *Family Feud* television show (minus the obnoxious Richard Dawson). In this game, you choose a category and are then given nine letters which begin nine words. Also provided are the number of letters in each word and a pool of available letters. The object is to guess all the words in the category and amass the most points. Different levels of difficulty make for family fun, as children of all ages can participate. Many find the game is most fun when played in a group setting. There are thousands of words in the hundred or so categories provided in the package—enough to keep the average player active for months. In fact, each category can be replayed a number of times before the words will be memorized. Even though it is a little-known game from a little known company, Buzzword should not be overlooked.

The second word game which comes to mind is *Crosscheck* from Datasoft. This word game reminds me of *Scrabble,* without all the wooden tiles to lose. Upon further examination though, it is a very original and multifaceted game. Up to four players take turns placing words on a crossword-style board. These words are guessed from clues given at the bottom of the screen. The number of letters in the word is determined randomly at the beginning of each player's turn by a roll of the electronic dice. Different versions of the game contain different objectives. You can race to connect two areas of the board or play for points, with or without a time limit. The board is huge with a magnify feature allowing you to zoom in to place your word in the best strategic position. And while some of the clues are simple, additional clue libraries ensure Crosscheck lovers are not left in the dark. Truly a diamond in the rough, and worth a look.

Solar Star
by Glenn Cassim
Drop Zone
by Archer Maclean

Microdaft
19 Harbor Drive
Lake Hopatcong, NJ 07849

Here we are in phase two of this month's discourse. A look at a couple of new games from a new company. The new company is Microdaft (which gets the award for the stupidest name), and the first game is *Drop Zone.* Or should I say the first game is *Defender*, as Drop Zone is a copy of William's arcade classic nearly verbatim; the only difference being that you are represented by a man in a spacesuit, rather than by a sleek star fighter.

In case you didn't catch Defender the first time around, it is your basic space shoot-'em-up, the main twist being the ruthlessness of the objects you encounter. Of course, in Drop Zone the names of all these items have been changed to protect the guilty. Androids and Nemesites, Spores and Nmeyes will plague you as you attempt to rescue your men who are stranded on a planet being overrun by aliens.

Using the joystick, you move up and down, left and right. The space bar releases a smart bomb, which destroys everything on the screen, while any other key activates your cloak, which makes you invisible to the enemy. The only other key you need to worry about is the escape key, which pauses the action when you feel you need a break. There's not a lot to learn about in this game—no great secrets. But what it lacks in complexity it makes up for in sheer difficulty. Each successive wave of enemy onslaught will tax your arcade skills to their limits.

The graphics are surprisingly good, given I've come to expect so little from 8-bit games of late. The action moves every bit as fast as any arcade game I've seen, commercial or otherwise, and the joystick is very responsive. The effect as you explode is especially impressive. The slight manual explains each control succinctly, making Drop Zone a pleasant surprise.

Solar Star is a little harder to describe, as it is one of those rare video games which was not inspired by or was not a rip-off of someone else's idea. And as it is new and unusual, it is also a little harder to like. It might be described as an electronic game of tag, or it might not.

The background story concerns a giant energy grid which is used to supply spacefaring vessels with solar fuel. Unfortunately, the computers which were designed to protect the grid began to do their job a little too well, depriving ships of much-needed fuel. Your mission is to gather as many energy crystals as possible, blasting disrupters (the enemy) until they release a crystal. It's basically a race against time to complete the mission before you exhaust your energy.

The screen is divided into four areas. The top half of the display contains a first-person point of view as you move about a large grid which is divided into areas by force fields. Contact with these fields reduces your energy, and when you're out of energy, your game is over. The bottom right- and left-hand corners contain overhead views of the grid, in increasing degrees of magnification. Centered in the bottom of the screen is a readout area to keep you abreast of the various game parameters, such as score and speed. This is a lot to keep track of, and what made it more confusing was the fact that I used one of the bottom displays more than the large main view.

Collect enough crystals, and you'll move on to the next level. The graphics are pretty standard stuff, about at the level of the 2600—blocky but fast-moving. The manual is fraught with misspelled words, but otherwise accurate. I didn't really like this one as much as Drop Zone, and I can't recommend it. But I do praise it for its originality.

That's about it for this month. Next month we'll take a look at the latest SSI simulation and evaluate it up against a new simulation from Datasoft.

Until then, happy holidays.

ST-Notes

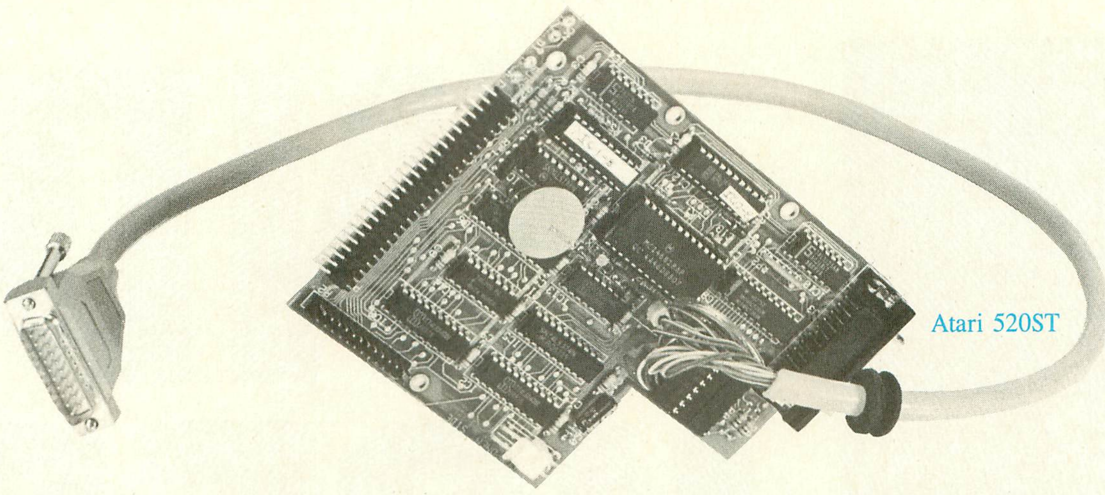
The hard-disk port on the back of your ST is really an amazing, and mostly overlooked, addition to the ST computer.

When the 520ST was first released, many of today's users were satisfying their computing needs with an Atari 130XE or even an old Atari 800. This was 1985, when inflation was at its lowest point in years, and the economy seemed ready for something new. The Atari 520ST was introduced, and for the moment it seemed to be the hottest, newest, sleekest microcomputer on the block. It had more memory, terrific graphics and a hard-disk port built in!

The hard-disk port on the back of your ST is really an amazing, and mostly overlooked, addition to the ST computer. If you own an IBM PC or an MS-DOS clone, adding a hard disk can become fairly difficult. There is no standard interface for hard disks on the PC, nor is there one for the Macintosh. So every hard-disk manufacturer has to determine their own product specifications, and this has led to a market filled with 101 varieties of hard disks. Some hard disks come on an IBM expansion card, some have cables that connect to the PC's mother board, and some plug into your floppy-disk port and sit above your normal floppy. They all look a little differently, sound a little different and operate a little differently.

The ST, on the other hand, has one DMA port which is easily used as the standard hard-disk interface. Other people have used this port to connect optical scanners, laser printers and local area networks. Since the DMA port is included on all STs—Mega STs use the same port—all of the hardware manufacturer's products have been designed to work with one another. The issue of adding a hard disk to an Atari ST could hardly be simpler.

When the ST was first announced at the winter Consumer Electronics Show in 1985, there were some rumors that Atari was working with Haba Systems, a computer software company in Los Angeles, California, that produced Commodore 64 and IBM PC software. Haba intended to produce a hard-disk



drive for the ST that would be bundled with some software and a 520ST. The complete package would be sold as a complete ST development system for developers. Haba was also putting together a series of classes on GEM programming for the ST. A turbulent year for Atari Corp. was 1985, and because of one thing or another, Haba's only ST product became their ST hard-disk drive.

The Haba 10 was a ten-megabyte hard disk for the ST that sold for less than $600. Haba received a large amount of publicity for its announced hard disk mostly because Atari's more expensive 20-megabyte hard disk was the only other alternative.

Haba sold a bunch of hard disks, but then the problems started to flood in. Haba's engineers didn't expect the ST's DMA port to be so electronically unstable. Many 520 and 1040STs wouldn't work with Haba's hard disk. Haba eventually created an improved hard-disk interface which worked with all ST computers.

Then many users found the Haba hard-disk operating-system software had bugs. Certain word processors would lose characters within a document file when the text was saved and loaded onto the Haba hard disk. The software problems were never corrected, mostly because Haba was about to go out of business by the time the problems were identified.

Supra Corp. began offering hard disks for the ST early on, but their prices were close to Atari's prices. Supra held the edge on the hard-disk market with several features not offered with the Atari or Haba disks. Supra offered superior operating-system software; users could create more partitions than the Atari hard-disk software allowed. Supra also found Atari's hard-disk boot software in TOS, which allowed the Supra's hard disks to autoboot when the ST was first switched on.

Other manufacturers began appearing for the ST. Astra, Calcom and others began offering well-built, low-cost hard disks for the ST. In 1987, the price of a hard disk was down to about $500 for 20 megabytes. Some companies even offered their interface cards for less than $150. With an interface card, any SCSI (Small Computer Serial Interface) compatible hard disk could be attached to the ST's DMA port. SCSI hard-disk drives were being sold in mail-order houses for less than $250, so if you were not satisfied with any of the commercial offerings, you could piece together the components and make your own hard-disk unit.

Supra has just recently upgraded their hard-disk interface card to support several new features. The new card has a calendar chip which can update the ST system clock every time the system is switched on. Also included is an additional DMA output port. The second port becomes important for the Atari laser-printer (SLM804) owners who must plug their laser printer into the DMA port to print documents. The Supra DMA port lets you "daisychain" many DMA devices. The new Supra card is fully SCSI compatible, so any advertised SCSI drive should work with it.

ST-Log built its own hard-disk system using a Supra interface card. We started with some basic components:

Atari 520ST
(with one-megabyte memory upgrade)
Supra Hard Disk Interface card
Seagate 20-megabyte hard disk
(with controller built in)
Power supply
Hard-disk enclosure

The Seagate 20 hard disk costs $250. It came with an integrated controller, which was mounted on top of the hard-disk unit. A flat ribbon cable extended from the controller; it attached to the Supra interface card. The hard disk, controller and interface card were mounted into the front of a Haba hard-disk enclosure. The nice thing about using the Haba enclosure was the easy mounting of the hard disk, and the built-in power supply provided the +5 and +12 volts D.C. power that the other components needed. These kind of enclosures are readily available from most mail-order houses.

A special power cable was created to supply the Supra interface card. The parts to build the cable were found at a local Radio Shack store. Total cost was around $10 in parts.

A cable extends off of the Supra interface card that attaches to the ST's DMA port directly. When the power to the drive was switched on, we immediately found that the controller ribbon connector was plugged in upside-down. With a quick switch of connectors, we started the 520ST and the GEM Desktop appeared instantly.

When you purchase the Supra interface card, the kit comes with all the disk operating software and a small (26-page) manual. The disk operating system is completely GEM based, so it is easy to use and somewhat intuitive. The software and interface card are somewhat sophisticated; when the formatting utility was started, the program indicated the correct type of hard-disk controller and drive being used. A couple of mouse clicks later, the formatting program was off and running.

A special partition control program was later used to determine the number of disk-drive partitions that would be used. Supra's unique software permits up to 12 partitions to be created. A partition appears on the GEM Desktop as a separate disk icon, so sorting your software into groups becomes easy.

The entire process of building a hard-disk system and installing the system software took less than three hours. The project cost was about $475. ST-Log does not recommend that you try this yourself, because a fair amount of technical knowledge is required. However, with the modularity and easy availability of all of the parts, it is amazing that more people don't try it!

END USER

by Arthur Leyenberger

I'm late again. My deadline for this month's column has come and gone two days ago. I was all set to sit down tonight and write the column when I decided to log onto DELPHI and check out the latest Atari news and gossip. What a mistake.

Mind you, it wasn't bad. It was, well, addictive. I've been doing a lot a traveling lately, so I had fallen behind on the ANALOG and ST-Log Atari SIGs. I signed on to DELPHI and have just spent the last two hours reading messages in the Forum section of the ST-Log group.

If you want to know what is happening within the Atari community, talk with representatives from Atari and other hardware/software companies; DELPHI is the place to be. You can talk directly with the editors and contributors of ANALOG and ST-Log. And best of all, other Atari users are available to answer questions or share your opinions with.

Maybe you knew that already. If not, call DELPHI at (617) 491-3393 and find out how you can get a user login and begin to take advantage of what is available on-line. Of course there is more to the DELPHI service than just the Forum. Files are available for downloading, electronic mail can be sent and received, and there is always plenty of excitement waiting for you in the ANALOG and ST-Log groups.

A quickie

It has happened to me and I'm sure it has happened to you. You read an ad about a new piece of software in ANALOG or perhaps a rave review about a product that you have been waiting for. So you rush out and buy it.

Once you get it home, you tear open the package, shove the disk in the drive and boot the machine. As the program is loading, you glance at the owner's manual, and there you see it: "[The software company] makes no expressed or implied warranty with respect to the program's quality, performance or fitness for use." Then it goes on for another couple of paragraphs that only a lawyer can understand.

Most reputable software companies only warrant the physical media (the disk) and will replace it if defective. But what if the program isn't so great? What if it is so difficult to use that you would never subject your worst enemy to its use? Or what if the program just doesn't fit your needs? What do you do? Who ya gonna call? Program Busters? No. You're stuck. No two ways about it.

I use a MS-DOS computer quite a bit and I recently saw an advertisement in a magazine that amazed me. The ad was for a product called *Excel*, a spreadsheet program that runs on a PC. The company is Microsoft, the largest PC software publisher.

The ad was titled "The Microsoft Excel Win-Win Guarantee." It read, in part, "If you find a spreadsheet you like better between

If you want to know what is happening within the Atari community, talk with representatives from Atari and other companies; DELPHI is the place to be.

now and January 31, 1990, we'll give you your money back. No questions asked." Whew! Can you believe that? I have *never* seen anything like it.

Here is a company that not only stands behind their product, but goes as far as guaranteeing that you'll like it. They guarantee that Excel will meet your needs. They want you to be satisfied with the program or you get your dollars back—without a hassle and for the next year and a half.

Now, I normally don't write about MS-DOS programs in ANALOG. Neither does anyone else for that matter. But I just had to share this with you. Microsoft is to be congratulated for having a software policy the way it ought to be. Can you imagine if Atari had a policy like that? Can you?

8-bit software

It's no secret that new 8-bit software is becoming more difficult to find these days. The fact is, very few companies are publishing new titles for the Atari computer. Even many of the "big" companies that have supported the 8-bit machine in the past, such as Electronic Arts, Batteries Included, OSS, Datasoft and Synapse, have either gone out of business or have had their more popular titles bought by another company.

I just received a new catalog in the mail from ICD. ICD has been around since 1984 and has had many innovative products for the 8-bit computer. In January of this year, ICD bought the OSS product line and continues to publish OSS products and provide support for them. Following is a brief look at some of the current ICD family of products.

The *P:R: Connection* is a flexible, compact and more economical alternative to the Atari 850 interface. It plugs into the disk drive (serial) port of any 8-bit Atari computer and provides two RS-232 serial ports and one centronics parallel port. It takes its power from the computer, and its serial ports possess the same signals and functions as the 850 Interface, including the R: handler. The P:R: Connection sells for $90.

The $60 *Printer Connection* provides a centronics parallel capability in a very small package. One end of the ten-foot cable plugs into any 8-bit computer (the 1200XL requires a slight modification) and the other end plugs into a parallel printer jack. No external power supply is necessary.

If you want to expand the memory on your 800XL or 1200XL to 256K, the ICD *Rambo XL* is what you need. This $40 upgrade board (DRAM chips are extra) not only makes your 800XL or 1200XL a 256K computer but also makes the memory compatible with that of the 130XE. This lets you take advantage of software that can use the extra memory as well as allow you to use a 128K RAMDisk. You'll need to be familiar with soldering to install this upgrade.

For advanced users, ICD offers the *Multi I/O.* This product features five functions in one box. It provides RS-232, parallel and hard-disk interfaces for your computer. It also gives you a either a 256K or one megabyte RAMDisk, of which any amount can be used as a print spooler. The former sells for $240 and the latter $470.

If you have an Atari 1050 Disk Drive and want to upgrade it, the *US Doubler* chip set will do the job. The $40 upgrade will give your 1050 true double-density capability for greater storage, 180K per disk. Once upgraded, the drive will be compatible with single-density (90K) disks and the dual-density (130K) disks. When used with *SpartaDOS,* the US Doubler will also triple the I/O speed of your computer and disk drive.

The SpartaDOS Construction Set is ICD's own DOS that is compatible with just about any disk drive you can use with your 8-bit Atari. It supports single, dual and double density, 40- and 80-track 5¼-inch drives and eight-inch drives with the Percom or ATR8000. It supports the 360K Atari XF551 drive and hard disks. The $40 program also provides date/time stamping of files, subdirectories, a menu-oriented program for rapid file copying and erasing and more. I have been using SpartaDOS for several years and have found it to be the best DOS available for the Atari 8-bit computer.

SpartaDOS X is a cartridge-based DOS that includes all of the features of SpartaDOS and more. This $80 cartridge features multi-file operations, high-speed I/O with US Doubler, Indus GT and Atari XF551 drives, the use of batch files and more. In addition, you can piggyback another cartridge on top of the SpartaDOS X and operate just as if you had booted from disk, except much faster.

The *R-Time 8* cartridge has a built-in battery that provides continuous and automatic date/time stamping of your files. It too is a piggyback cartridge that permits you to use another cartridge at the same time. When used with SpartaDOS, the R-Time 8 works automatically, tagging each file you create with the correct time and date. It sells for $70.

ICD is an excellent company, and I can highly recommend any of their products. If you would like more information about ICD products, contact them at: ICD Inc., 1220 Rock St., Rockford, IL 61101; or call them at (815) 968-2228. Be sure to request their catalog which, incidentally, looks as classy as their products.

Rumors

What fun is it reading an "End User" column without a couple of rumors? I'm not one to start any rumors, but I am usually more than happy to pass them on. And with the way Atari both announces products before their time and also plays things close to the vest, rumors are never far away.

The latest "hot" rumor concerns a supposed ST game machine. That's right, an *ST game machine.* I'll call it, for lack of a better name, the Atari STGS (not very original, I know). Here are the "facts"(?).

Since Atari is doing a "land office" business in video games and video-game systems, it seems only natural for them to come out with a game system based upon the 68000 microprocessor. This is the very same processor used in the ST, Commodore Amiga and Apple MacIntosh.

What will the STGS look like? A moment's consideration leads one to suspect that it will be roughly about the size of the 7800 game machine. It will probably not have a keyboard, although it may have an interface for a keyboard and disk drive. It will use cartridges for the game software. The price? How about under $200? A 68000-based game machine is really not a wacky idea. If this product were true, it would be the first 68000-based video game. One of the potential roadblocks for this product, I think, is the need for at least a dozen game carts to be available for it upon its introduction. Further, other companies besides Atari must develop games for the STGS for it to be a success.

This may not be a problem, though. In the last year, Atari has done an amazing job of licensing popular arcade video games, as well as working with third-party game developers for new game titles. Also consider the announcement at the Summer CES that Nolan Bushnell and company will be developing new games for "Atari video-game machines." But on the other hand, an STGS would require DRAM chips that are currently in short supply. If no new supply for these chips is found, the production of ST computers may suffer as the video-game machines are made. Further, if a STGS were made, *and* it became a big hit, what would that mean to the "game image" that Atari has earned? Is Atari a game company or is Atari a computer company? Can it be both? Has it been both?

All in all, a very interesting rumor.

Here's another one: Atari will introduce a laptop version of the ST with a built-in hard disk. This rumor is a little tougher to swallow than the STGS. The last time Atari displayed a portable computer, it was an 8-bitter, with a six-inch (I believe) monochrome, 40-character screen. I think it was called the XEP and was shown at the first CES after the Tramiels took over Atari.

Anyway, a portable ST may be a neat idea for some of us hard-core Atarians, but I don't think it would be a real challenge for the likes of Toshiba, Zenith or NEC.

Here's an "oldie but goodie": an Atari CD-ROM player by year's end. Remember the "under $500" CD-ROM player Atari showed three years ago? The time was not right then, but maybe it is now. There's also some talk of an Atari 80286 PC clone. We'll see.

At the summer CES, Atari's booth was all video games. There was hardly a computer in sight, not counting the XEGS. But several independent sources, both inside and outside of Atari, were talking about what Atari would be doing at the fall COMDEX (Computer Dealers Exposition) in Las Vegas.

The talk centered on space. No, not the announcement of a manned mission to Mars being controlled by Atari ABAQ computers: floor space! It was said that Atari has some 20,000 square feet of exhibit space reserved at the upcoming COMDEX. With that much space at a computer trade show, Atari may be planning to announce all of the above products and a dozen more. Stay tuned for what may prove to be the biggest Atari show yet seen.

Remember, these are just rumors. They may or may not be true. Chances are that, like most rumors, they are based on fact, but the final outcome will be somewhat different than stated here. However, keep in mind that if we didn't care so much about Atari computing, we wouldn't care so much about Atari rumors.

Keep on computing. See you next month.

ATARI 8-BIT POWER

ALPHA SYSTEMS is constantly innovating to provide more power for your 8-bit Ataris

NEW **HACK PACK**
Special OFFER

The Alpha Systems HACK PACK contains all our finest products for making Back-up copies, Analyzing, Understanding and Protecting your Atari programs. It comes complete with Atari Protection Techniques (Book and Disk I), Advanced Protection Techniques (Book and Disk II), The Chipmunk, The Scanalyzer, The Impersonator and Disk Pack 1000. Worth over $150. Get them all for the special price of **Just $99.95**

Atari Software Protection Techniques Vol I & II

These Book and Disk packages detail the most advanced copy protection methods in use today. They guide you through the methods used to create the protection as well as the copying techniques to get around them. They include information on Phreaking • Hacking • On-line security • Black boxes • Self-destructing programs • Pirate bulletin board systems • Logic bombs • New piracy laws • Hardware data keys • Weak sectoring (Phantom, Fuzzy and unstable sectors) • Overfilled tracks • CRC errors • Bank Select cartridges and MUCH, MUCH MORE. The disks include automatic program protectors, Protection Scanners, directory hiding and more.

BOOK I and DISK I	**$24.95**
BOOK II (Advanced protection) and DISK II	**$24.95**
Special Offer, Order both sets for Only	**$39.95**

CHIPMUNK

Automatic Disk Back-Up System. Make perfectly running unprotected back-up copies of hundreds of the most popular Atari programs. Chipmunk's sophisticated programming Automatically finds and **REMOVES copy protection** from most Atari programs. Back-up even heavily protected programs with ease. Finally, a back-up system that needs no special hardware or skills.
(If you need a full list of what Chipmunk copies, call or write for our free catalog) **$34.95**

Scanalyzer Automatically scan & analyze commercial programs. Unlock programming secrets and learn from the masters **$29.95**

Impersonator Cartridge to Disk back up system. Create running back-up copies of any cartridge (up to 16K) **$29.95**

NEW **CHEAT**

Get more from your games with CHEAT. Tired of spending days trying to beat a game? Tired of getting stuck just when you need another life? Cheat is an innovative new product that gives you the chance you need to beat your favorite games. Cheat works with hundreds of Atari games to give you unlimited lives or power. End the frustration and get hours more enjoyment from your games. (Call or write Alpha Systems for our free catalog with a full list of the programs that work with Cheat) **ONLY $24.95**

BASIC TURBOCHARGER

NOW for the first time a BASIC programmer can get the power, flexibility and incredible speed of machine language. BASIC TURBOCHARGER is a **book and disk package** that contains over 150 ready to use machine language routines. Complete instructions show how to add them to your own BASIC programs to get these features and more: • Smooth Scrolling • Player/Missile control • Load & Save Picture files • Sorting and Searching • Special Effects Graphics • Incredible Speed • Much, Much More • Over 150 programs. You've heard of the power of Assembler, now harness it for your own needs. **$24.95**

NEW **PARROT II**

An All New Parrot sound digitizer for your Atari. Parrot II is a sophisticated new hardware device that plugs into your joystick port. Parrot II has two inputs, One for a microphone and one for a powered source such as a tape player, radio or Compact Disk.

The Powerful Parrot II software lets you record sounds into your computer and play them back on any Atari. Parrot II turns your computers keyboard into a musical instrument with nine different sounds covering three octaves each. The sounds can be anything, a dogs bark, a piano, a complete drum set, a symphony or your own voice.

Parrot II lets you modify the sounds on a graphic display to create brand new sounds and special effects. Best of all, the sounds and voices can be put into your own programs that can be used on any standard Atari. Explore the world of digital sound and music. ONLY **$59.95**

Pre-Recorded Sound Disk More pre-recorded sounds for Parrot	**$4.95**
PARROT II Demo Disk (Does not require Parrot to run)	**$5.00**

NEW **POP-N-ROCKER** a fast paced, multi-player trivia game that mixes questions with real songs (digitized with Parrot). Be the first to identify the songs and answer the music trivia questions. *Pop-N-Rocker* comes with three data disks and lets you add new questions so it will never get old. You can use a Parrot Sound digitizer to add new songs too! Use any kind of music from Rock to Classical to Nursery Rhymes. A new concept in entertainment and a perfect add-on for Parrot. **$24.95**

COMPUTEREYES & MAGNIPRINT II+

Turn your computer into a digital portrait studio. This complete package lets you **capture, save & print** digital images from your **Video Camera, VCR or TV.** COMPUTEREYES hardware plugs directly into your joystick ports for easy use. Print your picture on a 6 foot poster. **$119.95**

GIANT WALL SIZED POSTERS.

ComputerEyes camera system
Comes complete with everything above, plus a black and white video camera and connecting cable. **$329.95**

Graphics 9 Software – Add a new dimension to your COMPUTEREYES pictures – captures images in 16 shades of grey. **$12.00**

Magniprint II+
Easily the most powerful print program available today. Print graphics from almost any format in hundreds of shapes, sizes, and shades. Supports **color printing** and lets you create **giant posters.** Magniprint II+ lets you stretch and squeeze, invert, add text, adjust shading and much more. Works with EPSON, NEC, Citoh, Panasonic, Gemini, Star, XMM801, and compatible printers. (850 interface or equivalent required). **$24.95**

Graphics Transformer
Now you can combine the most powerful features of all your graphics programs. Create print shop icons from a Koala pad picture, from a photo digitized with ComputerEyes, or any picture file. Graphics Transformer lets you **Shrink, Enlarge and Merge** pictures for unequaled flexibility. **$22.95**

YOUR ATARI COMES ALIVE

SAVE MONEY. Finally an alternative to buying expensive computer add-ons. Your Atari Comes Alive shows you how to **built them yourself.** This 'How-To' **book and disk package** gives you complete step by step instructions and programs needed to built and control these exciting devices and MORE: • Light Pen • Light & Motor Controllers •Alarm Systems • Voice Recognition • Environmental Sensors •Data Decoders • More than 150 pages. **Your Atari Comes Alive** **$24.95**

ALPHA SYSTEMS

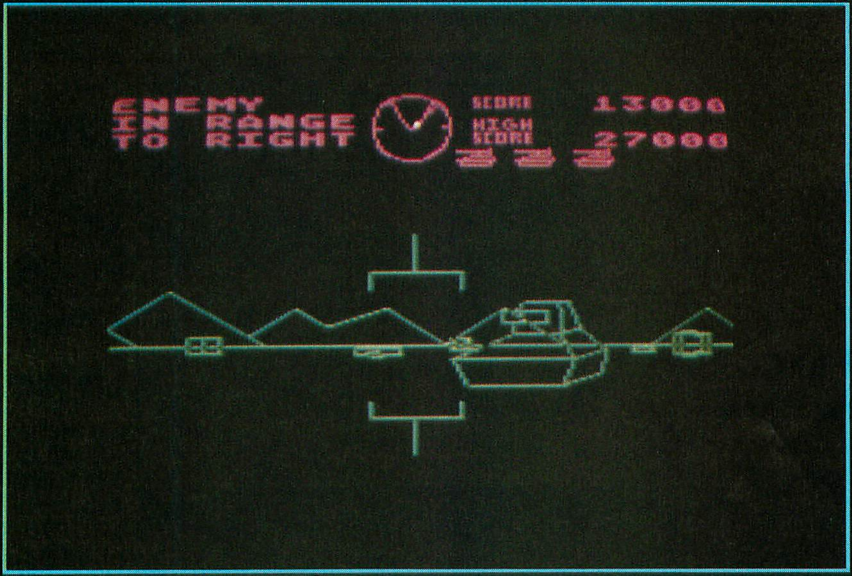
24 HOUR HOTLINE **216-374-7469**

VISA & MASTERCARD, ORDER BY PHONE, OR SEND MONEY ORDER TO:

ALPHA SYSTEMS 1012 SKYLAND DRIVE MACEDONIA, OH 44056 **FREE BONUS:** DELUXE SPACE GAMES (3 games on a disk) Free with any order of 3 or more items. Include $3.00 shp & hdlg (US Canada) Ohio res. add 5 1/2% sales tax. Foreign orders add $8.00 shp & hldg. Call or write for free catalog. Customer Service Line (216) 467-5665 M-F 9-3.

CIRCLE #107 ON READER SERVICE CARD.

R E V I E W



BATTLEZONE

Atari Corp.
1196 Borregas Avenue
Sunnyvale, CA 94086
(408) 745-2000
Cartridge $19.95

reviewed by Howard H. Wen

Regardless of what faults there might be with this game, one thing is for sure: It's a faithful translation of the arcade classic. Unfortunately, this means that if there were things you didn't like about the arcade *Battlezone*, you're going to find them here in this cartridge version for the XE Game System and Atari 8-bit computers.

The scenario takes place in the near future when all nations of the earth have agreed to world peace. However, a "power-hungry rabble of military malcontents" don't really like the idea of living in a world without war. So they do what typical military malcontents would normally do—they send out hordes of robot tanks programmed to destroy the world! Naturally, this is where you come in. Your job is to destroy these tanks before they destroy you, on a battlefield littered with three-dimensional objects—cubes, pyramids and rectangles. And you do this driving a slow-moving tank, which only shoots shells one at a time.

Battlezone is played from a first-person perspective, appearing as if you're looking at the battlefield through the tank's window. The top part of the screen displays an information panel. The left side of the panel tells you of the existence of enemy tanks and if your tank's movement is being hindered by an object. The radar scanner is a circle in the middle of the panel, which shows the overhead, entire view of the battlefield: the wedge-shaped area representing the player's point-of-view. Enemy tanks and other weapons appear as blinking dots on the radar. Finally, the right side of the information panel displays your score and the number of tanks you have remaining.

The object of the game is to simply locate enemy weapons on the radar scanner, move in on them, zero in with your gunsight and blast the target to bits as soon as you see the gunsight narrow on it. You do this by moving your tank around the battlefield with your joystick and firing shells with the joystick button. Of course, the enemy tanks will try shooting back. Plus, there are three-dimensional shapes scattered throughout the field, which may be used for cover from enemy fire, but they tend to be a nuisance by blocking your way.

Besides the slower-moving, normal tanks, you also have to deal with more aggressive supertanks, which move just as fast as you do. And for bonus points, nonattacking saucers move across every now and then. But the toughest of the enemies are the missiles that suddenly drop from the air and quickly zig-zag to collide into your tank.

Battlezone's screen display simulates the three-dimensional, vector graphics of the original arcade version. At first it's difficult to identify objects. Things look especially confusing when an enemy tank hides behind a see-through obstacle. The animation of the kamikaze missiles is erratic, making it extremely hard to shoot them. A nice visual touch added to Battlezone occurs when your tank is hit with an enemy shell, the screen "cracks."

The sound effects are well-done and important to game play—sometimes more so than what you see on the screen. Certain sounds tell whether a normal tank or super-tank is approaching you, warn if a missile is about to appear or alert that a saucer is moving across the field.

Veteran Atarians, who have played similar tank games such as *Dimension X* or *Encounter*, may find Battlezone sluggish in movement and lacking in features. But the slowness of the player's tank is probably deliberate in order to imitate a real tank.

There are five levels to chose from on Battlezone. On Level 1 tanks and other enemy weapons are easily blown away. Level 5 is the most challenging, even for a hard-core video gamer. The game itself doesn't progress from one level to another, but instead, enemy tanks come after you endlessly, one after another. The game ends only when you've lost all of your tanks.

Battlezone will probably satisfy fans of the arcade version and the many new owners of the XE Game System; but Atari 8-bit old-timers might be disappointed. Nevertheless, this one-player game is one of the best, pure shoot-'em-ups to come along for the Atari 8-bits in a long time.

When you want to talk Atari

XL/XE HARDWARE

INTERFACES

ICD

P:R Connection	61.99
Printer Connection	41.99

Supra

1150	39.99
1151 (1200 XL)	40.99

Xetec

Graphix Interface	38.99

Atari

850 Interface	109.00

COMPUTERS

CMO PACKAGE EXCLUSIVE

Atari 800XL & XF551 Disk Drive

w/5 Undocumented ROMS Asteroids, Defender, Missile Command, QIX, Star Raiders

$279

Atari

800XL	89.99
130XE	139.00

XL/XE ENHANCEMENTS

Axlon 32K Mem. Board (400/800)	19.99
Atari 80 Column Card	79.99

MODEMS

Atari

SX212 300/1200 (ST)	89.99
XMM301	42.99

Anchor

VM520 300/1200 ST Dir. Con.	119.00

Avatex

1200 HC	89.99
2400	159.00

Supra

2400 Baud XL/XE or ST	169.00
2400 Baud (no software)	149.00

MONITORS

Magnavox

CM8505 14" Composite/RGB/TTL	199.00

ST HARDWARE

ATARI 520 ST FM RGB/Color System **$789**

Includes: 520 ST FM with 3½" drive, mouse & 1224 color monitor.

SM124 Monochrome Monitor	179.00
SM1224 RGB Color Monitor	329.00

Call For Current Information On The Entire ST Line!

DRIVES

Atari

ST 314 DS/DD	219.00
XF551 Drive (XL/XE)	179.00
SHD204 20 Meg Hard Drive	599.00

Supra 30 Meg Hard Drive **$689**

I.B.

5¼" 40 Track (ST)	219.00
5¼" 80 Track (ST)	279.00

I.C.D.

FA•ST 20 Meg	629.00
FA•ST 30 Meg	869.00
FA•ST Dual Hard Drives	Call

Indus

GTS 100 3½" DS/DD (ST)	199.00
GT 1000 5¼" DS/DD (ST)	209.00
GT Drive (XL/XE)	189.00

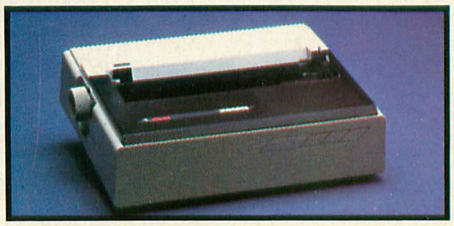
Supra

FD-10 10MB Removable Floppy w/SCSI	899.00
20 Meg Hard Drive (ST)	579.00
20 Meg Hard Drive (XL/XE)	689.00

PRINTERS

Atari

1027 LQ XL/XE	129.00



Atari XDM121 LQ (XL/XE) **$189**

XM-M801 XL/XE Dot Matrix	199.00
XM-M804 ST Dot Matrix	199.00
XDM 121 Letter Qlty. XL/XE	209.00

Brother

M-1109 100 cps Dot Matrix	169.00
M-1509 180 cps Dot Matrix	389.00
HR-20 22 cps Daisywheel	339.00

Citizen

120D 120 cps Dot Matrix	149.00
180D 180 cps Dot Matrix	179.00
Premier-35 35 cps Daisywheel	549.00

Epson

LX-800 150 cps, 80 col	189.00
Hi-80 4 pen plotter	269.00
FX-850 264 cps, 80 col	Call
FX-1050 264 cps, 132 col	Call
LQ-500 180 cps, 24-wire	Call
LQ-850 330 cps, 80 col	Call
LQ-1050 330 cps, 132 col.	New

NEC

P2200 pinwriter 24-wire	379.00
P5200 pinwriter 24-wire	599.00
P5300 pinwriter 132 col	799.00

Okidata

Okimate 20 color printer	129.00
ML-182 + 120 cps, 80 column	229.00
ML-320 + 300 cps, 80 column	379.00
ML-390 + 270 cps, 24-Wire	539.00

Panasonic

KX-P1080i 144 cps, 80 col	169.00
KX-P1091i 194 cps, 80 col	199.00

Star Micronics

NX-1000 140 cps, 80 column	179.00
NX-15 120 cps, 132 column	319.00

Toshiba

P321-SL 216 cps, 24-wire	499.00

SELECT FROM OVER 3000 PRODUCTS

WE SHIP 90% OF ALL ORDERS WITHIN 24 HOURS

CMO

COMPUTER MAIL ORDER

..........you want to talk to us.

XL/XE SOFTWARE

Access	
Leaderboard Golf	13.99
Accolade	
Hardball	19.99
Atari	
Atariwriter Plus	35.99
Filemanager	11.99
Music Painter	11.99

LOOSE/UNBOXED XL/XE ROM CARTS

$3⁴⁹ ea. or 5 for $14⁹⁹

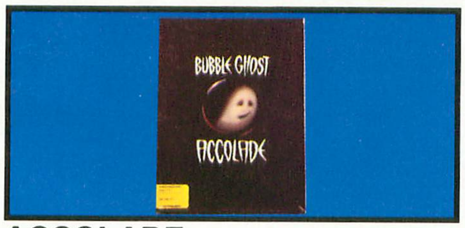
Includes: Space Invaders, Star Raiders, Missile Command, Asteroids, Pac Man, Galaxian, Defender, Qix

Atari Program Exchange	
Misc. Programs (cassettes)	at 1.99
Broderbund	
Graphics Library I, II, III	14.99
Printshop	26.99
Datasoft	
Alternate Reality (City)	23.99
221 Baker St.	20.99
Electronic Arts	
Auto Duel	29.99
Firebird	
Guild of Thieves	19.99
Silicon Dreams	19.99
Jewels of Darkness	19.99
Microprose	
Top Gunner	16.99
F-15 Eagle Strike	21.99
Silent Service	22.99
Origin Systems	
Ultima 4	36.99
Roklyn SPECIAL	
Anti-Sub/Journey to Planet	(ea.) 3.99
Strategic Simulations	
Colonial Conquest	24.99
Gemstone Warrior	11.99
Sublogic	
Scenery Arizona	14.99
X-Lent	
Typesetter	22.99
Printshop Interface	21.99

ACCESSORIES

MD1-M SS/DD 5¼"	8.49
MD2-DM DS/DD 5¼"	8.99
MF-1DDM SS/DD 3½"	11.99
MF2-DDM DS/DD 3½"	18.49
Sony	
MD1D SS/DD 5¼"	6.99
MD2D DS/DD 5¼"	7.99
MFD-1DD SS/DD 3½"	11.99
MFD-2DD DS/DD 3½"	17.99
Allsop Disk Holders	
Disk File 60-5¼"	9.99
Disk File 30-3½"	9.99
Curtis	
Emerald	39.99
Safe Strip	19.99
Universal Printer Stand	14.99
Tool Kit	22.99
ICD	
BBS Express (ST)	52.99
Sparta DOS Construction Set	28.99
US Doubler/Sparta DOS	47.99
Real Time Clock	48.99
Rambo XL	29.99
US Doubler	28.99

ST SOFTWARE

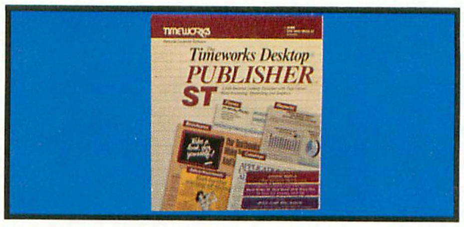


ACCOLADE Bubble Ghost $28⁹⁹

Abacus	
PC Board Designer	119.00
Access	
Leaderboard Golf	22.99
Activision	
Hacker II	28.99
Antic	
CAD 3-D	29.99
Avant Garde	
PC Ditto	59.99
Batteries Included	
Degas Elite	37.99

ST SOFTWARE

Dataeast	
Speed Buggy	24.99
Electronic Arts	
Gridiron Football/Auto Duel. (ea.)	26.99
Isgur Portfolio	119.00
Firebird	
Silicon Dreams	19.99
The Sentry/Tracker (ea.)	12.99
Infocom	
Beyond Zork	34.99
Metacomco	
ISO Pascal	59.99
Microprose	
Gunship	28.99
F-15 Strike/Silent Service (ea.)	24.99
Miles Software	
ST Wars	24.99
Mindscape	
Road Runner	36.99
Mark of the Unicorn	
PC Intercom	79.99
Mark Williams	
C	119.00
Paradox	
Wanderer (3D)	24.99
Progressive Computer	
Graphic Artist 1.5	119.00
Psygnosis	
Barbarian/Deep Space	(ea.) 25.99
Soft Logik Corp.	
Publishing Partner	54.99
Strategic Simulations	
Questron II	37.99
Sublogic	
Flight Simulator II	33.99
Timeworks	
Swiftcalc/Wordwriter	(ea.) 45.99
Partner ST	29.99



TIMEWORKS Desktop Publisher $79⁹⁹

Word Perfect Corp	
Word Perfect 4.1	179.00

In the U.S.A. and in Canada

Call toll-free: 1-800-233-8950

Outside the U.S.A. call 717-327-9575, Fax 717-327-1217
Educational, Governmental and Corporate Organizations call toll-free 1-800-221-4283
CMO, 101 Reighard Ave., Dept. B7, Williamsport, PA 17701

MEMBER MMC MICROCOMPUTER MARKETING COUNCIL of the Direct Marketing Association Inc

OVER 350,000 SATISFIED CUSTOMERS • ALL MAJOR CREDIT CARDS ACCEPTED • CREDIT CARDS ARE NOT CHARGED UNTIL WE SHIP

POLICY: Add 3% (minimum $7.00) shipping and handling. Larger shipments may require additional charges. Personal and company checks require 3 weeks to clear. For faster delivery, use your credit card or send cashier's check or bank money order. Credit cards are not charged until we ship. Pennsylvania residents add 6% sales tax. All prices are U.S.A. prices and are subject to change, and all items are subject to availability. Defective software will be replaced with the same item only. Hardware will be replaced or repaired at our discretion within the terms and limits of the manufacturer's warranty. We cannot guarantee compatibility. All sales are final and returned shipments are subject to a restocking fee. We are not responsible for typographic or photographic errors.

CIRCLE #108 ON READER SERVICE CARD.

B711

Refresh Your Memory

And Keep Your Cool.

Introducing the ST Hard Drive System from ICD that refreshes your memory better than any other ST hard drive around. *No problem.*

It's the drive that not only looks cool, but stays cool too. All because of a built-in fan that knows exactly how to beat the heat and maintain a calm, cool and collected environment . . . even in *your* most heated situations. *No sweat.*

And, it's the hard drive that takes a refreshing approach to aesthetic case design as well. See for yourself. It's easy on the space, fitting perfectly under the monitor. And it's easy on the eyes, tailored to look great in the company of your Atari ST. With adjustable front legs, your monitor gets the lift it needs for comfortable viewing. *No strain.*

Despite a sleek and compact exterior, the ICD ST Hard Drive System is packed full of overwhelming enhancements. Like an internal clock that tags each file with up-to-the-minute time and date information. Not to mention expansion capabilities that welcome the connection of up to six SCSI devices and daisy-chaining Atari's DMA Bus (ACSI). It's available in more memory capacities than you can imagine. With storage ranging from 20 megabyte systems up to 280 megabytes. And, there's dual drives too, that double your protection and double your confidence. *No stress.*

So, the next time you think about a hard drive for your Atari ST, think about the countless ways we can refresh your memory. It's the only drive worth remembering. Because it's from ICD. *No wonder.*

For further product information, please call or write for our catalog today.

ICD

1220 Rock Street
Rockford, IL 61101-1437
(815)968-2228
MODEM: (815)968-2229
FAX: (815)968-6888

Atari ST is a trademark of Atari Corporation.

CIRCLE #101 ON READER SERVICE CARD.