ABCs of Atari Computers

David E. Mentley

The definitive reference source for owners of Atari Home Computers

RUNONLY

ABCs of Atari Computers

ABCs of Atari Computers

By
David E. Mentley

Illustrations by
Peter Wickman

DATAMOST, INC.
20660 Nordhoff Street
Chatsworth, CA 91311-6152
(818) 709-1202

DATAMOST INC.

ISBN 0-88190-367-1

Printed in U.S.A.

DEDICATION

To my mom and dad, with love.

ACKNOWLEDGEMENTS

I would like to thank Joe Decuir for his careful review of this book and historical perspectives on the Atari hardware. Thanks to Roy Wolford and Michael Moore for their thorough review and suggestions. Also, thanks to Pete Goodeve for answering many questions at all hours of the day or night.

FOREWORD

The Atari Home Computer is the most powerful personal computer in its class. Like me, you probably did not know this when you made your purchase. You probably brought your new toy home, plugged it in, and said, "OK computer, entertain me." It probably didn't respond. Computers are not good for anything unless you know how to make them do what you want. This means that you have to provide some instructions. You can buy some software at your local computer retailer, get some public domain software from your local user group, or, for the ultimate exhilaration, write your own programs. The very things that make the Atari computer powerful also make it complex. You have to know the basics of the way the hardware is put together and also the way that the Operating System works if you want to control your machine yourself. And we do want to control the machine. My theory is that the success of home computers and video games is due to the fact that WE CONTROL the television picture instead of just watching it.

What makes the Atari computer so powerful? It was designed over six years ago. This should make the hardware obsolete by today's standards. The new XL line uses basically the same architecture as the original Colleen (800). The secret to the success is flexibility. Try to do a custom display list on a Commodore 64 or an Apple II. Try to make some four voice music on an Apple II. Try some high speed player motion on an Apple II. These features were put in place by the designers who had visions of the potential of a flexible machine. With the microprocessor bus available to be tapped on the new line, I expect to see some add-on peripherals that will allow the Atari line to compete against personal computers priced between $2,000 and $3,000.

This book is designed as a guide to take you from being a beginner to being an intermediate user. I did not see a need for another book on "How to Write BASIC Utilities for the Home" or "2,000 Games for Your Atari." What I did see a need for was a compilation of the tips, tricks, and lore for the Atari computers. As president of a large users' group, I was exposed to a constant flow of fascinating discoveries that probably only 1% of Atari owners could know about. These are usually expressed in the numerous club newsletters which criss-cross the country monthly. Some of these tips are only for the brave. You may not want to open up your computer and start cutting and soldering, but then again, you may. I have included some of my favorite public domain programs. When possible, I have given credit to the author. After all, that is part of the appeal of writing software for the public domain. Some of these have been around so long and have been modified so much that the author is not known. Please let me know if you can help here.

Although this book has consumed a good chunk of my life, and I swore I wouldn't type another word, I would like to keep it up to date. If you have an unusual discovery or a favorite utility or demonstration program which looks appropriate, please send it to me at the address below. You will be given appropriate credit.

If you are a vendor and you think that you have a truly useful product for the Atari community, I would be happy to review your hardware or software.

Happy Hacking,

Dave Mentley
P.O. Box 325
El Cerrito, CA 94530
May, 1984

ABCs of Atari Computers

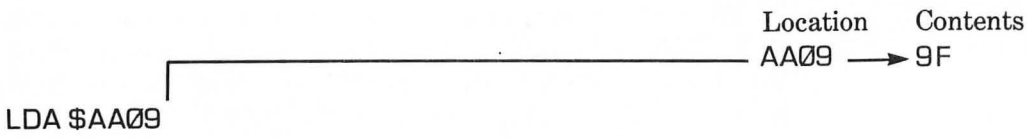
A — Short for ACCUMULATOR in assembly language mnemonics like LDA, STA, PHA, etc. The ACCUMULATOR is the workpiece for the central microprocessor (the 6502 chip) in which addition and bit manipulation is performed.

A — The A command from the Atari DOS 2.0S menu initiates a directory listing of the floppy disks in the drive or drives connected. A directory entry consists of a filename of up to eight characters and an extender of up to three characters. To get a directory of the disk in drive 1, just type: A<Return> <Return>. Drive 1 is the default drive, meaning that you need not specify "D1:" in order to read the directory. To read the directory of drive 2, type: A<Return> D2:<Return>. To print a directory, type: A<Return> ,P:<Return> while the printer is attached and ready to print. To print only directory entries which end in a specific extension, type: A<Return>*.EXT <Return>. The asterisk (*) is a wildcard which can be used to indicate all filenames and extensions.

ABS — ABS is a BASIC command which generates the ABSolute value of a number. This command simply drops the hyphen (-), which represents a negative or minus operator, from a number.

```
EXAMPLE:        ABS(-99)= 99
```

ABSOLUTE ADDRESSING MODE — The 6502 microprocessor can retrieve data in several ways. ABSOLUTE MODE addressing is a technique for specifying the address in the processor's memory space where it is to look for data. ABSOLUTE ADDRESSING is a three byte instruction. The OPCODE is given first, followed by the low order and high order bytes specifying the address which contains the data of interest. Programs written in the ABSOLUTE mode cannot be relocated to other parts of memory. Instructions which can be used in the ABSOLUTE mode are: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, JMP, JSR, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SEC, STA, STX, and STY.



This means: Go to absolute location $AA09, get the data, and put it in the accumulator. This is equivalent to

A=PEEK(43529) in BASIC.

43529 is the decimal equivalent of $AA09.

ACCUMULATOR — The ACCUMULATOR is the register in the 6502 processor in which most of the data modification is done. The ACCUMULATOR acts as the workpiece where numbers are added, compared, shifted, and so on. The data byte in the ACCUMULATOR can be acted upon with an arithmetic operation or the results of an operation can be stored there. Data is fetched from memory and placed in the ACCUMULATOR for further operating. The data must always be put into the ACCUMULATOR before the operation is started. The mnemonic STA takes the value in the ACCUMULATOR and copies it out to a memory address. The LDA mnemonic loads the value of a memory address into the ACCUMULATOR. PLA pulls the contents of the memory pointed at by the stack register (plus 1), and puts this value into the ACCUMULATOR.

ACE — Atari Computer Enthusiasts. This name is recommended for user groups by Atari Users Group Support. ACE of Eugene, Oregon was probably one of the larger groups to adopt this name. An excellent newsletter and many fine public domain disks are available to members. The address is ACE, 3662 Vine Maple Drive, Eugene, OR 97405.

ACTION! — ACTION! is a language from OSS, Inc. available on cartridge. ACTION is a structured language which features a very powerful editor and a fast compiler. It has properties of C, Pascal and ALGOL.

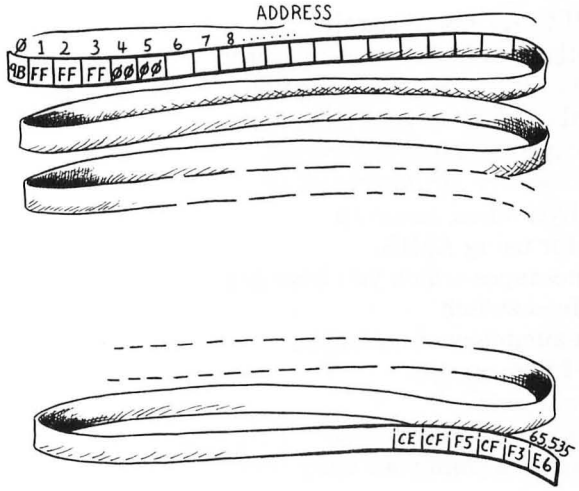
ACOUSTIC COUPLER — One method of connecting a modem to the Atari computer is through an ACOUSTIC COUPLER. The coupler consists of a speaker and a microphone which allows the telephone handset to be used to transfer data. The signal is transmitted by a combination of high and low frequencies which carry the stream of data. An acoustically coupled modem is subject to interference from a stereo or television if the music or volume is too loud. A DIRECT CONNECT modem is not subject to this interference.

AD ASTRA — To the Stars - Atari Microcomputer Net Amateur Radio Operators Group publishes a newsletter containing many tips for the hardware enthusiast, particularly for the HAM radio operator with an Atari computer. Subjects covered in the newsletter include such things as disk drive modifications, interfacing hardware to your Atari, and making your own cables. The NET organization meets on the air on a regular schedule. The National Net is at 14.325 Mhz at 1600Z on Sundays, with WD8BNG running the show. Ad Astr, 4749 S.R. 207 N.E., Washington Court House, OH 43160.

ADR — In Atari BASIC, the ADR statement returns the starting address in memory of a string named in the ADR statement. The string, once created, is always stored somewhere in the computer's memory. The ADR command pinpoints its location. This address can be the starting address for the machine language program. A string may be a subroutine which performs some activity. A USR jump can then be done to enter the subroutine from BASIC. The use of ADR is as follows.

1. Create string (X$ for example) by using machine language opcodes and addresses.
2. Find the address of the string by using the statement Y=ADR(X$)
3. You can then jump to the routine by using the statement Z= USR(Y)

ADDRESS — An ADDRESS is similar to a post office box number. One byte can be stored at each ADDRESS. There are 64K (65,536) separate ADDRESSes in the Atari computer, so the computer contains 64K bytes of memory. See MEMORY.



ADVENTURE GAME — The original computer adventure game was developed by and for systems people who worked on mainframes and minicomputers. The original adventure used text only, was very restrictive in terms of vocabulary, and was a complex journey to the netherworlds. As a traveler, you were constantly making decisions about which tools to pick up and use, which troll to kill, which room to enter, and which trail to take. Adventure games implemented on Atari computers often have helpful graphics and accept standard English commands. Scott Adams' Adventure International, Infocom, and Sierra Online produce many of the currently available adventure games. Adventure games are for gamers who like to strategize, while arcade games are for those with fast reflexes who like action in real time.

AFTER — In Microsoft BASIC II, the AFTER statement is a delay statement which will allow you to wait a certain number of JIFFIES (1/60ths of a second) before going to a certain line number. The format is:

```
AFTER(6000)100
```

(After 6000 jiffies (100 seconds) go to line number 100.)

ALPHANUMERIC — Characters which are either numbers or letters. This excludes graphics characters and control characters.

AMIS — AMIS is the Atari Message Information Service. MACE (Michigan Atari Computer Enthusiasts) developed AMIS and donated it to the public domain. Atari Users Group Support modified the program somewhat and distributed it to users groups. Many clubs use AMIS as their BBS although many modifications are developing. This means that not all of the commands below are found on all bulletin boards and some may have extra commands. For example, one system operator (SYSOP) found that callers often pressed Y to answer questions while the menu was displayed. As you see below, the Y option calls for the SYSOP. He changed this option to Z to allow him and his wife more sleep at night. AMIS is very easy to use. Once you get familiar with the basic stucture, you will be able to talk on many Atari BBSs around the country. The most common options are:

A	ASCII to ATASCII toggle
B	Bulletins put here by the operator
C	Callers file to list previous callers
D	Download files to your computer
E	Enter message into the system
F	File directory for downloading
G	Goodbye, adios, hang-up
H	Help for using AMIS
K	Kill messages which you have left
L	Line feed switch
Q	Quick summary of messages
R	Read full messages
S	Scan subjects
T	Time of day and date
U	Upload files from your computer

W Welcome message
X eXpert user mode for frequent callers
Y Yell for SYSOP
? Short list of functions

Control X - Stop transmission
Control S - Pause transmission
Control Q - Resume transmission

AMODEM — AMODEM is a versatile public domain terminal program written by J. Steinbrecher. AMODEM features the ability to transfer binary files, access and download CP/M bulletin boards, save files to cassette, and also includes the XMODEM Christensen error checking file transfer protocol. Version 4.3, which is included here, supports autodialing with the Hayes Smartmodem. Put your favorite BBSs in DATA statements 0 through 8. You can also use MCI or Sprint service by putting your access and code in the string in line 917. If you have an XL series computer, you must use the translator program to use the XMODEM protocol.

When the program is RUNning, you will at first be in the terminal mode. When the border is blue, you are in the ATASCII (eight bit) mode. When it is black, you are in the ASCII mode. You will be able to connect with any BBS. Hit the SELECT key when you see a menu similar to the following menu. This will give you the list of control options:

```
OPTION  =  TOGGLE MEMORY SAVE
SELECT  =  (B,C,W,M,D,R,S,T,U,H,X)
START   =  START TRANSMISSION
S FILE  =  D:FILE2SND
```

To download a file, type R for receive. You must then assign a filename for YOUR file which will be on your disk (or use C: for cassette). When your BBS tells you it is ready to send its file, hit the start key and transfer will begin. To upload, type S for Send, and provide the name of your file which you want to send. When prompted to begin sending, hit the START key.

The menu options in AMODEM are:

B — Change Baud rate to 300, 600 or 1200
C — Capture all data sent to you
W — Write captured memory to a file
M — Get a directory listing (menu) of the disk
D — Full or Half duplex toggle
R — Receive a file with XMODEM verify
S — Send a file with XMODEM verify
T — ASCII-ATASCII toggle
U — Uploads with no XMODEM verify
H — Hayes autodial from listing in lines 0-8
X — Expert mode (no help messages)

```
0 DATA ABACUS----,5878062
1 DATA BBS 2-----,1234567
2 DATA BBS 3-----,1234567
3 DATA BBS 4-----,1234567
4 DATA BBS 5-----,1234567
5 DATA BBS 6-----,1234567
6 DATA BBS 7-----,1234567
7 DATA BBS 8-----,7654321
8 DATA BBS 9-----,5555555
10 REM AMODEM4.BAS:VER 4.3; 12/28/82
15 DIM NUM$(25),TMP$(10),X$(1),PRMT$(2
5)
16 PRMT$="(B,C,W,M,D,R,S,T,U,H,X)"
20 GOTO 10000
900 GOSUB 13000:RESTORE 0:TRAP 999
902 ? #MODEM;"ATZ":? CHR$(125)
904 ? "              NUMBER LIST":? CHR$
(29):? CHR$(29)
906 FOR T9=0 TO 8:READ TMP$,NUM$
908 ? "    ";T9;"--";TMP$;"-----";NUM$
:NEXT T9
910 ? CHR$(29):? CHR$(29):? "Enter 0-"
;T9-1;:INPUT A9:IF A9>=9 THEN GOTO 900
912 RESTORE A9:READ TMP$,NUM$
913 ? "USE MCI (Y/N)";:INPUT X$
914 ? CHR$(125):? "Dialing-";TMP$:IF X
$="Y" THEN ? " USING MCI "
915 IF LEN(NUM$)<10 THEN 918
916 IF X$<>"Y" THEN ? " USING BELL ":?
 #MODEM;"ATDT1";NUM$:GOTO 999
917 ? #MODEM;"ATDTXXXXXXX,,,XXXXX";NUM
$:GOTO 999
918 ? #MODEM;"ATDT";NUM$
999 ? #MODEM:TRAP 40000:GOTO MENU
1000 TRAP 1000:GOSUB 13000:? :? " OPTI
ON  = TOGGLE MEMORY SAVE"
1010 ? " SELECT  = (B,C,W,M,D,R,S,T,U,
H,X)"
1020 ? " START   = START TRANSMISSION"
1030 SETCOLOR 2,7,2:C$=CHR$(SRFLAG):IF
 SRFLAG=ZERO THEN C$=" ":FILE$=C$
1040 ? C$;" FILE   = ";FILE$:?
1042 IF  NOT TRN THEN ? "*** ASCII";
1044 IF TRN THEN ? "*** ATARI";
1046 ? " TERMINAL MODE ***"
1050 ADDR=USR(ADR(IO$),ADDR,LEN(BUFF$)
+ADDR-1)
1055 C=PEEK(706):IF C=8 THEN PUT #MODE
```

```
M,19:? "*** BUFFER FULL ***":GOTO 1700
1200 IF C=6 THEN 5000
1210 IF C=5 THEN 6000
1220 IF C<>3 OR SRFLAG<>67 THEN GOTO T
ERM
1230 MSAVE=WON-MSAVE:POKE 704,MSAVE:?
:? "Capture ";
1240 IF MSAVE THEN SETCOLOR 2,0,2:? "O
n  ";
1250 IF  NOT MSAVE THEN SETCOLOR 2,7,2
:? "Off ";
1260 ? ADDR-BUFF;" BYTES"
1270 IF PEEK(CON)=3 THEN 1270
1280 GOTO TERM
1500 ? :? "*** NEW CAPTURE FILE ***"
1510 ? "*** SELECT W WILL SAVE IT!***"
1520 ADDR=BUFF:GOSUB 13000
1530 SETCOLOR 2,0,2:POKE 766,1
1540 MSAVE=1:POKE 704,MSAVE:GOTO TERM
1700 ? :CLOSE #MODEM:IF ADDR<=BUFF THE
N ? "*** BUFFER IS EMPTY ***":GOTO 176
0
1710 TRAP 1760:? "*** SAVING MEMORY **
*"
1720 OPEN #FILE,8,ZERO,FILE$
1730 OBJ=1:IF TRN THEN OBJ=0
1740 POKE 1536,OBJ
1750 C=USR(1610,BUFF,ADDR)
1760 MSAVE=ZERO:POKE 704,MSAVE:ADDR=BU
FF:L$=""
1790 SRFLAG=ZERO:GOTO MENU
2000 TRAN=32:GOSUB IO:A=NAK:POKE 766,1
2010 SETCOLOR 2,4,2:BLOCK=ZERO
2020 ? :? "*** RECEIVING ";FILE$;" ***
"
2300 POKE 77,ZERO:FOR TRY=WON TO ERRTR
Y-WON
2310 ? :? "*** GETTING SECTOR ";BLOCK+
WON;"/";TRY;" ***"
2315 IF PEEK(CON)=5 THEN A=CAN
2320 PUT #MODEM,A:A=ACK
2330 GET #MODEM,SH:SUM=SH:IF SH=EOT OR
 SH=CAN THEN 2380
2340 GET #MODEM,C:SUM=SUM+C:GET #MODEM
,C:SUM=SUM+C
2350 ADDR=BLOCK*128+BUFF:FOR BLK=0 TO
127:GET #MODEM,C:POKE ADDR+BLK,C:? CHR
$(C);:SUM=SUM+C:NEXT BLK
```

```
2360 GET #MODEM,C:SUM=ASC(CHR$(SUM)):I
F C=SUM THEN 2380
2370 CLOSE #MODEM:A=NAK:FOR C=WON TO 4
00:NEXT C:GOSUB IO:POKE 766,WON:GOTO 2
390
2380 TRY=ERRTRY
2390 NEXT TRY:BLOCK=BLOCK+1
2500 IF SH=EOT AND A=ACK THEN 2800
2510 IF SH=CAN OR A<>ACK THEN 2900
2530 GOTO 2300
2800 PUT #MODEM,ACK:? :? "*** SAVING F
ILE ***":TRAP 2860
2805 C=PEEK(ADDR+127)
2810 FOR A=ADDR+C TO ADDR+127:IF PEEK(
A)<>C THEN C=128
2812 NEXT A:ADDR=ADDR+C:CLOSE #MODEM
2820 OBJ=ZERO:A=PEEK(BUFF):IF A>ZERO A
ND A<255 THEN OBJ=WON
2825 A=ZERO:IF FILE$(1,1)="C" AND OBJ=
ZERO THEN A=128
2830 IF TRN THEN OBJ=ZERO
2840 POKE 1536,OBJ:POKE 195,WON:? "***
 ";ADDR-BUFF;" BYTES"
2850 OPEN #FILE,8,A,FILE$:C=USR(1610,B
UFF,ADDR)
2860 GOTO 2990
2900 ? :? "*** UNABLE TO RECEIVE FILE"
:A=NAK
2910 PUT #MODEM,CAN
2990 SRFLAG=ZERO:GOTO MENU
3000 TRAN=32:GOSUB IO:POKE 766,1
3010 SETCOLOR 2,WON,2:BLOCK=ZERO:BYTE=
BYTES
3020 ? :? "*** SENDING ";FILE$;" ***"
3300 POKE 77,ZERO:FOR TRY=WON TO ERRTR
Y
3310 ? :? "*** SENDING SECTOR ";BLOCK+
WON;"/";TRY;" ***"
3320 PUT #MODEM,SOH:SUM=ZERO
3330 PUT #MODEM,BLOCK+WON
3340 PUT #MODEM,254-BLOCK
3350 ADDR=BLOCK*128+BUFF:FOR BLK=0 TO
127:C=PEEK(ADDR+BLK):PUT #MODEM,C:? CH
R$(C);:SUM=SUM+C:NEXT BLK
3360 SUM=ASC(CHR$(SUM)):PUT #MODEM,SUM
3370 GET #MODEM,A:IF A=CAN OR PEEK(CON
)=5 THEN 3900
3380 IF A<>ACK THEN 3400
```

```
3390 TRY=ERRTRY
3400 NEXT TRY:BLOCK=BLOCK+1
3500 IF A<>ACK THEN 3900
3510 BYTE=BYTE-128:IF BYTE>ZERO THEN 3
300
3800 PUT #MODEM,EOT:PUT #MODEM,ZERO
3810 ? :? "*** TRANSFER COMPLETE ***"
3820 GOTO 3990
3900 ? :? "*** UNABLE TO SEND FILE ***
"
3910 PUT #MODEM,CAN
3990 GOTO MENU
4000 ? :CLOSE #MODEM
4010 FOR C=49 TO 52
4020 L$="D1:*.*":L$(2,2)=CHR$(C)
4030 TRAP 4060:OPEN #FILE,6,ZERO,L$:?
L$:TRAP 4050
4040 INPUT #FILE;L$:? L$:GOTO 4040
4050 PRINT
4060 TRAP 4065:CLOSE #FILE
4065 IF DR=WON THEN 4080
4070 NEXT C
4080 DR=ZERO:L$="":GOTO MENU
4500 POKE 766,WON:SETCOLOR 2,2,2:? :?
"*** UPLOADING ";FILE$;" ***"
4510 FOR I=BUFF TO BUFF+BYTES-129+BYTE
4520 PUT #MODEM,PEEK(I):IF PEEK(CON)=5
 THEN ? :? "*** ABORTED ***":GOTO 4550
4530 STATUS #MODEM,C:BLK=PEEK(747):IF
BLK THEN FOR A=WON TO BLK:GET #MODEM,C
:? CHR$(C);:NEXT A
4540 NEXT I
4550 FOR I=1 TO 100:NEXT I
4560 STATUS #MODEM,C:IF PEEK(747) THEN
 GET #MODEM,C:? CHR$(C);:GOTO 4560
4570 ? :? "*** UPLOAD COMPLETE ***":GO
TO MENU
5000 IF SRFLAG=67 THEN 1500
5010 IF SRFLAG=82 THEN 2000
5020 IF SRFLAG=83 THEN 3000
5030 IF SRFLAG=85 THEN 4500
5040 ? :? "*** MUST SELECT FIRST! ***"
5050 IF PEEK(CON)<>7 THEN 5040
5060 GOTO TERM
6000 ? CHR$(125):? "              FUNCTI
ON MENU":IF XP=1 THEN ? CHR$(29):? "
     ";PRMT$:GOTO 6012
6001 ? "Baud---------Change baud rateS
                     300,600,1200"
```

```
6002 ? "Capture------Captures anything
 sent                 to you"
6003 ? "Write--------Writes memory to
                      specified device"
6004 ? "Menu---------Displays disk one
                      files  or enter 1
-4"
6005 ? "Duplex-------Switches from
                      full or half dupl
ex"
6006 ? "Receive------Xmodem transfer u
sing                  the Christensen p
rotocol"
6007 ? "Send---------Sends a file usin
g                     XMODEM transfer"
6008 ? "Translation--Switches from ATA
SCII                  to ASCII and back
"
6009 ? "Upload-------Uploads a file wi
th                    no verify"
6010 ? "Hayes--------Goes to phone num
ber                   menu for autodial
ing"
6011 ? "Xpert--------Expert user mode"
6012 CLOSE #MODEM:? :? "What function?
";:GET #KEY,C:C$=CHR$(C):? C$
6013 IF C$="H" THEN 900
6014 IF C$="B" THEN 9900
6015 IF C$="C" THEN 7000
6020 IF C$="W" THEN 1700
6025 IF C$="U" THEN 8000
6030 IF C$="M" THEN 4000
6035 IF C$="R" THEN 7000
6040 IF C$="S" THEN 8000
6042 IF C$="X" AND XP=0 THEN XP=1:GOTO
 MENU
6044 IF C$="X" AND XP=1 THEN XP=0:GOTO
 MENU
6045 IF C$="T" THEN TRN=32-TRN:IF SRFL
AG>82 THEN SRFLAG=ZERO
6050 IF C$="D" THEN PLX=1-PLX:POKE 705
,PLX
6055 DR=0:IF C>48 AND C<53 THEN DR=WON
:GOTO 4020
6060 GOTO MENU
7000 SRFLAG=ZERO:MSAVE=ZERO:? :? "***
RECEIVE FILESPEC ";
7010 INPUT L$:IF L$="" THEN 7090
```

```
7015 TRAP 7000:IF L$(2,2)<>":" THEN IF
 L$(3,3)<>":" THEN ? "SPECIFY DEVICE!"
:GOTO 7000
7020 FILE$=L$:IF L$(1,1)<>"D" THEN 708
0
7030 TRAP 7080:OPEN #FILE,4,ZERO,FILE$
7040 ? :? "*** HAVE FILE ";FILE$
7050 ? "*** Type (Y) to ERASE ";FILE$;
" ";
7060 GET #KEY,A:? CHR$(A):IF A<>89 THE
N L$="":GOTO 7090
7070 CLOSE #FILE:XIO 36,#FILE,ZERO,ZER
O,FILE$:XIO 33,#FILE,ZERO,ZERO,FILE$
7080 SRFLAG=C:ADDR=BUFF
7090 TRAP 40000:GOTO MENU
8000 SRFLAG=ZERO:? :? "*** SEND FILESP
EC ";:INPUT L$:IF L$="" THEN 8090
8005 TRAP 8000:IF L$(2,2)<>":" THEN IF
 L$(3,3)<>":" THEN ? "SPECIFY DEVICE!"
:GOTO 8000
8010 A=ZERO:IF L$(1,2)="C:" THEN A=128
8014 SRFLAG=C:? "*** LOADING INTO BUFF
ER ***":OBJ=0
8015 ADDR=BUFF:TRAP 8080:FILE$=L$:OPEN
 #FILE,4,A,FILE$
8020 IF TRN THEN 8050
8030 GET #FILE,A:POKE ADDR,A:ADDR=ADDR
+1:IF A>ZERO AND A<255 THEN OBJ=1
8050 POKE 1536,OBJ
8060 C=USR(1537,ADDR):BYTES=C-BUFF:BYT
E=((BYTES/128)-INT(BYTES/128))*128
8065 IF PEEK(195)<>136 THEN ? "*** ERR
OR ";PEEK(195):GOTO 8085
8070 FOR A=C TO C+127-BYTE:POKE A,BYTE
:NEXT A:C=A:BYTES=C-BUFF:GOTO 8090
8080 ? CHR$(253);"*** FILE NOT FOUND *
**"
8085 SRFLAG=ZERO:L$=""
8090 TRAP 40000:GOTO MENU
9000 TRM=32-TRM
9010 GOSUB IO:GOTO MENU
9900 BAUD=BAUD+1:IF BAUD>10 THEN BAUD=
8
9910 IF BAUD<10 THEN ? 300*(BAUD-7);
9920 IF BAUD=10 THEN ? 1200;
9930 ? " BAUD":GOTO MENU
10000 C=FRE(0)-400:DIM BUFF$(C),IO$(17
0):BUFF=ADR(BUFF$):ADDR=BUFF
```

```
10005 ZERO=0:WON=1:SOH=1:EOT=4:ACK=6
10010 BEL=7:BS=8:LF=10:VT=11:CR=13
10020 NAK=21:CAN=24:EOF=26:EOL=ZERO
10030 KEY=1:FILE=2:PTR=3:MODEM=4
10040 DIM C$(1),FILE$(15),L$(130)
10050 MENU=1000:TERM=1050:PLX=0
10060 ERRTRY=10:CON=53279:IO=14000
10070 OPEN #KEY,4,ZERO,"K:"
10080 BAUD=8:GRAPHICS ZERO:?
10120 XIO 34,#MODEM,192,ZERO,"R1:"
10130 XIO 36,#MODEM,BAUD,ZERO,"R1:"
10180 BUFF$(1)=" ":BUFF$(C)=" "
10190 BUFF$(2,LEN(BUFF$))=BUFF$
11000 ? "          ATARI MODEM VER. 4.3"
11010 ? " COPYRIGHT(C) 1982 JIM STEINB
RECHER"
11020 ? "            37220 TRICIA DRIVE"
11030 ? "          STERLING HTS MI. 48077
"
11040 ? :? "  BUFFER= ";C;" BYTES, ";I
NT(C/128);" SECTORS":?
11050 ? "  WITH  WARD CHRISTENSEN'S  X
MODEM"
11060 ? "          FILE TRANSFER PROTOCOL
"
11070 ? "    FOR USE ON  ASCII CP/M SYS
TEMS"
11080 ? :? "     ATARI TO ATARI FILE TR
ANSFER"
11090 ? "       AND SELECTED ATARI SYSTE
MS"
12000 RESTORE 15000:FOR C=1536 TO 1736
:READ A:POKE C,A:NEXT C
12010 FOR C=1 TO 152:READ A:IO$(C)=CHR
$(A):NEXT C
12020 POKE 704,MSAVE:POKE 705,PLX
12030 GOTO MENU
13000 TRAP 13000:TRAN=TRN
14000 CLOSE #MODEM:CLOSE #PTR:CLOSE #F
ILE
14005 XIO 36,#MODEM,BAUD,ZERO,"R1:"
14010 XIO 38,#MODEM,TRAN,ZERO,"R1:"
14020 OPEN #MODEM,13,ZERO,"R1:"
14030 XIO 40,#MODEM,ZERO,ZERO,"R1:"
14040 POKE 712,TRN*4.1:POKE 707,0:POKE
 766,ZERO
14050 TRAP 40000:RETURN
15000 DATA 1,104,104,133,213,104,133,2
```

```
12,162,32,169,7,157,66,3,169,0,157,72,
3
15010 DATA 157,73,3,32,86,228,48,40,16
0,0,145,212,173,0,6,201,1,208
15020 DATA 20,177,212,201,155,208,14,1
69,13,145,212,230,212,208,2,230,213,16
9,10,145
15030 DATA 212,230,212,208,2,230,213,2
4,144,196,132,195,96,74,68,83
15040 DATA 104,104,133,204,104,133,203
,104,133,206,104,133,205,162,32,169,11
,157,66,3
15050 DATA 169,0,157,72,3,157,73,3,160
,0,173,0,6,201,1,208,26,177,203,201
15060 DATA 13,208,20,160,1,177,203,201
,10,208,12,160,0,230,203,208,2,230,204
,169
15070 DATA 155,145,203,160,0,177,203,3
2,86,228,230,203,208,2,230,204,165,203
,197,205
15080 DATA 208,187,165,204,197,206,208
,181,96
15090 DATA 169,13,157,66,3,76,86,228,1
69,7,32,189,6,76,86,228
15100 DATA 168,169,11,32,189,6,152,76,
86,228,157,66,3,169,0,157,72,3,157,73,
3,96
16000 DATA 104,104,133,213,104,133,212
,104,133,215,104,133,214
16010 DATA 162,64,32,163,6,173,235,2,2
01,0,240,68,162,64,32,171,6
16020 DATA 172,200,2,192,0,208,16,201,
7,208,2,169,253,201,8,208,2,169,126
16030 DATA 201,32,144,20,172,192,2,240
,10,162,0,129,212,230,212,208,2,230,21
3,162,0,32,179,6
16040 DATA 165,215,197,213,208,190,165
,214,197,212,208,184,169,8,141,194,2,9
6
16060 DATA 240,176,173,252,2,201,255,2
40,41,162,16,32,171,6,172,193,2,192
16070 DATA 0,240,5,162,0,32,179,6,172,
200,2,192,0,208,12,201,253,208,2
16080 DATA 169,7,201,126,208,2,169,8,1
62,64,32,179,6,173,31,208,201,7
16090 DATA 16,199,141,194,2,96
```

A.N.A.L.O.G. COMPUTING — ANALOG is a fast growing magazine for Atari computer owners. It was named *Atari Newsletter And Lots Of Games,* hence the acronymn ANALOG. The Atari computers are digital. The programs in the magazine are available on disk form. Some excellent public domain, machine language games can usually be found in ANALOG as well as useful utilities and critical reviews. ANALOG Computing.

ANALOG INPUT — The Atari computer can be used to make measurements such as temperature, humidity, light intensity, or resistance. These inputs are ANALOG as opposed to DIGITAL, meaning that they can vary continuously over a range. The input is made through the paddle ports on the front of the 400 and 800 computers. There are eight paddle ports which go directly to ANALOG to digital converters in the computer. An input device which changes resistance with changes in external conditions is required. For temperature, this would be a thermistor, for light — a photocell, and for humidity — a hygrometer. Memory locations 624 through 631 (decimal) are the shadow registers which hold a value proportional to the resistance measured. The value ranges between 0 and 228, with the higher values generated by higher resistances. The range of resistances that can be measured is 100 to 100,000 ohms. In order to convert the resistance to a standard value, two measurements should be made at known points (such as boiling water and ice water). Use a PEEK(624) to measure the resistance at the paddle port. The register is updated every sixtieth of a second.

AND — In BASIC, the logical operator AND is used to construct a conditional statement which is true only if both sides of the AND statement are true.

```
EXAMPLE: IF A=65 AND B=66 THEN 100
```

The program will go to line 100 only if A equals 65 AND B equals 66. Otherwise, the next line number will be executed.

On a bit level, the logical AND compares the bits which make up two different bytes. If the bits of one byte are set (equal to 1) AND if and only if the bits of the other byte are 1 then the result is a 1. Otherwise, the result is a 0.

```
EXAMPLE: FIRST BYTE        00001111
         SECOND BYTE       11111000

         Result of AND     00001000
```

ANSWER MODE — When a modem is set in the ANSWER MODE, it emits an audible tone (at 2025 or 2225 HZ) while it is waiting for an originating signal. When you dial a bulletin board system and hear the high pitched squeal from the other end, you are hearing a modem in ANSWER MODE.

ANTIC — From AlphaNumeric TV Interface Chip and also a device which exhibits ANTICS as in video ANTICS. This custom integrated circuit allows your Atari computer to control your television to a much greater degree than any currently available on other home computers. The ANTIC is responsible for the mixed screen modes and colors used in many games.

ANTIC — ANTIC is a magazine for Atari users featuring reviews, comparisons of applications software, hardware, and BASIC programs for Atari computers. Emphasis is given to helping new users with hardware or software problems. Product reviews cover the products and the advertisements are a good source of information about the latest products.

APPLICATION PROGRAMS — An APPLICATION PROGRAM is software which allows your computer to do some useful function such as word processing, home budget keeping, or financial analysis. The programs are applied to a number of useful activities.

ARGUMENT — Variables usually found in parentheses or brackets in a demonstration of a function or command. For example, LOAD"D:FILESPEC.BAS" is the method for loading a program from a disk. FILESPEC.BAS is the argument in this demonstration.

ARMUDIC — ARMUDIC is a bulletin board system for Atari computers. ARMUDIC is not a public domain system. The name ARMUDIC is derived from the original telephone number for the system in Washington, DC (202-276-8342). The following options are available when you call an ARMUDIC system.

A Activate message files
S Send message
G Get message
E Erase message
D Download files
L Leave message for system operator
Q Quit
U Upload files
Control S - Pause
Control Q - Continue
Control C - Return to option menu

ARRAY — An ARRAY is a one dimensional set of elements. For example, a series of numbers stored in memory or a string of characters. A matrix is a two dimensional set of elements. Atari BASIC supports one or two dimensional numerical arrays. The array must be initialized with a DIM statement before it can be used. You can use arrays in BASIC XL without DIMming the variables. Atari BASIC will not let you use two dimensional string arrays; you must use BASIC XL, Microsoft BASIC or BASIC A+, or else simulate the string array.

ARTIFACTS — ARTIFACTing is the separation of colors often found in high resolution graphics. This separation is due to the large size of the color "dots" which make up the television screen compared to the pixel size. ARTIFACTing is usually manifested by blue and orange colors found on black backgrounds. ARTIFACTing is possible in ANTIC modes 2, 3, and 15 because the pixels are ½ of a color clock wide. (In BASIC, these modes are called mode 0 for ANTIC mode 2, and mode 8 for ANTIC mode 15. ANTIC mode 3 is not accessible through BASIC). There are 160 visible color clocks across a scan line of the TV. A clock is a cycle of the signal that contains the information

on color, brightness, synchronization, blank, and overscan. In GR.7 a pixel is the same width as a color clock. In GR.0 and GR.8, a clock is comprised of two pixels (each pixel is ½ of a color clock). Depending on the background color and the horizontal position of the pixels on an odd or even location, up to four different color combinations can be generated in a two color mode such as GR.0 or GR.8. Some games use ARTIFACTing to produce special effects and these effects will not appear on a high resolution monitor such as the Commodore 1701/2 nor on a PAL system.

ARTIFICIAL INTELLIGENCE — One area of study in computer science is ARTIFICIAL INTELLIGENCE. The computer can be called intelligent when it can adapt itself to any novel situation by reasoning, understanding relationships, distinguishing truth from fiction, and discovering new meanings. Clearly this is a formidable task for any machine, let alone the humble Atari computer.

ASC — This BASIC function looks at a string and generates the ATASCII code in decimal for the first character in the string. A NULL STRING generates a decimal 44. The format for using ASC is ASC(STRING$).

ASCII — American Standard Code for Information Interchange. ASCII is a standard technique for representing characters with eight bits of data (1s or 0s). Only 128 of the characters are officially assigned. With eight bits, there are 256 different characters possible. Half of the 256 characters are inverse video copies of the other half. ASCII is used to communicate with other computers through modems; however, the Atari computer uses a modification of standard ASCII called ATASCII.

ASM — A program in the Atari Assembler/Editor Cartridge to begin assembly of a source program. The assembler takes the source program (which is written in assembly language) and converts it into machine code. The machine code is processed directly by the 6502 processor in the Atari computer. Besides the ASM program, the Assembler/Editor has an editor program (EDIT) and a debugger (BUG) used in the preparation of assembly language programs.

ASSEMBLER — An ASSEMBLER is a program which takes programs written in assembly language and produces machine language files. The machine language code can be executed directly by the microprocessor in the Atari computer. The ASSEMBLER is actually a language which acts like a word processor to produce SOURCE files which can be converted to machine language. Assemblers available for the Atari computer are: Synassembler by Synapse Software; EDIT 6502 by LJK Enterprises; Assembler/Editor by Atari; MAC/65 by Optimized Systems Software; Atari Macro Assembler.

ASSEMBLY LANGUAGE — ASSEMBLY LANGUAGE is a computer language developed to provide an easier way of giving instructions to a microprocessor. ASSEMBLY LANGUAGE is used to fetch data from memory and put it in the ACCUMULATOR, to operate on data in the ACCUMULATOR, and to stick data back into memory. There are several ways to load data into the accumulator. The following BASIC analog example will explain. Assume that the BASIC variable ACC is the ACCumulator.

ASSEMBLY	BASIC	MODE	FUNCTION
LDA #$FF	ACC=255	Immediate	Load ACC with the value 255
LDA $FF	ACC=PEEK (255)	Absolute	Load ACC with number in location 255
LDA $FF,X	ACC=PEEK (255+X)	Absolute, X	Load ACC with number in 255 + X
LDA ($FF,X)	ACC=PEEK (PEEK (255 + X))	Indirect,X	Load ACC with number pointed to in location 255 + X

To store the data in the ACCUMULATOR into memory, we use the STA form in ASSEMBLY LANGUAGE.

ASSEMBLY	BASIC	MODE	FUNCTION
STA $FF	POKE 255, ACC	Absolute	Put value in ACC into location 255
STA $FF,X	POKE 255+ X,ACC	Absolute, X	Put value in ACC into 255+X
STA($FF,X)	POKE PEEK (255+X),ACC	(Indirect,X)	Put value pointed at in 255+X in ACC
STA($FF),Y	POKE PEEK (255)+Y,ACC	(Indirect),Y	Put value + Y for the pointed at location in 255 into ACC

Note that this is actually simplified because the locations pointed at must be two byte locations, unless they are in the zero page.

ASYNCHRONOUS TRANSMISSION — In telecommunications (using a modem to connect computers), the dialogue between computers can be very structured and coordinated (synchronous) or it may be randomly interactive (asynchrounous). In ASYNCHRONOUS TRANSMISSION, the time intervals between characters may be of unequal length and characters are separated by start and stop elements or bits at the beginning and end of each character. This is the only type of communication possible on the Atari computer.

ATARI KEY — Special key on the Atari 400 and 800 computer keyboard which sets the highest of eight bits when the keyboard is used to generate characters. On the XL series, this key is in the lower right corner and has the legend and a box with a diagonal through the box. The effect of pressing this key is to add 128 to the normal ATASCII value and the character set is displayed in inverse video (dark character on light background.) These characters are ATASCII numbers 128 to 255. Inputs to some BASIC programs may not expect an inverse character and may crash when an inverse charac-

ter is sent. This key is also used to kill the type-ahead buffer in Letter Perfect V.3.x. This key is also called the FUJI key because it looks roughly like Mount Fuji in Japan.

ATARI WRITER — Atari Writer is a powerful word processor for the Atari computer series. All of the Atari printer models are supported and most of the popular printers (Epson, Gemini, Oki, NEC 8023A, etc.) can be driven with an additional printer driver package available on disk from APX. Printer commands can be inserted in text by using a CTRL O followed by the decimal equivalent of the character string code you want to include.(See your printer manual for these codes or look under PRINTER CODES). There is an 80 column print preview which scrolls left and right to show you what your final hard copy will look like. Atari Writer is contained in a Cartridge so it is ready to go as soon as you plug it in (but you must load DOS.SYS if you are using the disk drive). Files can be saved on cassette.

ATASCII — ATAri Standard Code For Information Exchange. A version of ASCII which has identical codes for alphanumeric characters but differs in the first 32 characters (the control characters) and in characters 123 through 127. A translation is required when transferring ATASCII code through modems or the control characters may be interpreted in a way such that transmission will be stopped. The difference is an artifact of the way that the Atari Operating System authors attempted to work around the way that the ANTIC chip handles characters. In high resolution character modes, the right seven bits of the byte are used to generate the character graphics address and these match ASCII codes. When large character modes are used (GR. 1 or 2), only six bits are used for the character and the other two specify colors. If ASCII codes were used, this would allow either upper and lower case OR graphics characters and numbers, but NOT upper case and numbers. ATASCII swaps lower case and numbers to allow uppercase and numbers simultaneously in large character modes.

ATN — in BASIC, ATN returns the arctangent of the argument in parentheses. The answer is expressed in radians unless the DEG statement has been executed.

ATR8000 INTERFACE — This interface is getting rave reviews from the hardcore Atari users' community. The ATR8000 includes all functions of the 850 interface but it allows the Atari computer to connect to many of the most popular peripherals available for other computer systems. Provided on the ATR are: a parallel port for a printer, an RS-232 serial port for a modem or other serial device, an intelligent disk drive controller, and a 4K printer buffer (expandable to 48K). The intelligent disk controller will support single or double sided, single or double density 5¼" or 8" disks. This is quite a combination and allows you to use any of the low cost drives on the market. The interface is controlled by a Z80 microprocessor and can load CP/M so that the Atari thinks the CP/M is in charge and you can actually run CP/M programs. The drives will read almost any format disk: Heath, Osborne, Kaypro, TRS, etc. The ATR8000 is very expandable in both RAM and processor. You can add up to a total of 64K RAM. It is possible to add an 8088 processor and run the 16 bit CP/M-86 or MSDOS and upgrade to 256K of RAM. This makes a very powerful Atari, equivalent to a loaded IBM PC.

```
10 REM ** AUTORUN.SYS BUILDER **
20 REM ** FILE: ARSMAKER.BAS
30 REM ** ABCS OF ATARI COMPUTERS
40 REM
50 GRAPHICS 0:DIM A$(128),B$(12)
60 ? "THIS PROGRAM WILL CREATE A FILE"
70 ? "ON THE DISK CALLED AUTORUN.SYS."
:?
80 ? "FOR EXAMPLE BY ENTERING THE FILE
NAME"
90 ? " 'MENU', IT CREATES A PROGRAM WH
ICH"
100 ? " AUTORUNS ANY FILE CALLED 'MENU
'":?
110 ? :? "ENTER FILE NAME TO AUTORUN";
:INPUT B$
120 A$(1,6)="RUN D:":A$(4,4)=CHR$(34):
A$(7,7+LEN(B$))=B$:A$(7+LEN(B$))=CHR$(
34)
130 OPEN #1,8,0,"D:AUTORUN.SYS"
140 PUT #1,255
150 PUT #1,255
160 PUT #1,0
170 PUT #1,6
180 L=123+LEN(A$)-1
190 PUT #1,L
200 PUT #1,6
210 FOR I=1 TO 123
220 READ D
230 IF I=64 THEN PUT #1,LEN(A$)-1:GOTO
 250
240 PUT #1,D
250 NEXT I
260 FOR I=LEN(A$) TO 1 STEP -1
270 PUT #1,ASC(A$(I,I))
280 NEXT I
290 PUT #1,255
300 PUT #1,255
310 PUT #1,226
320 PUT #1,2
330 PUT #1,227
340 PUT #1,2
350 PUT #1,0
360 PUT #1,6
370 CLOSE #1
380 END
390 DATA 162,0,189,26,3,201,69,240,5,2
32
```

```
400 DATA 232,232,208,244,232,142,105,6
,189,26
410 DATA 3,133,205,169,107,157,26,3,23
2,189
420 DATA 26,3,133,206,169,6,157,26,3,1
60
430 DATA 0,162,16,177,205,153,107,6,20
0,202
440 DATA 208,247,169,67,141,111,6,169,
6,141
450 DATA 112,6,169,10,141,106,6,96,172
,106
460 DATA 6,240,9,185,123,6,206,106,6,1
60
470 DATA 1,96,138,72,174,105,6,165,205
,157
480 DATA 26,3,232,165,206,157,26,3,104
,170
490 DATA 169,155,160,1,96,0,0,0,0,0
500 DATA 0,0,0,0,0,0,0,0,0,76
510 DATA 0,0,0
```

ATTRACT MODE — In order to prevent overworking and overheating of the mask and phosphors of the color television hooked to your Atari computer, a technique called ATTRACT MODE is implemented. ATTRACT MODE is activated when location 77 ($4D) contains a value of 127. This location is incrementally POKEd every four seconds until after about nine minutes the maximum of 127 is reached. You can reset the attract timer by doing a POKE 77,0. Every time you hit a key on the keyboard, it is reset on the theory that you are changing the screen by typing characters. Using the joystick or trigger does not reset the attract mode. While the colors are rotating, the brightness is lowered to further prevent damage.

AUTO — In Microsoft BASIC II AUTO Start, Increment enables the AUTO line numberer with the first line number as defined in Start and the steps between line numbers as given in Increment.

AUTO LINK — AUTO LINKing is a technique by which files to be printed by a word processing program can be automatically loaded into memory and then printed without intervention. You can AUTOLINK files with the Atari Writer, Letter Perfect, and Text Wizard. In Letter Perfect, the technique is to use CTRL V "sample. In Text Wizard, the command is CTRL V D:SAMPLE, where SAMPLE is the name of the next file to print. In the Atari Writer, the command is CTRL V D:SAMPLE. The next file loaded in will be printed with the format directed by the commands contained in the file.

AUTO.SYS — AUTO.SYS was used in the original Atari DOS (Version 1) to set up system parameters, such as the left and right margins. DOS 1 is very rarely found these days. When the system is powered up with a disk drive connected, it will first load in

DOS.SYS and then try to run any AUTO.SYS file on the disk. The AUTO.SYS file must be constructed with the first six bytes of the file supplying the following information. (Note that this is for DOS 1 ONLY.)

BYTE	HEX	DEC	Meaning
1	$84	132	DOS 1 ID
2	$09	9	DOS 1 ID
3	LO	LO	Starting memory address (low byte)
4	HI	HI	Starting memory address (high byte)
5	LO	LO	Ending memory address (low byte)
6	HI	HI	Ending memory address (high byte)

FILE DATA FOLLOWS

AUTO DIAL — If you have a Hayes Smartmodem or another brand capable of automatic dialing, you can use your keyboard to dial your favorite BBS. The command to AUTODIAL with the Hayes is ATDT 5551234 where 5551234 is the phone number you want to call. The command ATDT is given while the terminal program is running and is in the terminal mode. The "AT" puts the modem at "ATtention" and the "DT" stands for "Dial Tone". The AMODEM program listed here can do auto dialing.

AUTORUN.SYS — AUTORUN.SYS is the name of a file which DOS 2.0S version disks will attempt to load and run before turning control over to you. The file must be a "binary load" file with certain parameters set up according to the following format. The Starting address is found in the third and fourth bytes of an AUTORUN.SYS file. (Bytes 1 and 2 are $FF and $FF, the hexadecimal equivalent of 255). The starting address is the location in memory where the file is to begin loading. Bytes 3 and 4 of the AUTORUN.SYS file are the Ending address in memory where the file should stop loading. The difference between the ending address and the starting address is the length of the file in bytes. At the end of the file the RUN address is loaded into the RUNAD vector at locations 736 and 737 ($2E0 and $2E1). This passes control of the system through this vector to the starting address of the binary file. If an INIT address is loaded into locations 738 and 739 ($2E2 and $2E3), then this program will begin executing immediately as the data is loaded into that memory. This might be used for music or an introduction during loading.

AUTORUN.SYS BUILDER — The Atari Operating System looks for DOS.SYS when the computer is powered up (coldstart) and then it looks for a file called AUTORUN.SYS. The AUTORUN.SYS file is a machine language program which may be used to run a BASIC program. It can also be a self-contained machine language program (such as a game), which will run if the INIT and RUN addresses are appended. The following BASIC program will build an AUTORUN.SYS for you. The program is designed to create a RUN"D:filespec" command where all you have to do is enter the filespec. You can have only one AUTORUN.SYS file on a disk. Multiple files will be ignored.

```
10 REM ** AUTORUN.SYS BUILDER **
20 REM ** FILE: ARSMAKER.BAS
30 REM ** ABCS OF ATARI COMPUTERS
40 REM
50 GRAPHICS 0:DIM A$(128),B$(12)
60 ? "THIS PROGRAM WILL CREATE A FILE"
70 ? "ON THE DISK CALLED AUTORUN.SYS.":?

80 ? "FOR EXAMPLE BY ENTERING THE FILENA
ME"
90 ? " 'MENU', IT CREATES A PROGRAM WHIC
H"
100 ? " AUTORUNS ANY FILE CALLED 'MENU'"
:?
110 ? :? "ENTER FILE NAME TO AUTORUN";:I
NPUT B$
120 A$(1,6)="RUN D:":A$(4,4)=CHR$(34):A$
(7,7+LEN(B$))=B$:A$(7+LEN(B$))=CHR$(34)
130 OPEN #1,8,0,"D:AUTORUN.SYS"
140 PUT #1,255
150 PUT #1,255
160 PUT #1,0
170 PUT #1,6
180 L=123+LEN(A$)-1
190 PUT #1,L
200 PUT #1,6
210 FOR I=1 TO 123
220 READ D
230 IF I=64 THEN PUT #1,LEN(A$)-1:GOTO 2
50
240 PUT #1,D
250 NEXT I
260 FOR I=LEN(A$) TO 1 STEP -1
270 PUT #1,ASC(A$(I,I))
280 NEXT I
290 PUT #1,255
300 PUT #1,255
310 PUT #1,226
320 PUT #1,2
330 PUT #1,227
340 PUT #1,2
350 PUT #1,0
360 PUT #1,6
370 CLOSE #1
380 END
390 DATA 162,0,189,26,3,201,69,240,5,232
400 DATA 232,232,208,244,232,142,105,6,1
89,26
```

```
410 DATA 3,133,205,169,107,157,26,3,232,
189
420 DATA 26,3,133,206,169,6,157,26,3,160
430 DATA 0,162,16,177,205,153,107,6,200,
202
440 DATA 208,247,169,67,141,111,6,169,6,
141
450 DATA 112,6,169,10,141,106,6,96,172,1
06
460 DATA 6,240,9,185,123,6,206,106,6,160
470 DATA 1,96,138,72,174,105,6,165,205,1
57
480 DATA 26,3,232,165,206,157,26,3,104,1
70
490 DATA 169,155,160,1,96,0,0,0,0,0
500 DATA 0,0,0,0,0,0,0,0,0,76
510 DATA 0,0,0
```

AUTOTAB—The Bit 3 Full View 80 column board for the Atari 800 has an automatic column tabulator. All data PRINTed to the screen with a comma (,) after the variable to be printed will be lined up in regular columnar fashion.

B

B SLASH (ƀ) — Older program listings often use the B-SLASH symbol to indicate a blank space in a statement. This reduces confusion as to how many spaces are intended or whether or not a space is required.

BACKGROUND — The BACKGROUND is the part of the screen inside the borders and behind the text, if any. The color of the background is set by using a SETCOLOR 2,C,L command in BASIC where C is the color (from 0 to 15) and L is the brightness or luminance (from 0 to 14). The register at location $02C6 (710) (COLOR2) is the background register in graphics modes 0 and 8. Register $02C8 (712) (COLOR4) is the background register for GR. 1,2,3,4,5,6 and 7.

BAD SECTOR — This was a commonly used disk copy protection technique. A bad sector can be made by physically damaging a disk or by writing data to a sector in a slowed down mode or with a weak write current. Program code is then used to read a bad sector and if an error is not produced, the program terminates. It is not possible to write a bad sector to your disk without modifying the 810 disk drive hardware, hence it is a copy protection scheme. This modification involves slowing down the speed of the drive motor or changing the voltage to the write head.

BANDWIDTH — BANDWIDTH is the range of frequencies assigned to a channel; for example, the video monitor input. It is the difference in Hertz (cycles per second) between the highest and lowest frequencies of a band. CRT monitors have more information on display at any one time and need more data/second to fill the screen than televisions so they operate at a higher BANDWIDTH, or data rate. A television receiver cannot accept data at this rate and so it is either blurry or does not work at all. Monitors usually have a BANDWIDTH of 18 to 25 MHz.

BANK STREET WRITER — This is a word processor program designed for simplicity and ease of use. A tutorial on the flip side of the program disk is really all the documentation you need to use this package. Once you know the structure of the system, you will be able to use the ever-present prompts at the top of the screen. Although Bank Street Writer has plenty of editing features and is truly easy to use, the penalty is the lack of versatility. Formatting of printed files is very limited. For example, it is virtually impossible to change to double spaced lines or to narrower margins within a document. You will not be able to control all of the fancy features on a versatile printer, either. A small kid just learning to write should be able to use this word processor without trouble. Even older computer users will have little difficulty.

BANK SELECT MEMORY — An extra 4K of RAM can be addressed without modifications to the operating system of the 400 and 800 Atari computers. This 4K block starts at $C000 and is used by the Operating System in the XL series. This means that 52K of memory can be addressed in a contiguous block. By installing a board with 16K of memory in addition to the regular 48K in a 400 or 800, the extra 4K can be BANK SELECTED. Since it can be switched on by writing to a special location, the 4K block of

memory can be switched in and out by a technique called BANK SWITCHING. If this memory is used to store screen data, then pictures can be flipped very quickly. It could also be used for player/missile data or alternate character sets. Unfortunately, most commercial software does not support Bank Switching. The Mosaic 64K RAM Select board allows this technique.

BASIC — BASIC is the most universally used computer language for beginners. It is easy to write short utility programs in BASIC. Atari BASIC is available on an 8K ROM cartridge for the 400, 800, and 1200 models and is built into the XL series computers. BASIC is a programming language which is entered through memory location $A000 (40969 decimal). The program converts English-like statements into a series of tokens which are then translated into machine language instructions. All of this interpretation is what slows down the execution of BASIC. Machine language programs do not need these intermediate steps. Alternatives to Atari BASIC are: Microsoft BASIC from Atari, BASIC XL from OSS and BASIC A+ from OSS.

BASIC COMMANDER — BASIC COMMANDER is a machine language routine that loads into just 4K of memory and becomes a hidden helper for the BASIC programmer. Several features which are missing from the BASIC cartridge from Atari are now available. Block deletions of unwanted line numbers are possible through the Commander. Auto renumber also works very fast and takes care of all GOTOs and GOSUBs. An auto line numberer is also included to save a few keystrokes while writing in BASIC. Three user definable keys can be made to enter a commonly used command. CTRL-1, for example, could be defined to enter a command such as RUN"D:TESTER": X=USR(54818). Every time CTRL-1 was pressed, the command would be executed. Commonly used lines such as LOAD"D:__ and RUN"D:__ are built in to the code. The latest version takes 0.4K less memory and does not disappear when System Reset is pressed.

BASIC DEBUGGER — This utility allows you to trace a BASIC program by stepping through it or by getting reports of line numbers and variables as it executes. A split screen allows you to view the program and to look at the reports as the debugging proceeds. A cross reference function prints a list of variables and the line numbers in which they occur.

BAUD — A unit of signal speed used in communications, particularly with modems and terminal programs for personal computers. The term BAUD is derived from the last name of J.M.E. Baudot, a nineteenth century Frenchman who developed the Baudot code for telegraph transmission. BAUD is the number of discrete signal events per second, usually meaning the number of bits per second. Transmission at 300 BAUD is roughly equivalent to 37 characters per second. There are eight bits per ASCII or ATASCII character with a stop bit and a start bit to separate characters. The most commonly used baud is 300, although 1200 baud modems are becoming less expensive and more popular.

BAUD RATE — See BAUD. BAUD RATE is actually redundant since BAUD is a rate.

BCD-BINARY CODED DECIMAL — This system is a hybrid of the binary digit system and the decimal number system. Numeric constants in ATARI BASIC are stored in memory as six BYTE BCD numbers. Each byte is composed of two digits, each of which is made of four bits.

decimal 17 = binary 0001 0001 = BCD 0001 0111

In binary, the eight places of the byte are used to build a number from 0 to 255 with each column from right to left representing a higher power of 2. In the BCD system, the byte is broken down into two four-bit nibbles with values from 0 to 9. Actually, the nibble could hold up to 16 values but since we are dealing in DECIMAL, only 0 through 9 are used. In Atari BASIC the six byte BCD value can have up to ten significant places. One byte is used for the decimal point and sign. The number 12,345 (decimal) would be represented in BCD as follows:

12,345 = BCD 20 01 23 45 00 00

BENCHMARK — To compare the processing speed of different computers and computer languages, a BENCHMARK program is used. A typical BENCHMARK program will find all the prime numbers between 0 and 100,000 and measure the time to completion. The test is subject to the language used. BENCHMARKs are easily distorted by subtleties and should not be rigorously adhered to.

BGET — In BASIC XL, BGET is used to "get" a specified number of bytes through an OPENed channel and store them at a specified location in memory. The format for using BGET is:

```
BGET #iocb, addr, numbytes
```

where #iocb is the channel which is OPENed or a string address, addr is the location in memory where you want to store the bytes you get, and numbytes is the number of bytes to get.

BIAS — A recording tape is recorded with a certain voltage applied to the recording head. Different magnetic media require different voltages which are usually selectable via a BIAS switch. Chromium dioxide, iron oxide, and metal tapes will not play back properly if they are recorded with the wrong BIAS. Inexpensive conventional iron oxide tapes will work fine on the Atari 410 and 1010 Program recorders.

BINARY — A BINARY system has only two states. This is the simplest of all numerical systems, as the element can only be on or off. In a decimal system each element can be valued from 0 to 9. The major drawback of binary is that many positions or locations are required to represent larger numbers or to perform activities. Many computers provide 524,288 (65,536 multiplied by 8) bits in which to do this. A number is processed in groups of eight bits which can represent a number from 0 to 255. Binary digits make up bytes in the following fashion. Each column in the byte is raised to an increasing power of 2.

COLUMN							
7	6	5	4	3	2	1	0
VALUE OF COLUMN							
2X64=128	2X32=64	2X16=32	2X8=16	2X4=8	2X2=4	2X1=2	1

A binary eight bit number with all numbers "ON" (all 1s) would be the sum of all the column values:

Binary 1111 1111 = 255

1X128 + 1X64 + 1X32 + 1X16 + 1X8 + 1X4 + 1X2 + 1X1 = 255

If any of the digits were zero (0), then that column would not be added to the total for the value of the byte.

BINARY LOAD — As a verb, to load a file from a disk directly into the memory of the computer. The data comes in as a series of bytes made up of eight bits each which are stored in various parts of the memory map. An examination of the disk sectors of a binary load file would show very few or no recognizable English or computer words, just a series of ATASCII characters. A RUN address and an INIT address are used to turn over control of the system to the binary load program after loading. The INIT address is the location in the computer's memory where the file data starts. If no RUN address is found at the end of the file, nothing will occur after loading. Binary loads are implemented from the DOS menu by using the L option. After L is pressed, the filename of the file to be loaded is requested.

The following program will check the starting address, ending address, length in bytes, and the DOS version of a BINARY LOAD FILE. Type in this program, save it to a disk file, and RUN it. To analyze a BINARY LOAD file, just type in the name of the file when prompted. Do not forget to type the D:.

```
10 REM **ABCS OF ATARI COMPUTERS
20 REM ** PROGRAM NAME: BLCHECK.BAS
30 REM ** BY PETE GOODEVE
40 REM
50 DIM FILE$(20)
60 ? CHR$(125);"  Binary Load Address
Check"
70 ? :? :? "Enter Device:Filename";:IN
PUT FILE$
80 OPEN #1,4,0,FILE$
90 ? :? :GET #1,DUMMY:GET #1,DUMMY2:GO
SUB 160
100 GET #1,S1:GET #1,S2:GET #1,E1:GET
#1,E2:START=S1+S2*256:FINISH=E1+E2*256
110 ? "Starting Address---";START
120 ? "Ending Address-----";FINISH
130 ? "Length------------";FINISH-STA
RT+1
```

```
140 ? "                          ";((FINISH-S
TART)+1)/1024;"K"
150 END
160 IF DUMMY=255 AND DUMMY2=255 THEN ?
 "DOS 2.0 Binary Load Format":RETURN
170 IF DUMMY=132 AND DUMMY2=9 THEN ? "
DOS 1.0 Binary Load Format":RETURN
180 ? "Not a Binary Load File":POP :EN
D
```

BINARY LOAD FROM BASIC — Binary Load files are normally loaded in by using the L option in the DUP.SYS part of Atari DOS. The routine to load and run the files is in the FMS (File Management System) which resides in memory. As long as the program does not interface with BASIC and it has INIT and RUN addresses, the file will run. Put the name of the file to run in the FILENAME.OBJ string in this program and type RUN.

```
10 DIM FN$(16)
20 FN$="D:FILENAME.OBJ"
30 FN$(LEN(FN$)+1)=CHR$(155)
40 POKE 5534,0:POKE 5534,192
50 X=ADR(FN$)
60 Y=INT(X/256)
70 POKE 853,Y:POKE 852,X-256*Y
80 X=USR(ADR("hL ")):REM String is lowe
r case h, capital L, INVERSE SHIFT-0, CO
NTROL-U
```

The USR call is PLA, JMP $15A9 and it goes into the subroutine for the DOS L command.

Note: From Indianapolis Atari Users' Group, May 1983, Vol. 3, No. 5.

BINARY SAVE — To save data from the memory of the computer into a disk file. The K option on the DOS menu initiates the BINARY SAVE function. The data between two addresses in memory are copied out through the serial bus to the disk drive and placed on the disk as a series of bytes representing the file. The K option puts six bytes as a header on the file you create.

BYTE	DEC	HEX	MEANING
1	255	$FF	DOS 2 ID
2	255	$FF	DOS 2 ID
3	LO	LO	Starting Memory Address (Low byte)
4	HI	HI	Starting Memory Address (High byte)
5	LO	LO	Ending Memory Address (Low byte)
6	HI	HI	Ending Memory Address (High byte)

You must supply the starting address of the data you want to place in the file from memory. Also include the ending address, and the name of the file to which the data will be saved. You will then need to append a RUN address to the end of the file if you want to have it run automatically when you do a binary load. Otherwise, you will load the file into memory and then you will have to use the M option to "RUN at address" to get the program to run.

BIT — A bit is the smallest unit of information that can be stored in a computer memory. Bits are stored on disks, cassette tapes, in RAM, on ROM cartridges, and in other types of memory. In order to have a bit, there must be a two state situation: on/off, high/low, black/white, and so on. One state is defined as a 1 and the other as a 0. The Atari computer uses bits to build larger information units called bytes. Bits are used as digits in a binary numbering system which will be described here. Instead of the familiar decimal system where each subsequent digit in a number (moving from right to left) is 10 raised to a power, the binary system uses 2 raised to the next power.

Decimal column

8	7	6	5	4	3	2	1	0
100,000,000	10,000,000	1,000,000	100,000	10,000	1000	100	10	1

Binary column

8	7	6	5	4	3	2	1	0
256	128	64	32	16	8	4	2	1

A number such as 6,789 can be broken down by multiplying the value in each column by the number corresponding to that place and adding the components.

```
6 X 1000 =  6000
7 X  100 =   700
8 X   10 =    80
9 X    1 =     9
          ------
          6789  TOTAL
```

A binary number, such as those used in the Atari computer, can be evaluated the same way. Binary numbers are stored as eight digit or eight bit numbers. Let us look at the number 00110101.

```
0 X 128 =  0
0 X  64 =  0
1 X  32 = 32
1 X  16 = 16
0 X   8 =  0
1 X   4 =  4
0 X   2 =  0
1 X   1 =  1
          --
          53  TOTAL
```

Thus, the binary number 00110101 is equal to 53 in the decimal system. With eight bits we can represent any number between 0 and 255. By having various bits turned on or off in each of the memory locations in the computer, the Atari constructs a series of numbers each between 0 and 255, and each has a unique meaning. The individual bits are never visible to the user. The smallest visible information unit is the byte which is a collection of eight bits. The byte is represented by a character, conveniently numbered from 0 to 255. By looking up the character on an ATASCII chart and translating the decimal equivalent into binary digits, you can determine which bits are 1s and which bits are 0s. See BYTE.

BIT 3 BOARD - FULL VIEW 80 — This 80 column option card plugs into the third slot in the back of the Atari 800 and allows representation of a full 80 columns of 24 lines on a CRT monitor. A regular television is not workable with this card because the TV has too narrow a bandwidth to process all of the information. The monitor must present 560 pixels across the screen and a TV can only resolve about 320. A 32K board must be installed in slot 2 if a full 48K system is required, because the BIT 3 board takes the last slot. The output from the DIN plug on the side of the 800 is fed into the board and an RCA socket on the rear is used to connect to a monitor. A TV can be used simultaneously when not in the 80 column mode. In other graphics modes besides Mode 0, the BIT 3 board is not noticeable. BASIC programs can use the 80 column capability by software switching through a USR jump such as X=USR(54818). This will switch the screen to the 80 column mode. A POKE 1276,80 is required to enable Microsoft BASIC to operate in 80 columns. This peripheral is the first requirement for making the Atari 800 a serious computer. Letter Perfect and Data Perfect are written in versions which support the BIT 3 board.

BIT BLITTING — The Atari computer has the capability of moving images around the screen by simply changing a few registers (memory locations). This is the technique of player movement. Other computers (like the Apple II) require moving large amounts of data through RAM in order to get movement. Animation requires rewriting the data to get images to change on the screen. Since a high resolution screen can use over 8K, this activity can be quite slow. This technique of changing large amounts of screen memory during a program operation is called BIT BLITTING. Many programs translated from Apple to Atari code use this technique. A rippling effect is often seen if large images are scrolled across the screen.

BIT MAP — A BIT MAP is a representation of a graphical image in the memory of a computer. The display is comprised of picture elements (pixels), which are the smallest addressable units on a screen. Bit mapping is a technique where one bit (binary element) can be on or off in memory and the corresponding pixel on the monitor will be on or off respectively. The Atari uses different amounts of memory for the screen depending on the resoultion of the graphics mode used. The address of the start of the screen memory bit map can be found in locations 88 and 89 ($58 and $59). The location found in this address is the start of screen memory which maps out to the upper left corner of the screen. The end or top of screen memory depends on the resolution which determines the number of bytes used.

BLINKING CURSOR — A machine language program which will cause the cursor and any inverse characters to blink is printed below. The program fits into page 6 (location 1536) and is initialized by a USR(1536) statement. This program is taken from the Portland Atari Club newsletter.

```
100 FOR X=1536 TO 1567
110 READ Y
120 POKE X,Y:NEXT X
130 DATA 104,162,6,160,11,169,6,32,92,
228,96,165,20,110,243,2
140 DATA 110,243,2,106,106,106,106,46,
243,2,46,243,2,76,95,228
150 ? USR(1536)
```

BOLDFACE — Many printers have the option to emphasize print by double striking or overlapping dots while printing. The BOLDFACE type will appear darker or heavier than normal type in order to draw attention. See the printer conversion tables in the appendix for your printer code.

BOOKKEEPER — This software package from Atari is suitable for a very small business provided you are familiar with basic accounting principles. Journal entries are required and balance sheets and income statements can be generated at appropriate intervals. Since the program is written for one disk drive, a lot of disk swapping is required to fully use this package. Atari, Inc.

BOOLEAN OPERATORS — Atari BASIC uses a logical operation developed in the 19th century by mathematician George Boole. The Boolean is a statement or comparison set off by parentheses, which, if true, takes the value of one. If the comparison is not true then it takes the value of zero. You may have seen some BASIC programs with Booleans and assumed that they were typos. Documentation on Boolean operators is very skimpy in the *Atari BASIC Manual.* The following example will help clarify their usage.

Assume that a variable called FLAG has a current value of 1.

```
FLAG = 1
```

To test the value of FLAG, you can use a BASIC statement such as:

```
IF FLAG =1 THEN 100
```

The Boolean test is true if FLAG equals one (1), so you do not need the "= 1" component.

```
IF FLAG THEN 100 is equivalent.
```

Another more practical use is to read the joystick. The STICK command returns a value which depends upon the position of the joystick when the command is executed.

You can use Boolean logic to read the STICK and to increment or decrement some other value accordingly.

```
10 A = STICK(0)
20 B = B + ((A=6)+(A=7)+(A=5))-((A=10)+(A=11)+(A=9))
30 C = C + ((A=10)+(A=14)+(A=6))-((A=9)+(A=13)+(A=5))
```

Moving the stick down and right will give a value of 5 to A. In this case the values of B and C will be:

```
B=B+(0+0+1)-(0+0+0)=B+1
C=C+(0+0+0)-(0+0+1)=C-1
```

In this case, B is left and right, and C is up and down. The cursor will be moved one to the right (+1) and one down (–1).

BOOT DISK FORMAT — When you turn on your Atari computer with your disk drive on line, the operating system looks for a BOOT DISK FORMAT. The OS looks for the BOOT SECTORS and then for either an AUTORUN.SYS file or it goes to look for the data called for in the BOOT SECTORS. The AUTORUN.SYS file may call in a BASIC program to run. A BINARY LOAD file may be renamed AUTORUN.SYS and it will boot automatically when it is present on a disk with DOS.SYS and the INIT and RUN addresses are appended.

BOOT ERROR — BOOT ERROR is the message you will get if you try to start up your Atari system with your disk drive turned on and connected and either NO DOS.SYS on the disk in the drive OR without a properly set up auto boot disk in the drive. Most commercial games use an auto boot format. (See BOOT SECTORS.) One other possibility is that your drive has been physically damaged by a fall and it is out of alignment. If you suspect that this is the case, try to boot up a disk which you are sure has a proper boot sector format. You can usually recover a DOS disk which gives a BOOT ERROR by using the H option on the DOS menu. This will rebuild the boot sectors and allow DOS to find the other files on the disk through the directory.

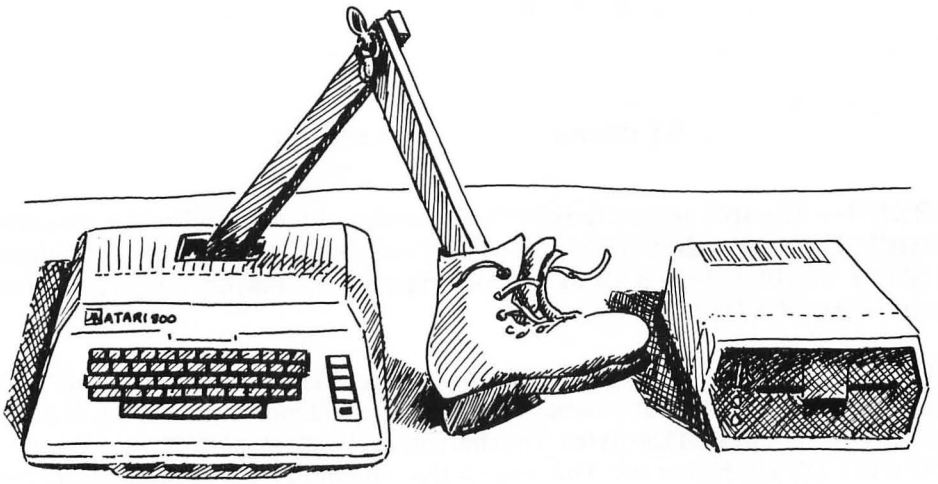
BOOT FILE — A BOOT FILE is a disk or cassette file which loads directly into memory on power-up of your system. A BOOT FILE does not need DOS to begin loading in as the routines are contained in the OS ROM to start loading. The BOOT FILE does not need a directory entry as does a DOS file. You cannot copy a BOOT FILE with DOS nor can you load in the file using the L option from DOS. See BOOT SECTORS for a description of how the boot record is constructed.

BOOT SECTORS — DOS 2.0S uses three boot sectors. When the computer is turned on (the disk drive must have been turned on previously) the first sector on the disk is read and several bytes are sought. Byte 0 is a flag. A zero (0) in byte 0 means that this is a boot file. Byte 1 contains the number of sectors to load. Bytes 2 and 3 contain the address of the boot code. Bytes 4 and 5 are the initial address of the program to be loaded. This can be anywhere except where a conflict with the Operating System may occur. Byte 6 is a machine code to be executed after the boot is loaded. The OS contains

the program to read these bytes so this instruction comes from the OS ROM. Since byte 1, the number of sectors to read, is limited to the range 0 to 255, only 255 × 128 bytes (32K) can be booted from this boot program. This means that an additional program is needed to load in longer code. After sector 1 is read, sectors 2 and 3 can be read giving additional instructions on where to go to find the application program. On an Atari DOS disk, sectors 2 and 3 contain information needed to go find and load the DOS. They could be pointed to a game or a word processor.

SECTOR 0 BOOT RECORD

BYTE	DESCRIPTION
0	Flag-0=Boot file
1	Number of sectors to load for boot
2	LO address of place to start loading file in memory
3	HI address of place to start loading file in memory
4	LO address of initialization (SYSTEM RESET jumps here)
5	HI address of initialization



BOOT-UP PROCEDURE — The Atari computer initializes when you turn on the power. The program coding for the initialization procedure is in the Operating System ROM and can be called from software as well as by turning the power off and back on again.

A RESET interrupt is started by bringing the RESET pin on the 6502 to a low state. This causes control to go to the initialization subroutine. The RESET goes through locations 65532 and 65533 ($FFFC and $FFFD) to the address of the subroutine (58487 of $E477). This is the start of the initialization code (finally). There are several ways to

simulate turning the switch off and on to start a BOOT-UP. Usually though, you need it when the program has crashed and you do not have the ability to enter data. To do a BOOT-UP:

1. In BASIC, type ? USR(58487)
2. In DUP.SYS, use M (run at address) and type E477 for the address
3. In Assembler/Editor, G E477

You can do a hard RESET by modifying your 800. This operation is only for real hardware hackers. Atari originally planned for this reset switch, but it was never implemented. Take your 800 apart and look at the motherboard. In the center of the board you will find a row of resistors. The last one is marked R156. There are two empty solder through-holes next to R156, and this is where the reset switch goes. Get a small momentary switch and a 1/4W 47 ohm resistor and put them in series across these two empty holes. This is the hardware reset which will bring you out of any crash. All memory will be wiped out, just as in a coldstart.

47 ohms **switch**

BORDER — The area around the playfield near the edge of the television screen is the BORDER. The color of the BORDER is usually controlled by color register 4 in location 712 ($2C8). The BORDER is set to black upon power-up. Players and missiles cannot move into the BORDER.

BOUNDARY — Some procedures require that all data remain within certain limits or a discontinuity will exist. For example, the character set is contained in 1K of memory, consisting of four pages of 256 bytes. The character set data must start on a 1K boundary if it is a full 128 character set. This means that the high byte of the address for the character set must end in a 0, 4, 8, or C. There is no low byte for the character set address since a 1K boundary will not have any component in the 0 to 255 range (just 1024, 2048, ...).

BPUT — In BASIC XL, BPUT is used to output a specified number of bytes from a specified address through an OPEN channel and device. The bytes which comprise the screen memory can be saved to disk by using BPUT. The format for using BPUT is:

```
BPUT #iocb, addr, len
```

where #iocb is the OPEN channel, addr is the starting address in memory of the data, and len is the length of the block to transfer.

BRANCH — BRANCHing is the transfer of the program flow to another part of the program based on the outcome of some test. In BASIC, IF and ON commands will BRANCH to another line number if some condition (such as a Boolean operation) is true. These are called conditional BRANCHes. GOSUB and GOTO send the program to another line number under any condition and are called unconditional BRANCHes.

In machine language there are eight different branch instructions which can go forward or backward in memory by a number of location steps. The BRANCH instructions are: BCC-Branch on Clear Carry; BCS-Branch on Carry Set; BEQ-Branch on result EQual to zero; BMI-Branch on result MInus; BNE-Branch on result Not Equal to zero; BPL-Branch on result PLus; BVC-Branch on oVerflow Clear and BVS-Branch on oVerflow Set.

BREAK — Pressing the BREAK key interrupts the program being executed unless the BREAK key has been specifically disabled. In a BASIC program, typing CONT (RETURN) for CONTinue will usually resume operation unless the program has modified itself by deleting lines. The display screen will probably be disturbed by the STOP and CONT messages. If no valuable data will be lost you can start over by typing RUN to rerun the program. The following subroutine can be used to disable the BREAK key. Several things will re-enable the BREAK after you do this once, so it is best to make this routine a subroutine, and GOSUB to it often. A GRAPHICS command, OPENing a S: or E: device, SYSTEM RESET, or PRINT to the Screen will require re-disabling.

```
0 REM ** BREAK KEY DISABLE SUBROUTINE
2 REM ** USE GOSUB 32000
4 REM
32000 BRKDIS=PEEK(16)-128
32100 IF BRKDIS<0 THEN RETURN
32200 POKE 16,BRKDIS
32300 POKE 53774,BRKDIS:RETURN
```

BUFFER — A section of memory used for temporary storage of data. The buffer can be written to and read from. An external buffer can be used as a printer spooler. The spooler is a device which will accept data headed for the printer. The spooler reads very rapidly and can output at a rate nearer to the print speed 30 to 160 characters per second. The computer can output characters at 1,920 characters per second.

BUG — The command in the Atari Assembler/Editor cartridge to enter the debug mode is BUG. The response on the screen to the BUG command will be DEBUG at which time you can begin tracing, stepping through, and testing the program.

BUGS — A bug is an error in logic or structure of a program. The BASIC cartridge and 10K Operating System cartridge are programs which reside in ROM and can only be changed or debugged by changing the ROM chips. Atari, Inc. has provided a Revision B set of ROMs for the Operating System and the Rev. B corrects a few of the bugs. The BASIC cartridge has a few known bugs which may affect your programming. A new Revision C of the BASIC cartridge should fix most of these bugs.

1. LOG(Ø), CLOG(Ø), LOG(1), / and CLOG(1) will produce erroneous results. Almost all higher level functions will produce an approximation only because of the polynomial expansion algorithm in the floating point program.

2. The BASIC cartridge sometimes locks up during line editing.

3. A string of exactly 256 bytes will sometimes end up in a location not expected if it is moved.

4. An INPUT without a variable does not return an error when interpreted.

5. PRINT X=NOT Y will surrender control of the keyboard (lockup!).

6. Loops with LPRINT commands cannot be interrupted by BREAK.

7. A blank is usually not a problem in Atari BASIC line except when placed between a DIMmed variable and the parentheses containing the array dimension.

8. Control-R and Control-U print out as a semicolon.

The OS has a few bugs. The most bothersome is the "going to sleep" problem found in older machines with the Revision A OS chips. During disk input or output and printer output, the system will often take a nap which lasts from five seconds up to many minutes. It will then wake up and continue where it left off. Hitting the BREAK key will often wake up the machine unless it has been disabled. Part or all of a printed line may be reprinted when the computer wakes up.

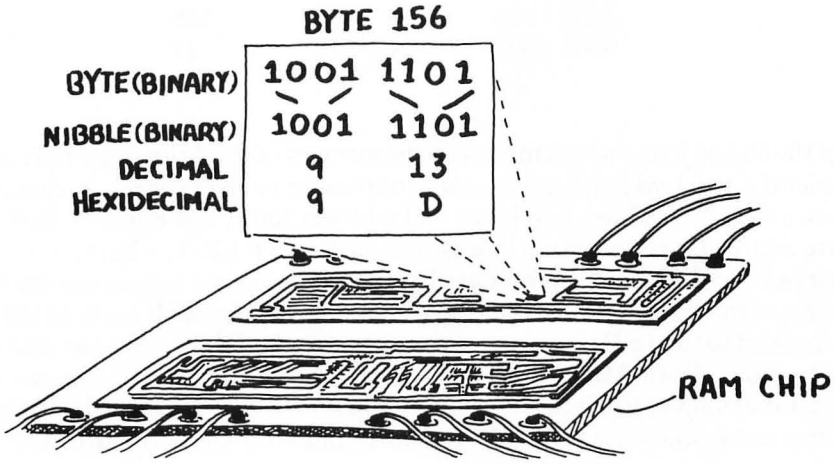
Another bug involves the RS 232 handler and the DOS. Normally, output can be sent to any output device: E:, D:, P:, S:, C:, or R:. For example, while using the DUP.SYS menu part of the DOS, directories can be sent to the screen or printer. The R: device should be able to accept output also but a bug makes the DUP.SYS wipe out the RS 232 handler when it is loaded. Therefore, no output to the R: device is possible from the DOS menu.

BULLETIN BOARD — A revolution is sweeping the country because of the availability and low prices of computers and communications devices. BULLETIN BOARDs are a major part of this revolution. A BULLETIN BOARD is a computer with a BULLETIN BOARD program, auto-answer modem, and a telephone line. Many users groups have set up BULLETIN BOARDs as a central exchange for ideas and public domain software. Messages are typed in by callers and are stored on the BBS disk to be read by one or all of the future callers. Other uses are for uploading or downloading programs. Uploading involves sending a BASIC LISTed program to the BBS for others to read or receive. Downloading involves taking a copy of the program through the telephone line and modem and saving it on tape or disk at your own location. There are over 1,000 BULLETIN BOARDs around the country and the number is growing fast. One caution, turnover is high on the BULLETIN BOARD telephone numbers and if a number is reassigned to an innocent non-computer person, many deep slumbers may be broken by phone calls with only a high-pitched squeal on the calling end. Try to be careful and verify your numbers. See AMODEM and AMIS.

BUSY LIGHT — The upper light on the Atari 810 disk drive is the BUSY LIGHT. The light indicates that the motor is turning. A disk which is removed before the BUSY LIGHT goes out can be damaged by not closing any open files or updating the VTOC. It is also possible to scratch the recording media by removing a disk while the BUSY LIGHT is on (but everybody does it anyway!).

BYE — This little used BASIC command sends you on a trip to the MEMO PAD mode in which no processing can be done. SYSTEM RESET will return control to BASIC. One use for the BYE and MEMO PAD mode is to use a screen dump utility, such as Printwiz from Allen Macroware. Screens can be made up using the keyboard editor in the MEMO PAD mode without getting ERROR messages. The Screen dump can then be activated to print out the screen for a quick label maker.

BYTE — A single group of eight bits is a byte. The Atari computer memory is organized as a series of bytes numbered from 0 to 65,535. Some of these locations will not be useable, depending on the amount of RAM you have installed. Disecting a byte we find eight bits traditionally designated D0 through D7.



D7 D6 D5 D4 D3 D2 D1 D0

Each of the eight locations may contain a 1 or a 0. If all the locations contained a 1 then the byte would have a value of 255 (128+64+32+16+8+4+2+1). See the discussion of BIT. When the value of a particular byte is displayed or printed, it appears as a hexadecimal pair of numbers. The hexadecimal system has 16 digits- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Each of these hex digits can be represented by four bits.

0 0 0 0	= 0	= 0
0 0 0 1	= 1	= 1
0 0 1 0	= 2	= 2
0 0 1 1	= 3	= 3
0 1 0 0	= 4	= 4
0 1 0 1	= 5	= 5
0 1 1 0	= 6	= 6
0 1 1 1	= 7	= 7
1 0 0 0	= 8	= 8
1 0 0 1	= 9	= 9
1 0 1 0	= A	= 10
1 0 1 1	= B	= 11
1 1 0 0	= C	= 12
1 1 0 1	= D	= 13
1 1 1 0	= E	= 14
1 1 1 1	= F	= 15

A number which is represented by FF in hexadecimal would be 1111 1111 in binary. (Hexadecimal is traditionally preceded by a dollar sign.) Here are some more examples of hex to binary conversions.

1111 1110	= $FE	= 254
1111 0000	= $F0	= 240
1000 1000	= $88	= 136
0001 0001	= $11	= 17

Each of the 65,536 bytes which makes up the memory of the Atari must have an address. Conveniently, two bytes will serve as an address for each of the bytes. Since each byte can have one of 256 values, two bytes will address 256 × 256 bytes or 65,636 bytes. A two byte address is represented in a format such as FF FF. The leftmost part is called the high byte or MSB (most significant byte). The rightmost byte is the the low byte or LSB (least significant byte). To get the decimal value, the MSB must be multiplied by 256 and added to the LSB. This is commonly done to find the value of an address stored in two locations. To find the value corresponding to the address of the start of the display list, for example, try `DLIST = PEEK(560) + 256 * PEEK(561)`. Unfortunately, the values are usually stored LSB then MSB, but we read them MSB then LSB.

BYTE COUNT — In the BASIC NOTE and POINT commands, the BYTE COUNT determines the position of the pointer within a sector of a formatted disk. The BYTE COUNT can range from 0 to 124.

C

C — C is a high level, structured programming language. Bell Laboratories produced C and it is renowned for being the language in which the UNIX operating system is written. Atari Program Exchange offers a C Compiler called Deep Blue C Compiler which is an implementation of Small C. A text editor is required to produce source code. Another C implementation is C/65 from OSS.

CABLE — CABLEs for printers and modems are a constant source of irritation and frustration to computer users. A CABLE is used to allow electrical signals to flow from your computer to another device such as a printer, modem, or monitor. Since most brands of computers are different from each other, you will need a special CABLE for your Atari computer. You will probably be too excited to take time to read your manual thoroughly to find the information to make your CABLE. It is usually not obvious exactly how to do it. Instead of paying $40 to $50, you can make one for about $10 (in most cases). Just connect the pins from the 850 interface or 5 pin DIN socket to the correctly numbered pins on the printer or modem. Use an ohmmeter to be sure you have continuity. Make sure to check your printer manual to be absolutely sure before you make the connections, because you could do some damage if you connect the pins incorrectly.

For Centronics interface (Epson, NEC, Okidata, Gemini, Prowriter) to the 850 parallel port, use a 15 pin "D" connector or D15, such as a Canon DB-15-P or AMP 205-206-1 and twelve conductor wire (don't make it over six feet long or you can mess up the signal timing), and an AMP Champ-36 connector or a 57-30360 36 pin connector to the Epson. Connect pins as follows:

850 interface pins

1	2	3	4	5	6	7	8	11	12	13	15

EPSON MX 80, GEMINI and MOST CENTRONICS INTERFACE PRINTERS

1	2	3	4	5	6	7	8	16	32	11	9

NEC

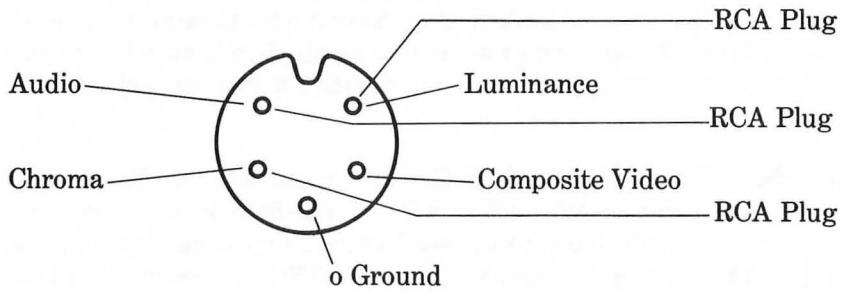
1	2	3	4	5	6	7	8	14	32	11	9

For a Hayes SmartModem to the 850 interface, use the following connections. Note that the modem connects to serial port #1 on the 850 and uses an 8 pin "D" connector

instead of the 15 pin as on the parallel port. The important modem connections are the signal in, signal out, and ground. Unless you are doing auto-answer or fancy "hand-shaking," you can probably get started with these three pins alone.

850 INTERFACE		HAYES SMARTMODEM		ANCHOR MARK 7	
pin	fn. dir.	pin	fn. dir.	pin	fn. dir.
1	DTR (out)				
2	CRX (in)	26			
3	RXD (in)	2	TXD (out)	2	TXD (out)
4	TXD (out)	3	RXD (in)	3	RXD (in)
5	Signal Ground	7	GND	7	GND
6	DSR (in)	22			
7	RTS (out)			25	CPC
8	CTS (in)				

To connect a monitor to your 800 or 800XL, make up a cable using a 5 pin DIN plug according to the following model. You will get the highest possible quality with this arrangement. Note that there are two types of 5 pin DIN plugs. One type has the pins arranged around a 180 degree arc. The other type spreads around 270 degrees. You want the 180 degree type.



DIN Jack looking AT the computer

You must run the ground to each of the RCA plugs. The ground is the bottommost pin and all can be connected together. For an 800 XL, use the Composite video and the Audio outputs and run them to the FRONT jacks of the Commodore 1701 or 1702 monitor. The quality will not be as high as the separate chroma and luma set-up, but it will be higher than a regular TV receiver. This is necessary because the chroma output is not connected to the DIN jack.

CALCULATION HEIRARCHY— Operators within algebraic expressions in Atari BASIC statements must be arranged or separated properly to give correct results. Expressions arranged within parentheses are said to be "nested." Expressions within the nested function are evaluated from left to right if all operators have the same precedence (like + and –), otherwise a specific order is followed. The order of precedence for the operators in algebraic statements is:

Relational operators such as <, >, =, <>, >=, <=
Minus sign (–)
Exponentiation (∧)
Multiplication and division (*,/)
Addition and subtraction (+,–)
Unary operator (NOT)
Binary operators (AND and OR)

How would you evaluate an expression like:

3 * 4 + 2

It could be 12 + 2 = 14 or 3 * 6 = 18. The rules above say that your computer would do the multiplication first and then the addition, so 14 is correct.

CAPS LOWR DISABLE — POKE 702,64 prevents the accidental switch to lower case characters. This is important as some programs will actually crash irrecoverably with an input of a lower case character.

CARRY FLAG — The Processor Status Register of the 6502 processor is an eight bit register with a series of flags. The flags contain values of 1 or 0 when certain conditions are met. There are seven flags, and the rightmost one, bit 0, is the CARRY FLAG.

N	V	*	B	D	I	Z	C

N = Negative
V = Overflow
* = Unused
B = Break Command
D = Decimal Mode
I = Interrupt Disable
Z = Zero
C = Carry

The CARRY FLAG is set (the value is one) when the ACCUMULATOR has a value greater than eight bits can hold (greater than 255). Two branches can be made when the CARRY FLAG changes: BCC and BCS. BCC is a Branch on Carry Clear which will branch to another part of memory when the CARRY FLAG is 0. BCS will branch to another part of memory when the CARRY FLAG is set to 1. The Assembler/Editor cartridge can be used to monitor the Processor Register Status in the TRACE mode.

CARTRIDGES — The Atari 800 has two slots for plug-in ROM (Read Only Memory) CARTRIDGEs. All other Atari computers have one slot. The ROM CARTRIDGEs contain machine language programs which reside in memory. The left CARTRIDGE (Cartridge A) occupies 8K of memory between 40960 and 49151 ($A000 to $BFFF). If no CARTRIDGE is inserted, this memory is free for other use. The left slot is most often used. The right slot uses memory between 32768 and 40959 ($8000 and $9FFF). A single 16K CARTRIDGE can use this area in combination with the area for the left slot

by bank selecting the addresses from the left slot. This is the technique used by OS, Inc. for BASIC XL. At the top end of the CARTRIDGE memory, there must be six bytes to tell the OS where to jump and where the program loads. These six bytes define four items:

DATA STORED AT LOCATIONS

ITEM	LEFT CART	RIGHT CART
Run address	$BFFA,B (49146,7)	$9FFA,B (40954,5)
Cartridge-in byte	$BFFC (49148)	$9FFC (40956)
Cartridge option	$BFFD (49149)	$9FFD (40957)
Initialization address	$BFFE,F (49150,1)	$9FFE (40958,9)

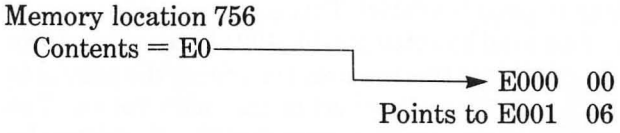
To jump into BASIC with the cartridge installed from DOS, for example, type M from the DOS menu. For the address, type BFF9 <Return> which is the INIT address contained in memory locations $BFFE and $BFFF on the CARTRIDGE. This is the same as using the B option, or RUN CARTRIDGE from DOS.

CASSETTE BOOT FILE — This BOOT FILE is designed to load directly into memory and to begin execution upon loading without any keyboard entries required. A CASSETTE BOOT FILE is loaded by holding the START key while turning on the computer.

CHAINING PROGRAMS — One BASIC program can be used to run another by including a statement such as RUN"D:NEXTPROG.BAS" in the first program. When the statement is executed, NEXTPROG.BAS will be LOADed and RUN and the original program will be erased from memory.

CHARACTER GRAPHICS — The character set in the Atari computer is changeable. A built-in set is available and is used by the system on start-up. See CHARACTER SETS. The character set is used for graphics modes 0, 1, and 2. Although these modes are often called text modes, character modes is more descriptive. The other graphics modes can only display blocks or pixels. The character set can be redefined as figures or objects and can even be changed during the program operation to achieve animation. Many famous games such as Shamus by Synapse and Space Invaders by Atari use constantly redefined characters for animation.

The operating system goes to a certain location in memory to look for the data to print a character on the screen. The beginning address of the character set data is stored in location 756 ($02F4). The normal start of the character set data is location 57344 ($E000). Four pages (1K) of memory are required for a GR.0 character set. Two pages (512 bytes) are needed for GR.1 and GR.2 character sets because no lower case is used. You can flip between character sets by POKEing CHBAS with the address of the new set. Each character is comprised of eight bytes of eight bits each.



Each character is comprised of 64 bits of information which tell every one of the pixels in an 8 x 8 block whether to be on or off. The pointer in 756 points to the first byte of the character set data and every eighth byte is the start of a new character. You can change the memory to which the pointer points by POKEing location 756 with another number. Try the following experiment in BASIC. Type POKE 756,Ø. This will make the data in page 0 become the character set data. The screen will be filled with some pattern which is not the character set page 0 is used for the Operating System. Type the double quote sign (") and see what happens. This character happens to be composed of some of the memory locations which include the internal clocks (18,19 and 20). The character will sparkle and blink. SYSTEM RESET will return the computer to normal.

To build your own characters you must reserve a part of RAM exclusively for your character set. The best way to do this is to find the top of free RAM and back it down a few pages. The value of RAMTOP is stored in location 106. This is the location of the first location of memory which is ROM of the Operating System. The memory for the screen is directly below the top of RAM and the display list is directly below this. You can find the address of this memory boundary and set the RAMTOP 1K lower to save room for the new character set. The easiest way to build the actual characters is to use a commercial utility such as Instedit from APX. The following program will copy the resident character set into an area below the normal RAMTOP. RAMTOP is actually moved by the first line of the program.

```
100 RT=PEEK(106):POKE 106,RT-4:REM Look
at RAMTOP and reduce it by 4
110 GRAPHICS 0:LET NEWSET=256*(RT-4):REM
 Define NEWSET as address of new charact
ers
120 FOR X=0 TO 1023:REM Start loop to co
py set
130 POKE NEWSET+X,PEEK(57344+X):REM Get
old characters
140 NEXT X:REM End loop
150 POKE 756,NEWSET/256:REM Point charac
ter pointer to new data
```

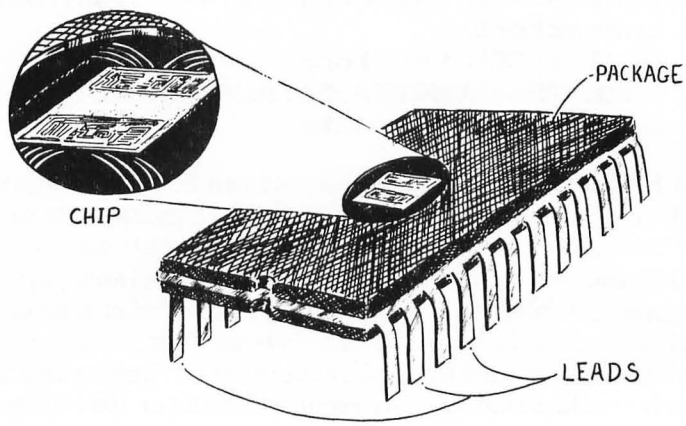
CHARACTER SETS — Custom character sets can be used on Atari computers. Many high quality commercial games use redefined character sets to achieve high speed, high resolution animation. Characters are made up of blocks of 8×8 pixels. Each line of pixels is represented by an eight bit byte. See *De Re Atari*, page 3-4, for a description of this format. A full character set is comprised of 256 characters, but half are simply inverse characters of the lower half. The inverse video function is set by putting a 1 in the high bit of the character byte. Since there are 128 unique characters and we need eight bytes for each, a character set requires 8×128 or 1024 bytes. The built in

Atari character set is in ROM starting at page 224 ($E0). This corresponds to memory location 57344 ($E000). A new set can be used by setting aside 1024 bytes of your own memory, copying or modifying the Atari set into this area, and changing the pointer to the set. The pointer points to the high byte (page number) of the chatacter set. The pointer is at 756 ($2F4). Instedit from APX, Graphics Generator (see The Next Step) by Datasoft, and other utilities are available to help you design your own character sets.

CHBAS — This pointer is the CHaracter BASe register and it is located at memory location 756 ($2F4). It contains the page number (high byte) of the start of the character set data. On starting up the system, CHBAS contains a 224 ($E0) pointing to the character set data from location 57344 on. Four pages (1024 bytes) are needed for a full character set. This means you must reserve at least 1K of memory for a new set and the data must reside with 1K byte boundaries (that is, not spread over five pages).

CHECKSUM — A CHECKSUM is a technique for error checking whereby a series of numbers are added and the total compared against a known correct value. This is often done to verify typing in published BASIC programs. The TYPO program used by Antic Magazine uses a CHECKSUM technique. Tokenized values for a small range of line numbers are added and compared against a those of a program which is known to run correctly. The cassette recorder uses a checksum byte which is the sum of all of the bytes in a record. This byte is recorded on the tape and it is calculated by the computer when data are received. If the two sums are different an error is generated. The error may come from defective tape, noise or hardware malfunction.

CHIP — CHIP is an informal way of referring to an integrated circuit such as a microprocessor. The 6502 is a CHIP. The actual CHIP is a small slab of silicon which has been processed to facilitate very tiny electrical devices on the surface. The devices process data by retrieving, shifting, and sending signals out to other chips. RAM, ROM, EPROMs, EAROMs, POKEY, ANTIC, and PIA are all CHIPs. The devices are actually very delicate and sensitive to the environment and they must be packaged in a sturdy housing. The familiar black plastic packages with metallic leads (similar to a centipede) are called Dual Inline Packages or DIPs. A tip for those with a computer malfunction is to open up your system and examine every CHIP. Sometimes they work loose from their sockets and cause trouble. If a CHIP is not seated firmly in its socket, press it firmly back into place.



**CHR$** — This BASIC command is used to send an ATASCII character to a device (screen, printer, etc.) when the equivalent decimal value is given. Do not forget to add the dollar sign ($) to the CHR. If you do forget, your BASIC program will just be using another variable. This common error is very difficult to track down. To see what the uppercase letter set looks like, RUN the following program:

```
10 FOR X=65 TO 90: PRINT CHR$(X): NEXT X
```

Type in the program and then type RUN.

CHR$(125) — The "CLEAR SCREEN" symbol is an inverse upward-left curving arrow in double quotes and it is often printed out as a bracket on many printers. A clearer way of including a CLEAR SCREEN command in your BASIC program is to substitute:

```
PRINT CHR$(125);
```

for the Escape Shift-Clear key sequence.

CIO — Central Input/Output. The CIO is the routine contained in the Operating System which sends all input and ouput data to the correct device handler and then gives control to the HANDLER. This involves control of the computer at the very lowest level such as reading a key from the keyboard, finding a character for the key, and sending it to the monitor or screen for presentation to your eye (and brain).

CLEAR — In Microsoft BASIC II, CLEAR resets all variables, arrays, and strings. This is the same as the CLR function in Atari BASIC.

CLEAR DECIMAL FLAG — One of the most frustrating bugs a programmer runs into is the CLEAR the DECIMAL FLAG bug. The 6502 Processor Status Register has seven flags. One of them is the DECIMAL FLAG. When you are doing arithmetic in binary (or hexadecimal), you are in the HEX mode. When you are doing floating point arithmetic in BASIC, you are in the decimal mode (the flag is set or D=1). If you have a machine language routine in a BASIC program, and the machine language program must do arithmetic, and it expects HEX numbers, your program will crash — sometimes. The best way to prevent this is to start your ML routine off with a CLD instruction to CLEAR the DECIMAL mode FLAG.

CLEAR SCREEN BEFORE NEXT PRINT STATEMENT — POKE 87,1 will CLEAR THE SCREEN BEFORE the NEXT PRINT STATEMENT is executed.

CLEAR STACK — In Microsoft BASIC II, CLEAR STACK resets all time dependent entries, such as AFTER.

CLOAD — CLOAD will transfer a CSAVEd BASIC program from cassette tape into the BASIC free memory area. A LISTed program will not work with CLOAD.

CLOG — CLOG returns the log base 10 of the argument in parentheses.

EXAMPLE: PRINT CLOG(100) will print a 2.

CLOSE (CL.) — CLOSE is the command to close an IOCB which has been OPENed for input or output by a previous statement. If you try to OPEN a particular channel which is already OPENed, you will get an error message. The format to CLOSE an IOCB is simply: CLOSE #X, where X is the OPENed IOCB. An END statement CLOSEs all OPENed channels.

CLR — The CLR command in BASIC does not clear the screen. It is used to clear the values of all numeric variables. It also frees up the memory used by DIMensioning strings and arrays. The names of the variables remain in the VARIABLE NAME TABLE. CLR also executes a RESTORE command which sets the pointer to the DATA statements to the beginning.

COARSE SCROLLING — COARSE SCROLLING is the movement of data across the screen by shifting it one byte at a time. By adding or subtracting 1 from the address specified in the Load Memory Scan instruction in the display list, the screen data will move horizontally or vertically one character or byte at a time. This contrasts with smooth scrolling where the image is moved one bit at a time. Horizontal and vertical scroll enable registers must be enabled for smooth or fine scrolling.

CODE — CODE is the program which comprises a software package. CODE is computer jargon for the program written in some non-English language.

CODE (verb) — To CODE a program is to write it in computer language.

CODE CONVERSIONS — The following table is used to convert decimal to hexadecimal or binary. Also included are the keying sequences to produce the character which represents the value. The assembler instructions for all addressing modes of the 6502 processor are also included. This table is a valuable reference tool for machine language programming and for examining and changing register values. The lower case *c* means CTRL, the *s* means SHIFT, and the hyphen (-) indicates an inverse character generated by the ATARI (FUJI) symbol key. The *e* indicates an ESCAPE key sequence.

CODE CONVERSION TABLE

DEC	HEX	BINARY	KEY	Assembler
0	0	00000000	*c*,	BRK
1	1	00000001	*c*A	ORA ind X
2	2	00000010	*c*B	
3	3	00000011	*c*C	
4	4	00000100	*c*D	
5	5	00000101	*c*E	ORA 0pg
6	6	00000110	*c*F	ASL 0pg
7	7	00000111	*c*G	

DEC	HEX	BINARY	KEY	Assembler
8	8	00001000	*c*H	PHP
9	9	00001001	*c*I	ORA imm
10	A	00001010	*c*J	ASL acc
11	B	00001011	*c*K	
12	C	00001100	*c*L	
13	D	00001101	*c*M	ORA abs
14	E	00001110	*c*N	ASL abs
15	F	00001111	*c*O	
16	10	00010000	*c*P	BPL rel
17	11	00010001	*c*Q	ORA ind Y
18	12	00010010	*c*R	
19	13	00010011	*c*S	
20	14	00010100	*c*T	
21	15	00010101	*c*U	ORA 0pg X
22	16	00010110	*c*V	ASL 0pg X
23	17	00010111	*c*W	
24	18	00011000	*c*X	CLC
25	19	00011001	*c*Y	ORA abs Y
26	1A	00011010	*c*Z	
27	1B	00011011	*ee*	
28	1C	00011100	*ec*–	
29	1D	00011101	*ec*=	ORA abs X
30	1E	00011110	*ec*+	ASL abs X
31	1F	00011111	*ec**	
32	20	00100000	SPA	JSR abs
33	21	00100001	!	AND ind X
34	22	00100010	"	
35	23	00100011	#	
36	24	00100100	$	BIT 0pg
37	25	00100101	%	AND 0pg
38	26	00100110	&	ROL 0pg
39	27	00100111	'	
40	28	00101000	(	PLP
41	29	00101001	)	AND imm
42	2A	00101010	*	ROL acc
43	2B	00101011	+	
44	2C	00101100	,	BIT abs
45	2D	00101101	–	AND abs
46	2E	00101110	.	ROL abs
47	2F	00101111	/	
48	30	00110000	0	BMI rel
49	31	00110001	1	AND ind Y
50	32	00110010	2	
51	33	00110011	3	
52	34	00110100	4	
53	35	00110101	5	AND 0pg X
54	36	00110110	6	ROL 0pg X

DEC	HEX	BINARY	KEY	Assembler
55	37	00110111	7	
56	38	00111000	8	SEC
57	39	00111001	9	AND abs Y
58	3A	00111010	:	
59	3B	00111011	;	
60	3C	00111100	<	
61	3D	00111101	=	AND abs X
62	3E	00111110	>	ROL abs X
63	3F	00111111	?	
64	40	01000000	@	RTI
65	41	01000001	A	EOR ind X
66	42	01000010	B	
67	43	01000011	C	
68	44	01000100	D	
69	45	01000101	E	EOR 0pg
70	46	01000110	F	LSR 0pg
71	47	01000111	G	
72	48	01001000	H	PHA
73	49	01001001	I	EOR imm
74	4A	01001010	J	LSR acc
75	4B	01001011	K	
76	4C	01001100	L	JMP abs
77	4D	01001101	M	EOR abs
78	4E	01001110	N	LSR abs
79	4F	01001111	O	
80	50	01010000	P	BVC rel
81	51	01010001	Q	EOR ind Y
82	52	01010010	R	
83	53	01010011	S	
84	54	01010100	T	
85	55	01010101	U	EOR 0pg X
86	56	01010110	V	LSR 0pg X
87	57	01010111	W	
88	58	01011000	X	CLI
89	59	01011001	Y	EOR abs Y
90	5A	01011010	Z	
91	5B	01011011	*s*,	
92	5C	01011100	*s*+	
93	5D	01011101	*s*.	EOR abs X
94	5E	01011110	*s**	LSR abs X
95	5F	01011111	*s*–	
96	60	01100000	*c*.	RTS
97	61	01100001	a	ADC ind X
98	62	01100010	b	
99	63	01100011	c	
100	64	01100100	d	
101	65	01100101	e	ADC 0pg

DEC	HEX	BINARY	KEY	Assembler
102	66	01100110	f	ROR 0pg
103	67	01100111	g	
104	68	01101000	h	PLA
105	69	01101001	i	ADC imm
106	6A	01101010	j	ROR acc
107	6B	01101011	k	
108	6C	01101100	l	JMP ind
109	6D	01101101	m	ADC abs
110	6E	01101110	n	ROR abs
111	6F	01101111	o	
112	70	01110000	p	BVS rel
113	71	01110001	q	ADC ind Y
114	72	01110010	r	
115	73	01110011	s	
116	74	01110100	t	
117	75	01110101	u	ADC 0pg X
118	76	01110110	v	ROR 0pg X
119	77	01110111	w	
120	78	01111000	x	SEI
121	79	01111001	y	ADC abs Y
122	7A	01111010	z	
123	7B	01111011	*c*;	
124	7C	01111100	s=	
125	7D	01111101	*es*<	ADC abs X
126	7E	01111110	*e*BS	ROR abs X
127	7F	01111111	*e*TAB	
128	80	10000000	-*c*,	
129	81	10000001	-*c*A	STA ind X
130	82	10000010	-*c*B	
131	83	10000011	-*c*C	
132	84	10000100	-*c*D	STY 0pg
133	85	10000101	-*c*E	STA 0pg
134	86	10000110	-*c*F	STX 0pg
135	87	10000111	-*c*G	
136	88	10001000	-*c*H	DEY
137	89	10001001	-*c*I	
138	8A	10001010	-*c*J	TXA
139	8B	10001011	-*c*K	
140	8C	10001100	-*c*L	STY abs
141	8D	10001101	-*c*M	STA abs
142	8E	10001110	-*c*N	STX abs
143	8F	10001111	-*c*O	
144	90	10010000	-*c*P	BCC rel
145	91	10010001	-*c*Q	STA ind Y
146	92	10010010	-*c*R	
147	93	10010011	-*c*S	
148	94	10010100	-*c*T	STY 0pg X

DEC	HEX	BINARY	KEY	Assembler
149	95	10010101	-*c*U	STA 0pg X
150	96	10010110	-*c*V	STX 0pg Y
151	97	10010111	-*c*W	
152	98	10011000	-*c*X	TYA
153	99	10011001	-*c*Y	STA abs Y
154	9A	10011010	-*c*Z	TXS
155	9B	10011011	RET	
156	9C	10011100	*es*BS	
157	9D	10011101	*es*>	STA abs X
158	9E	10011110	*ec*TAB	
159	9F	10011111	*es*TAB	
160	A0	10100000	-SPA	LDY imm
161	A1	10100001	-!	LDA ind X
162	A2	10100010	-"	LDX imm
163	A3	10100011	-#	
164	A4	10100100	-$	LDY 0pg
165	A5	10100101	-%	LDA 0pg
166	A6	10100110	-&	LDX 0pg
167	A7	10100111	-'	
168	A8	10101000	-(	TAY
169	A9	10101001	-)	LDA imm
170	AA	10101010	-*	TAX
171	AB	10101011	-+	
172	AC	10101100	-,	LDY abs
173	AD	10101101	-–	LDA abs
174	AE	10101110	-.	LDX abs
175	AF	10101111	-/	
176	B0	10110000	-0	BCS rel
177	B1	10110001	-1	LDA ind Y
178	B2	10110010	-2	
179	B3	10110011	-3	
180	B4	10110100	-4	LDY 0pg X
181	B5	10110101	-5	LDA 0pg X
182	B6	10110110	-6	LDX 0pg Y
183	B7	10110111	-7	
184	B8	10111000	-8	CLV
185	B9	10111001	-9	LDA abs Y
186	BA	10111010	-:	TSX
187	BB	10111011	-;	
188	BC	10111100	-<	LDY abs X
189	BD	10111101	-=	LDA abs X
190	BE	10111110	->	LDX abs Y
191	BF	10111111	-?	
192	C0	11000000	-@	CPY imm
193	C1	11000001	-A	CMP ind X
194	C2	11000010	-B	
195	C3	11000011	-C	

DEC	HEX	BINARY	KEY	Assembler
196	C4	11000100	-D	CPY 0pg
197	C5	11000101	-E	CMP 0pg
198	C6	11000110	-F	DEC 0pg
199	C7	11000111	-G	
200	C8	11001000	-H	INY
201	C9	11001001	-I	CMP imm
202	CA	11001010	-J	DEX
203	CB	11001011	-K	
204	CC	11001100	-L	CPY abs
205	CD	11001101	-M	CMP abs
206	CE	11001110	-N	DEC abs
207	CF	11001111	-O	
208	D0	11010000	-P	BNE rel
209	D1	11010001	-Q	CMP ind Y
210	D2	11010010	-R	
211	D3	11010011	-S	
212	D4	11010100	-T	
213	D5	11010101	-U	CMP 0pg X
214	D6	11010110	-V	DEC 0pg X
215	D7	11010111	-W	
216	D8	11011000	-X	CLD
217	D9	11011001	-Y	CMP abs Y
218	DA	11011010	-Z	
219	DB	11011011	-*s*,	
220	DC	11011100	-*s*+	
221	DD	11011101	-*s*.	CMP abs X
222	DE	11011110	-*s**	DEC abs X
223	DF	11011111	-*s*—	
224	E0	11100000	-*c*.	CPX imm
225	E1	11100001	-a	SBC ind X
226	E2	11100010	-b	
227	E3	11100011	-c	
228	E4	11100100	-d	CPX 0pg
229	E5	11100101	-e	SBC 0pg
230	E6	11100110	-f	INC 0pg
231	E7	11100111	-g	
232	E8	11101000	-h	INX
233	E9	11101001	-i	SBC imm
234	EA	11101010	-j	NOP
235	EB	11101011	-k	
236	EC	11101100	-l	CPX abs
237	ED	11101101	-m	SBC abs
238	EE	11101110	-n	INC abs
239	EF	11101111	-o	
240	F0	11110000	-p	BEQ rel
241	F1	11110001	-q	SBC ind Y
242	F2	11110010	-r	

DEC	HEX	BINARY	KEY	Assembler
243	F3	11110011	-s	
244	F4	11110100	-t	
245	F5	11110101	-u	SBC 0pg X
246	F6	11110110	-v	INC 0pg X
247	F7	11110111	-w	
248	F8	11111000	-x	SED
249	F9	11111001	-y	SBC abs Y
250	FA	11111010	-z	
251	FB	11111011	-*c*;	
252	FC	11111100	-*s*=	
253	FD	11111101	*ec*2	SBC abs X
254	FE	11111110	*ec*BS	INC abs X
255	FF	11111111	*ec*>	

Based on article in San Diego ACE Newsletter, June 1982 by Ron Miller.

COLDSTART — A COLDSTART is just like turning on the computer for the first time. This can be done while the system is operating by jumping to memory location 58487 ($E477) by a ?USR(58487) from BASIC, or using the M option in DOS and typing E477<Return>. The Operating System is initialized and any program in memory is lost and any AUTORUN.SYS program on the disk will be booted in. A warmstart is like pushing the SYSTEM RESET button and does not wipe out the program in memory. The OS is initailized. A warmstart is entered at 58484 ($$E474).

Another way to do a COLDSTART is to POKE 580,1 and press SYSTEM RESET. You could put this at the beginning of your BASIC program and prevent listing of the file. You must also disable the BREAK KEY to prevent BREAKing and LISTing.

Still another way to do a COLDSTART is to yank on the 6502 RESET pin. The hardware is in place on the Atari 800. See BOOT-UP for the COLDSTART switch installation procedure. This modification requires soldering and drilling on your computer.

COLLISIONS — Players and Missiles — One of the most important things that a good shoot 'em up space game must do is react quickly and accurately to the joystick trigger. When you shoot a missile from your space ship you expect to see the appropriate explosion and consequent destruction (when you are on target). This is the job of the COLLISION REGISTERS. There are 15 registers used for detecting up to 54 combinations of collisions. Everytime a player or missile touches a playfield or each other, one of the registers will be set. By reading the registers and checking the contents, some other activity can be started (such as an explosion). Location 53278 is used to clear all of the collision registers. See COLLISION REGISTER.

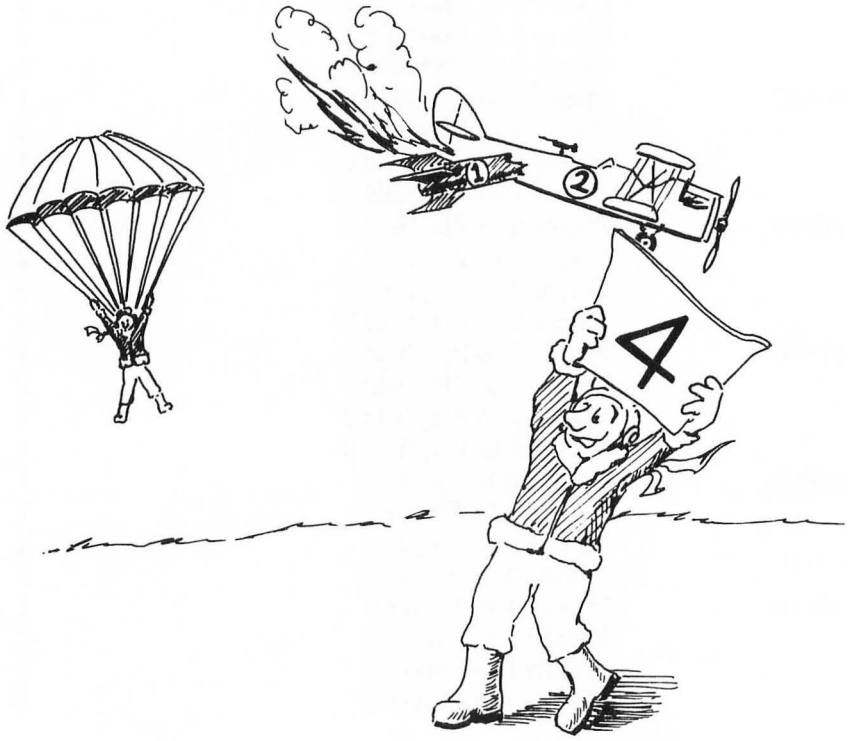
COLLISION REGISTER — A COLLISION is an event in which a player, missile, or playfield overlap or collide. The Atari computer uses hardware registers to monitor collisions. The COLLISION REGISTERs are located between 53249 and 53259 ($D001

and $D00B) in system memory. By checking the contents of the register, your program can tell exactly what kind of collision has occurred. These registers are used in the READ mode for collisions. For writing, they are used as horizontal position registers.

COLLISION REGISTER USAGE

Location	Collision	Register Value
53248 $D000	Missile 0 to Playfield 0	1
	Missile 0 to Playfield 1	2
	Missile 0 to Playfield 2	4
	Missile 0 to Playfield 3	8
53249 $D001	Missile 1 to Playfield 0	1
	Missile 1 to Playfield 1	2
	Missile 1 to Playfield 2	4
	Missile 1 to Playfield 3	8
53250 $D002	Missile 2 to Playfield 0	1
	Missile 2 to Playfield 1	2
	Missile 2 to Playfield 2	4
	Missile 2 to Playfield 3	8
53251 $D003	Missile 3 to Playfield 0	1
	Missile 3 to Playfield 1	2
	Missile 3 to Playfield 2	4
	Missile 3 to Playfield 3	8
53252 $D004	Player 0 to Playfield 0	1
	Player 0 to Playfield 1	2
	Player 0 to Playfield 2	4
	Player 0 to Playfield 3	8
53253 $D005	Player 1 to Playfield 0	1
	Player 1 to Playfield 1	2
	Player 1 to Playfield 2	4
	Player 1 to Playfield 3	8
53254 $D006	Player 2 to Playfield 0	1
	Player 2 to Playfield 1	2
	Player 2 to Playfield 2	4
	Player 2 to Playfield 3	8
53255 $D007	Player 3 to Playfield 0	1
	Player 3 to Playfield 1	2
	Player 3 to Playfield 2	4
	Player 3 to Playfield 3	8
53256 $D008	Missile 0 to Player 0	1
	Missile 0 to Player 1	2
	Missile 0 to Player 2	4
	Missile 0 to Player 3	8
53257 $D009	Missile 1 to Player 0	1
	Missile 1 to Player 1	2
	Missile 1 to Player 2	4
	Missile 1 to Player 3	8

Location	Collision	Register Value
53258 $D00A	Missile 2 to Player 0	1
	Missile 2 to Player 1	2
	Missile 2 to Player 2	4
	Missile 2 to Player 3	8
53259 $D00B	Missile 3 to Player 0	1
	Missile 3 to Player 1	2
	Missile 3 to Player 2	4
	Missile 3 to Player 3	8
53260 $D00C	Player 0 to Player 0	1
	Player 0 to Player 1	2
	Player 0 to Player 2	4
	Player 0 to Player 3	8
53261 $D00D	Player 1 to Player 0	1
	Player 1 to Player 1	2
	Player 1 to Player 2	4
	Player 1 to Player 3	8
53262 $D00E	Player 2 to Player 0	1
	Player 2 to Player 1	2
	Player 2 to Player 2	4
	Player 2 to Player 3	8
53263 $D00F	Player 3 to Player 0	1
	Player 3 to Player 1	2
	Player 3 to Player 2	4
	Player 3 to Player 3	8



COLOR — The value following COLOR (in the BASIC command) sets up the color register for use in the next DRAWTO or PLOT command. You must use a COLOR statement. If not, the PLOTting and DRAWingTO will be done in the background color and will not be visible. This is IMPORTANT! The color registers for playfields are located between 708 and 712 ($2C4 and $2C8). In non-graphics (text) modes 0, 1 and 2, the COLOR statement plots characters from the data in the color registers in the XY position specified by a PLOT statement. Registers use different numbers for COLOR for different graphics modes.

COLOR REGISTER NUMBER

	0	1	2	3	4
LOCATION (dec.)	708	709	710	711	712
GR. Modes					
3, 5 & 7	1	2	3	-	0
4, 6	1				0
0, 8 (LUM.only)	-	1	0	-	-
1, 2	-	-	-	Color of L.C. chars	-
10 (704=BCKGRND)					0

NUMBER TO BE USED IN COLOR STATEMENTS
FOR DIFFERENT GRAPHICS MODES

COLOR CLOCK — The Atari computer chip set uses the NTSC (National Television Standards Committee) standard color modulation frequency to generate all master timing. This frequency is 3.579545 (5 × 63 / 88) Megahertz. The period of this frequency is 280 nanoseconds. The Atari computer uses 228 of these cycles, or COLOR CLOCKS, to generate one horizontal line in the wide playfield mode. In the standard width playfield only 160 COLOR CLOCKS are visible. In GR.8 there are two pixels per COLOR CLOCK.

COLOR REGISTERS — There are nine COLOR REGISTERs for Atari computers. COLOR REGISTERs are memory locations which accept tint and brightness values and indirectly (but very quickly) change the player, playfield, or background of the screen. The COLOR REGISTERs are actually in GTIA hardware starting at location $D012 but the are changed by writing to the "shadows" at 704 through 712 ($2C0 through $2C8). (See COLOR and SETCOLOR). The following chart describes the COLOR REGISTERs.

COLOR REGISTER SHADOWS

OBJECT	ADDRESS	BASIC COMMAND to set the color	
PLAYER 0	704		
PLAYER 1	705		
PLAYER 2	706		
PLAYER 3	707		
PLAYFIELD 0	708	SETCOLOR 0	UPPER CASE IN GR.1&2
PLAYFIELD 1	709	SETCOLOR 1	ALSO LOWER CASE IN GR.1&2
PLAYFIELD 2	710	SETCOLOR 2	ALSO INVERSE/UPPER CASE
PLAYFIELD 3	711	SETCOLOR 3	ALSO INVERSE/LOWER CASE
BACKGROUND	712	SETCOLOR 4	BORDER

Colors numbered for 0 to 15 can be put into the register shadows listed above. Luminance values from 0 to 14 (even numbers only) can be added to the color values. This is done in BASIC through the SETCOLOR command. The COLOR command uses the color and luminance of the register to PLOT or DRAWTO in graphics modes.

COLOR REGISTER INDIRECTION — INDIRECTION is the term for the ability of the hardware to control the color, position, character set data, etc., by simply changing the value of one byte in one register (see REGISTER). The COLOR REGISTER is like a switch or control box which determines the color of all the screen data in a particular playfield. This is much easier and faster than changing all of the individual pixels or bits which represent the information in that screen data.

COLUMN — The characters on the television or monitor screen are arranged in COLUMNs and rows. There are 40 COLUMNs across the screen. The Atari computer is set up to start writing in the second COLUMN giving, in effect, a 38 COLUMN wide screen. To widen the screen to a full 40 COLUMNs, type in POKE 82,Ø. Memory location 82 holds the number of the starting COLUMN. Location 83 holds the data for the right-hand COLUMN limit.

COM — Undocumented BASIC command which works just like DIM.

COMMAND — A COMMAND is a statement in BASIC which performs some function.

COMMON — In Microsoft BASIC II, COMMON allows you to keep variable names and values the same in two different programs. COMMON works like DIM except the Variable Name Table is apparently maintained when another program is loaded in.

COMPILER — A COMPILER is a program which takes a high level language program, such as a BASIC program, and converts it through a series of steps to a machine language program. A machine language program is written so that the central microprocessor receives instructions directly. It is just a series of characters which look unintelligible to the novice, but which actually contains very simple and specific instructions for accomplishing some task. COMPILERS often take three or four passes through a program in order to completely compile the source program into machine

language. An additional program called a "run time package" is added to the original program during compilation to provide some additional instructions for the microprocessor. This will increase the length of short BASIC programs substantially. The benefit of a COMPILER is that the program can be written in BASIC, and it will run much faster, up to 100 times faster in some cases. Not every BASIC program can be compiled. Special care must be taken during programming to make the program as simple as possible.

COMPOSITE VIDEO — The signal which comes out of the 5 pin DIN plug on the Atari 800 is a COMPOSITE VIDEO. The COMPOSITE VIDEO signal contains the sum of the horizontal sync signal, the vertical sync signal, the luminance signal, and the color carrier (3.58 Megahertz, phase modulated). This signal is also added to the audio carrier (4.5 Megahertz, frequency modulated) which is used to modulate the amplitude of a radio frequency carrier at approximately 60 Megahertz (channel 2 or 3 on your TV) and is sent out the rear cable of the Atari computer to the antenna inputs on your home TV receiver. See CABLES for a diagram showing how to connect a monitor to your computer.

COMPUTE! BOOKS — A division of COMPUTE! magazine, Small System Services, Inc. publishes some useful books for Atari users. Some of the books are reprints of articles from COMPUTE! magazine, while others are new and unpublished material. Some of the current books in print are:

Inside Atari DOS. Bill Wilkinson. Source listing of the Atari DOS with useful comments. Not for beginners.

COMPUTE!'s First Book of Atari. Mostly reprints. 1981

COMPUTE!'s First Book of Atari Graphics. New Material. 1982

COMPUTE!'s Second Book of Atari. New material. 1982

Mapping the Atari by Ian Chadwick. This is by far the best memory map of the Atari computer. Locations are discussed in numerical order from the bottom of the OS RAM to the top.

BASIC Source Book by Bill Wilkinson. Source code listing of Atari BASIC.

COMPUTE! — Compute! Magazine often prints fairly substantial programs and articles about the Atari computers. Unfortunately, the space must be shared with Commodore and Apple computers. Back issues make excellent references to problems or undocumented items. Prior to 1982, Compute! had an entire section dedicated to Atari. Now, articles from all computers are mixed throughout the magazine.

CONCATENATE — Strings can be joined together to make a longer string by CONCATENATION. In order to CONCATENATE, you must find the length of the first string by using a LEN statement and add the next string to the place one element beyond the length of the old string. A string can be up to 32,767 characters long. By

CONCATENATEing strings, one can build long machine language programs and place them in memory. By jumping to the beginning address of the string, the program can be run. Here is an example of CONCATENATION.

```
10 DIM FIRST$(10),SECOND$(5)
20 FIRST$="HOWDY"
30 SECOND$="DOODY"
40 FIRST$(LEN(FIRST$)+1)=SECOND$
50 PRINT FIRST$
```

The second string, "DOODY", was CONCATENATEd onto the first string, creating a ten byte string, "HOWDYDOODY".

To CONCATENATE a machine language program or a BASIC LISTed program to another, use the C option in DOS 2. After typing C <return>, type the filespec of the program you want to append, a comma, the name of the program you want to append to, and then /A.

```
C <Return>
COPY-FROM, TO?
FILESPEC,APPENDEE/A <Return>
```

This will append FILESPEC to the program called APPENDEE. Do not try to append BASIC SAVEd programs or the Variable Table will be placed in the middle of the file and will not be readable.

CONDITIONAL BRANCH — There are two types of CONDITIONAL BRANCHES in ATARI BASIC, IF/THEN and ON-GOSUB. A CONDITIONAL BRANCH will send the program pointer to another line number depending on the value of a named variable. IF/THEN statements can be used to branch if a variable has certain values. A GOTO is implied after each IF/THEN statement and therefore does not need to be stated.

CONSOLE KEYS — The OPTION, SELECT, and START keys are hardware controls for special user input. The way the CONSOLE KEY works is that the program must be sent to a special hardware register to see if a key is being pressed. You cannot store a pressed console key. It must be depressed while the program is looking at it. The location of this register is 53279 ($D01F). The three rightmost bits (D0 to D2) are reset while a key is pressed. Bit D0 is reset to 0 while the START key is pressed. Bit D1 is reset while SELECT is pressed. Bit D2 is reset while OPTION is pressed.

LOCATION 53279	CONSOLE KEY
BIT D0	START
BIT D1	SELECT
BIT D2	OPTION

CONSTANT — A CONSTANT is a number or group of letters used in a program and which do not change during the program. Numbers used in a program such as 1, 2, 3, 10, 20000, etc. are CONSTANTs. Strings of letters such as "TEST", "stringsample", etc. are CONSTANTs if they are not assigned as values to a variable name. Each CONSTANT requires six bytes of memory in Atari BASIC plus one for an identification regardless of the magnitude of the number. Every time the CONSTANT is used it requires another seven bytes. For this reason you can save memory by assigning the value of the CONSTANT to a variable, such as, X1=1, and using X1 instead of 1. Each recurrent use of X1 will take only 1 byte instead of 7. CONSTANTs cannot have a value greater than 32,767.

Atari Microsoft BASIC uses two bytes for each CONSTANT in single precision mode. You can specify a double precision mode by putting a D at the end of the mantissa, before the exponent. Double precision CONSTANTs are stored in eight bytes and are accurate to 16 decimal places.

CONT — A program which has been stopped due to a BREAK key or a programming error can be restarted on the next line following the stoppage by typing CONT (while in the immediate mode). The address of the next line number in the program is stored in an area of memory called the "run time stack." After a BREAK or error, the CONT command will find the address of the next line number and begin execution at that line. If the halt occured in a line with multiple statements, none of the statements after the stoppage will be executed.

CONTROL BYTE — In a cassette record, the CONTROL BYTE is the third of the 132 byte record. It must have one of three values: $FA, $FC, or $FE. The $FA byte means that the record is the last one in the file and is followed by 128 zeros. The $FC byte simply signals that the record contains a full 128 data bytes. A $FE means that the record is partially full and that the number of bytes in the record can be found in the next to the last byte of the record.

CASSETTE FILE STRUCTURE

MARKER 1
MARKER 2
CONTROL BYTE
DATA 128 BYTES
NUMBER OF BYTES IF PARTIALLY FILLED
CHECKSUM BYTE

CONTROL CHARACTERS - PRINT ONLY — To use the control characters as characters only (not to perform their editing functions), POKE 766,1.

COPY PROTECTION — Most commercially available programs for the Atari computer are COPY PROTECTED. This usually involves writing the disk in such a way that an Atari 810 disk drive cannot reproduce all of the data (exactly). Experience with other microcomputer systems has demonstrated that COPY PROTECTION techniques change constantly, evolving as those enthusiasts inclined to circumvent the protection schemes become more experienced. There is probably no end to the competition.

CP/M — CP/M (Control Program for Microcomputers) is an operating system for microcomputers built around the 80 series (Z80 or 8080) microprocessor. CP/M handles disk reading and writing and is a standard used by many software manufacturers. The fact that CP/M was the first widely distributed standard disk operating system has made it into one of the most supported systems for microcomputers (at least until the IBM PC came along). There are over 10,000 public domain programs available in CP/M format. A huge number of commercial business software packages use CP/M. Although graphics is supported in the newer versions, very little graphics oriented software is available and the quality is not even close to the high speed and resolution of the Atari computer.

Atari has announced a CP/M hardware addition. The addition is essentially another computer based on the Z80 microprocessor. The console and keyboard of the Atari computer will be used basically as a terminal to access the CP/M computer. Additional hardware options to give CP/M compatibility are the ATR-8000 by SWP, MF-1681 by MicroMainframe, and the Critical Connection by USS Enterprises of San Jose, CA.

CPI — Characters Per Inch. Standard single spacing on typwriters and line printers is 10 CPI. Dot matrix printers often use a condensed mode of 16.7 CPI to include more information in a small space. Double width printing expands the character so that only five CPI are produced. Some printers are capable of proportional spacing in which the number of CPI varies to fill an entire line with evenly spaced words and characters. Proportional spacing is more attractive to the reader as there are no large gaps between words when the right hand margin of the page is aligned (right justified).

CPS — Characters Per Second. Although the data rate for output from the Atari computer can be up to 2,000 CPS, most printers operate at less than 200 CPS. The newer model printers are capable of a speedy 160 CPS. Daisy wheel printers, especially the lower cost types, operate at around eight CPS which translates to about five minutes per double spaced page. Speed is a serious consideration for someone trying to use a printer for business or literary work. A more expensive daisy wheel printer will work at 35 to 45 CPS and is worth the extra cost to a heavy user. (See BUFFER.)

CPU — Central Processing Unit. The 6502 microprocessor is the CPU for the Atari computer. This device is one of a family of similar microprocessors developed by MOS Technology, Inc. The Atari 810 disk drive contains a CPU called the 6507 which is a member of the same family as the 6502. The Apple][and Commodore PET are built

around similar CPU's. The 6502 processor is a byte oriented processor and data comes in, goes out, and is processed in a path consisting of eight parallel lines. This makes it an eight bit processor.

CR — The CR symbol is the ASCII name for carriage return. A CTRL-M will generate this character (which is often useful when connected to a bulletin board system via a modem). A distinction must be made between the CR and the character generated when the RETURN key is pressed. RETURN generates an ATASCII 155 (an End-Of-Line character) which is interpreted as a carriage return and a line feed. This resets the cursor to the first column and moves it down to the next line.

CRC — Cyclical Redundancy Check. A CRC is a checksum. This means that incoming data (usually from a floppy disk) is added up and the sum is compared against a number which is known or assumed to be correct. This technique is used on Atari disk files to insure accuracy of data tranfers. There are four CRC bytes between sector and track identification which you cannot read or examine, but which the disk drive floppy disk controller uses. If you have "The Chip" or a Happy Enhancement with the Archiver, you can read and modify CRC bytes.

CRASH — CRASH is computer jargon for a program interruption and failure. Sometimes a CRASH will cause you to lose control of the computer so that the only way to regain control is to switch the computer off and back on (chip reset). This type of CRASH can occur when a jump is made to a non-operable program area in memory. A voltage fluctuation can also cause such a CRASH. Simple CRASHes which generate an error can usually be fixed by DEBUGGING the program.

CRT — Cathode Ray Tube. This is the technology used to display images in televisions and computer terminals. The principle is nearly 100 years old and the tubes are very inexpensive. Very simply, a beam is made to scan the faceplate inside the tube and as the beam hits the phosphor coated face, the phosphor glows. A video controller modulates the beam to form characters or graphics. Other technologies known as flat panels are not as bulky as CRT's and will allow much more compact portable computers within five years.

CSAVE — CSAVE SAVEs the tokenized BASIC program in memory to a cassette file.

CTIA — The CTIA is the processor chip which was replaced by the GTIA in all Atari 400 and 800 computers.

CURSOR — The cursor is a character (inverse space) which is used to indicate the position of the next character to go on the screen. The cursor can be turned off by POKEing a number (any number except zero) into location 752. The cursor is normally transparent. That is, if you back up over a character it will show through (in inverse mode). The cursor can be made opaque by POKEing a 2 into location 755. See BLINKING CURSOR.

CURSOR OFF — You can turn off the cursor in BASIC by typing the statement POKE 752,1 <Return>.

CURSOR ON — You can turn your cursor back on by typing POKE 752,Ø <Return>.

CYCLE STEALING — CYCLE STEALING is the process in which the ANTIC chip interrupts the 6502 processor from its activity to grab some data from memory for display on the screen. The cycles are "stolen" in that no other accumulator activity can occur during this interruption.

D

DATA — In Atari BASIC the DATA statement is used after a line number to identify a series of string or numeric variables. The data in a DATA statement can be loaded by a READ statement. The data should be processed or acted upon after it is read. When you see programs with READ-DATA routines that have many numbers or hexadecimal digits, the program is placing the OPCODES for a machine language program into memory. After the program is put into memory, it must be run by jumping to the RUN address via a USR statement. If a READ statement tries to go past the last piece of data in a DATA statement, an ERROR 6 will result. If there are more data elements than are read by a READ command, a pointer will remain on the next piece of data to be read. RESTORE or CLR will set the pointer back to the first DATA statement. Most errors found in typing in machine language ATASCII strings (all numbers) are caused by typing Ø instead of 1 and by leaving out a comma. Copy and run the following program. The program employs a READ-DATA routine.

```
10 REM * ABCS OF ATARI COMPUTERS
20 REM ** SHOPPING LIST MAKER
30 REM ** PROGRAM: SHOPPING.BAS
40 REM **
50 SETCOLOR 4,1,4:SETCOLOR 0,14,10:SET
COLOR 1,12,0:SETCOLOR 2,7,8
60 POKE 82,2:POKE 752,1:? CHR$(125)
70 OPEN #1,4,0,"K:"
80 ? :?
90 ? "          GROCERY LIST HELPER
   "
100 ?
110 ? "  THIS PROGRAM WILL PRODUCE A P
RINTED"
120 ? "  LIST OF ITEMS TO ASSIST YOU I
N YOUR"
130 ? "  WEEKLY TRIP TO THE SUPERMARKE
T"
140 ? :?
150 ? "    BE SURE TO TURN ON YOUR PRI
NTER"
```

```
160 ?
170 ? "  TO ADD A NEW ITEM TO YOUR LIS
T,":? "  TYPE THE ITEM IN THE DATA STA
TEMENT"
180 ? "  STARTING  AT LINE 640"
190 POSITION 2,22:? "  HIT ANY KEY TO
PROCEED ":GET #1,B
200 POKE 752,1
210 DIM IT$(200),I$(200),A$(50),N$(50)
,D$(20),G$(20)
220 TRAP 630
230 ? CHR$(125):? "GROCERY LIST MAKER"
:?
240 ? "TODAY'S DATE";:INPUT D$
250 LPRINT "GROCERY LIST FOR ";D$
260 LPRINT
270 TRAP 32767
280 ? :? "Y       TO PRINT AN ITEM"
290 ? "N       TO MAKE AN ADDITIONAL NO
TE"
300 ? "A       TO ADD AN ITEM NOT ON LI
ST"
310 ? "ANY KEY  TO SKIP AN ITEM":POKE
82,8:?
320 TRAP 530
330 FOR E=1 TO 200
340 READ IT$
350 IF IT$="END" THEN GOSUB 550
360 I$(E)=IT$
370 POSITION 8,14
380 ? CHR$(156);CHR$(156)
390 ? CHR$(156);CHR$(156)
400 ? CHR$(156);CHR$(156)
410 ? CHR$(156);CHR$(156)
420 ? CHR$(156);CHR$(156)
430 ? CHR$(156);CHR$(156)
440 ? CHR$(156);CHR$(156)
450 POSITION 8,14:? CHR$(156)
460 ? IT$
470 GET #1,R
480 IF R=89 THEN LPRINT I$(E)
490 IF R=78 THEN GOSUB 540:LPRINT I$(E
),"=>";A$:GOTO 520
500 IF R=65 THEN GOSUB 550:LPRINT N$
510 REM
520 NEXT E
530 ? "END OF LIST":GOTO 550
540 ? "ADDITIONAL NOTE";:INPUT A$:RETU
```

```
RN
550 POSITION 8,14:POKE 752,0
560 ? "ITEMS NOT ON THE LIST"
570 ? " =>E TO END":? " =>R TO RETURN
TO LIST":INPUT G$:IF G$<>"E" AND G$<>"
R" THEN LPRINT G$
580 POSITION 8,17:? CHR$(156)
590 IF G$="E" THEN GOTO 620
600 POKE 752,1:IF G$="R" THEN GOTO 330
610 POKE 752,1:GOTO 550
620 POKE 82,2:END
630 ? " TURN ON PRINTER ";:? " TURN ON
 PRINTER ";:GOTO 630
640 DATA APPLES,ARTICHOKES
650 DATA BREAD,BUTTER,BEANS,BANANA,BRE
AD CRUMBS,BRUSSEL SPROUTS,BAGELS
660 DATA CRACKERS,CHEESE,COOKIES,CEREA
L,CHICKEN,CREAM CHEESE
670 DATA CELERY,CORN,CARROTS,COTTAGE C
HEESE,COFFEE,CHILI
680 DATA DISH SOAP
690 DATA ENGLISH MUFFINS
700 DATA FLOUR,FIGS,FISH
710 DATA HONEY
720 DATA ICE CREAM
730 DATA JELLY,JUICE
740 DATA KETCHUP
750 DATA LAUNDRY SOAP,LUNCH MEAT,LASAG
NA NOODLES,LETTUCE
760 DATA MILK,MUSHROOMS,MAYONNAISE,MUS
TARD,MEAT
770 DATA NAPKINS,NUTS
780 DATA OYSTERS,ONIONS,OIL,ORANGES,OR
ANGE JUICE
790 DATA POTATOES,POTATO CHIPS,PRETZEL
S,PORK ROAST,PEARS,PEACHES,PASTA,PICKL
ES,PEANUT BUTTER
800 DATA RICE,RICE-A-RONI
810 DATA SOAP,SHAMPOO,SOUP,SPICES,SOUR
 CREAM,SPAGHETTI,SPINACH
820 DATA TOFU,TUNA,TORTELLINI,TACOS,TO
ILET PAPER,TEA,TOMATO PASTE
830 DATA WINE,WATERCRESS
840 DATA ZUCCHINI
850 DATA END
```

DATABASE MANAGERS — A DATABASE MANAGEMENT program is an organizational tool which facilitates using your computer and disk drive or cassette recorder to maintain records. A cassette recorder is not recommended because of slow access to stored information and because of the lack of random access to any record on file. DATABASE programs are designed in two types. One type loads all data into computer memory and the information is pulled out, sorted, and searched very quickly. The limitation is the amount of memory in your system. Even a full 48K or 52K system is not really suitable for business use. The other type maintains an abbreviated marker or pointer (actually, an index) in memory, and all data is stored on the floppy disk. Many more records can be stored, but access is slower, and in some cases, every item must be searched to find a specific piece of data.

Typical applications for DATABASE programs are mailing lists, inventories, personal collections (such as record albums), membership rosters, product marketing information, test results, and so on. Data Perfect by LJK, Filemanager+ and Syn-file by Synapse, File-fax by TMQ, and CCA DBMS by Custom Electronics are examples of relatively large DATABASE MANAGERS. Smaller systems are APX Data Base Report, Home Filing Manager by Atari, APX Data Management System, and MMG File Manager.

DATA PERFECT — DATA PERFECT is a very compact and versatile database program. Its major shortcoming has been its documentation. The intermediate user may need to spend four to eight hours wading through the manual to build a small database file. Utilities are very powerful, but are also very difficult to master. Even printing a report requires extensive experimentation unless one is familiar with IBM's RPG language. While the program is not menu driven, keyboard options are represented in an option line at the top of the page. The exact function of each option is not intuitively obvious and will require many referrals to the manual.

Screens are defined for each application and data entry is made using the screen as a mask. The actual record entry process is very easy once the screen is defined. Particular thought and care should be given to screen layout as changes are possible but difficult. Math functions are supported for totals, formulae, etc. For example, quantity * (times) unit price + (plus) 6% could be a computed line. Files up to 80 records or so can be manipulated totally within memory in a 48K system so searching on any field is fast. A key is stored in memory and is used to access a random record on disk. Searching for a non-key item on a large disk file requires sequential reading of all records, which, depending on the length of the file, can be a slow process. Only one database is stored on a disk. Multiple report programs can be stored on a disk. DATA PERFECT uses the LJK DOS; a DOS which is not directly compatible with Atari DOS. (See LJK DOS).

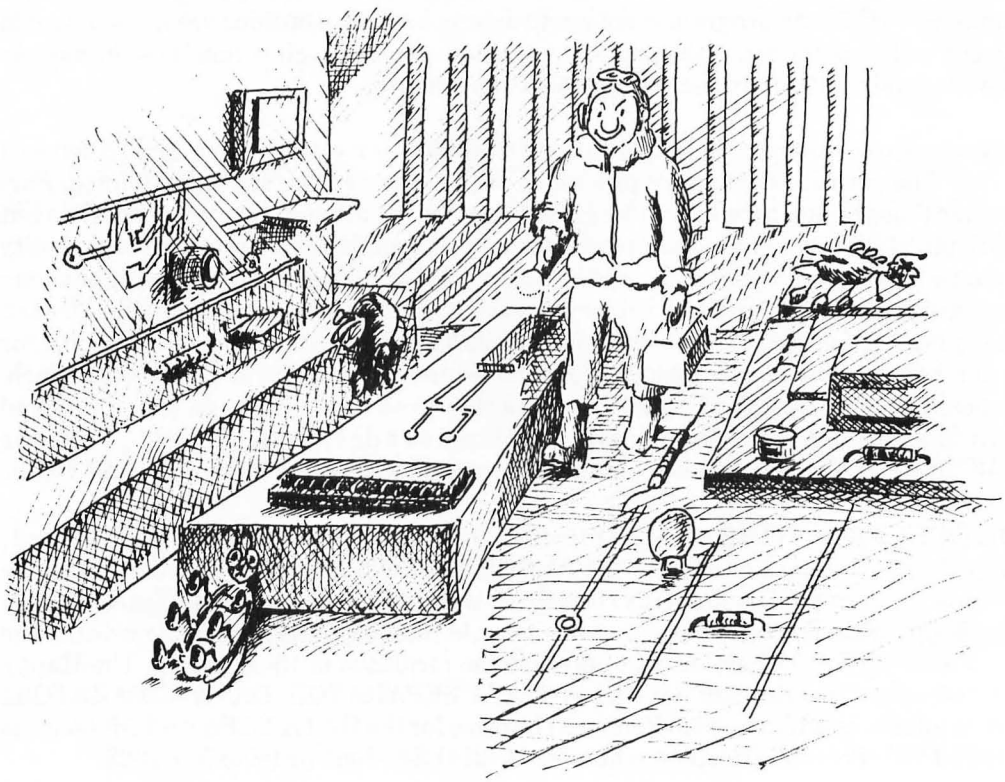
DATA SEPARATORS — Atari 810 disk drives manufactured before September 1, 1981 do not contain an external DATA SEPARATOR circuit. This circuit helps the Floppy Disk Controller chip (1771) to process weak or distorted signals picked up from the floppy disk. Early drives may have trouble reading data from inner tracks. The DATA SEPARATOR circuit will decrease the incidence of these errors. The Happy Enhancement (for the 810) has its own DATA SEPARATOR. DATA SEPARATORs are available as add-ons from Percom. The price for the DATA SEPARATOR board is around $30. Percom's telephone number is (214) 340-7081 or (800) 527-1222.

DCB HANDLERS — The Device Control Block HANDLERS control the stream of data moving through the serial bus. These handlers are short machine language routines in the Operating System which handle all of the details of data transferral.

DCB — Device Control Block — The DCB is the area of memory in which the serial input and output for the disk drives, printer, and RS232 ports is controlled. The DCB is located between 768 and 779 ($300 and $30B).

DE RE ATARI — This book, which is primarily about graphics, is a collection of writings by Chris Crawford. It also includes material about sound and the operating systems. A wealth of information is presented in DE RE ATARI, but a lack of continuity is the book's major drawback. There is no index to find specific entries.

DEBUG — Bugs are errors which cause a program to do something other than the original intention. The process of correcting the errors, usually one step at a time, is called DEBUGging. DEBUGging in BASIC can be done with a utility, such as BASIC Debugger by MMG or by adding lines to print out signals or variables at certain milestones in the program. In Microsoft BASIC or BASIC XL, you can use the TRACE function, which will print out the line number of each successfully completed line. DEBUGging in assembly language can be done with the BUG program; however, this is only good for correcting syntax errors. Gross logical errors require flowcharting or some macroscopic DEBUGging technique.



DEF — In Microsoft BASIC II, on the extension disk. DEF is the command to DEFine a mathematical function which can be called any time thereafter. You might DEFine a geometric mean by the command- DEF GEOM(X,Y)= SQR(X*Y). For example, every time you wanted the mean of two numbers you could use GEOM(2,3).

DEFAULT — The DEFAULT value is a number or symbol which is supplied when the user does not specify otherwise. For example, in specifying a drive number, D: refers to drive 1, not drive 2, 3, or 4, because no number was specified, and 1 is the DEFAULT drive.

DEFDBL — In Microsoft BASIC II, DEFDBL DEFines the variables starting with the letters given in the DEFDBL command as DouBLe precision real variables. Double precision numbers are represented in memory by eight bytes, seven for the mantissa and one for the exponent.

DEFINT — In Microsoft BASIC II, DEFINT defines the variables starting with the letters given after the DEFINT statement as integer variables.

DEFSNG — In Microsoft BASIC II, DEFSNG DEFines the variables starting with the variables given after the DEFSNG command as single precision real variables. Single precision real numbers are stored in four bytes, three for the mantissa and one for the exponent.

DEFSTR — In Microsoft BASIC II, DEFSTR is a command used to DEFine a STRing by using the starting letter for that string.

DEG — In Atari BASIC, DEG is the command to switch the processing of trigonometric functions between DEGrees and radians.

DEL — In Microsoft BASIC II, on the extension disk, DEL is the command to DELete a range of BASIC lines. DEL can be used to delete up to a certain line number, after a certain line number to the end, or within a range of line numbers.

DELAY LOOP — To slow down a statement in a BASIC program, you can insert a DELAY LOOP. This would be useful, for example, when displaying instructions on the screen. To make sure the instructions are read and enough time is allowed, a DELAY LOOP is placed before the next statement to be executed. The loop is simply a FOR-NEXT loop with nothing to do except count. (FOR TIME=1 TO 100:NEXT TIME). This technique cannot be used for reliable timing. The duration of the loop depends upon the line numbers of the loop routine. Try the following example:

```
10 PRINT "READ THIS"
20 FOR X=1 TO 1000:NEXT X
30 PRINT CHR$(125)
```

DELETE — Atari BASIC does not support extensive file editing. Often it is necessary to DELETE large blocks of statements. There are three ways to DELETE. You can LIST the file out to a disk file, copying only the desired line numbers. The format is

LIST"D:TEST",100,200. This will save lines 100 to 200 in your program to a LISTed file called TEST. Another way is to type in the line numbers you want to DELETE with no statement for that line. This way is tedious. Another way is to use a deletion utility. BASIC commander and BASIC XL have a built-in block DELETEr as does the Monkey Wrench. The following program will also do the deletion. Type in the program and LIST it to a disk or tape file. When your own program is in memory, ENTER the DELETER program and type GOTO 32000 to DELETE blocks of lines.

```
0 REM ** PROGRAM LINE DELETER
1 REM ** FILE: D:DELETER.LST
2 REM ** USE AS A SUBROUTINE AT END
3 REM ** ABCS OF ATARI COMPUTERS
4 REM
32000 GRAPHICS 0
32001 TRAP 32001:POKE 84,11:? "ENTER F
IRST LINE # TO DELETE";:INPUT F:? "ENT
ER LAST LINE # TO DELETE";:INPUT L
32002 IF INT(F)<>ABS(F) OR INT(L)<>ABS
(L) OR L>32000 OR L<F THEN ? CHR$(253)
:GOTO 32001
32003 GRAPHICS 0:? :? :? F:F=F+1
32004 ? "CONT":POSITION 0,0:POKE 842,1
3:STOP
32005 POKE 842,12:IF F<=L THEN 32003
32006 GRAPHICS 0:? "DELETION COMPLETE!
":END
```

DELIMITER — The mark used to stop a process is called a DELIMITER. In word processors, for example, a block of text is often moved to a buffer or deleted. All text after the cursor can be moved, but a DELIMITER is needed to tell the program where to stop taking text. The DELIMITER is placed exactly at the end of the block to move and the cursor is placed at the beginning. The cursor is moveable and the DELIMITER is stationary.

DEMOPACS — There are eight DEMOPACS produced by Atari Product Support. The DEMOPAC is a tutorial and demonstration of some aspect of Programming Atari Computers. The subject of the DEMOPACS are: Strings and Formatting, Data File Processing, Programming Examples, Atari Color Graphics, Advanced Graphics, and Advanced System Features.

DESTINATION DISKETTE — During the copying of a diskette with DOS option J or any other utility, you will be asked for a source diskette and a DESTINATION DISKETTE. The source is the original disk which you want to duplicate. You should have a write protect tab on this disk to make sure you do not inadvertantly destroy it. The DESTINATION DISKETTE is the one on which the copy will be written. This disk should be formatted and must not be write protected or you will get an ERROR 144. If your source is not write protected nor labeled, you risk copying blank data over your original files. This could be very disappointing.

DEVICE — The DEVICE is an input, input/output, or an output entity controlled by the CIO. The DEVICEs are: P: (printer; output only); K: (keyboard; input only); D: (disk; input/output); C: (input/output); E: (display editor; output); S: (screen; input/output) and R: (RS232, input/output). New DEVICEs can be defined and used if an entry is made in the Handler Address TABleS (HATABS) at locations 794 to 831 ($31A to $33F) and the handler routine is written. Resident handlers are P:, C:, E:, S:, and K:. If DOS is booted, a D: DEVICE is added to the table. If the RS 232 handler is booted in, the R: DEVICE is added.

DEVICE HANDLER — A DEVICE HANDLER is a routine (a program) in the Operating System software that controls the functions of the peripheral device (printer, RS-232, cassette, disk drive, screen, etc.) See HANDLER.

DEVICE TIMEOUT — An attempt to access a device on the serial bus which does not respond will give a DEVICE TIMEOUT error (ERROR 138). This occurs because a timer called CDTMV1 in locations 536 and 537 ($218 and $219) counts down from 255 starting from the time the command to access the device is given. If the device does not respond by the time the timer reaches 0, a DEVICE TIMEOUT error results. This time is between four and five seconds. Check to see if the drive or printer is plugged in and turned on, and that all cables are connected to remedy the problem.

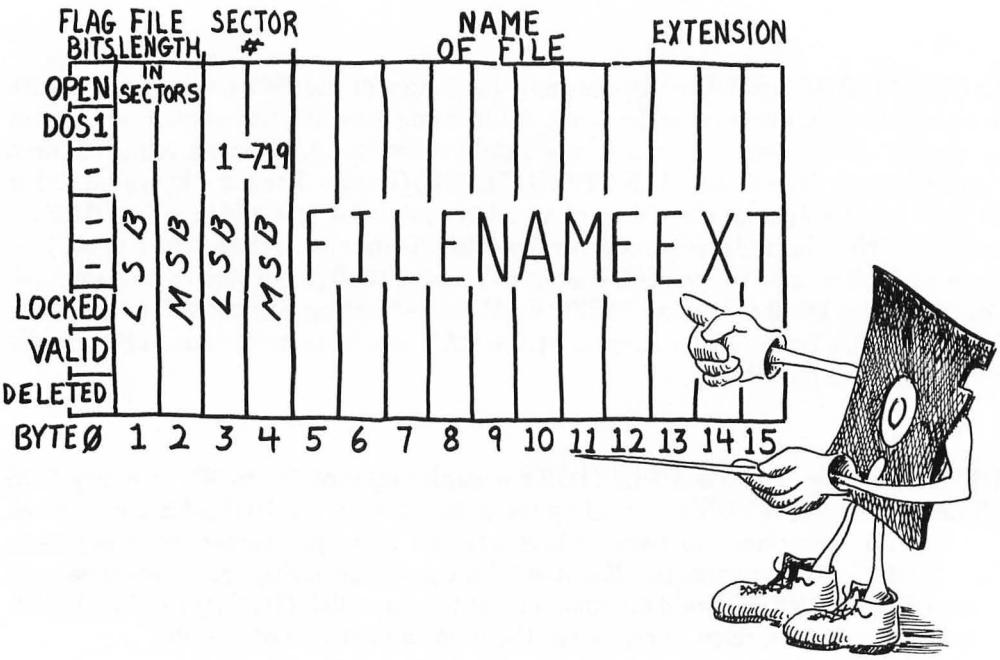
DIM (DI.) — In Atari BASIC, DIM is used to reserve memory space in the string/array area of memory for strings and arrays. DIM is not required in Microsoft BASIC. If you try to use a string variable without first DIMensioning it, you will cause an ERROR 9.

DIRECT ADDRESSING — In assembly language for the 6502 processor there are several methods or modes of addressing. Addressing is calling the memory location of the operand for the next assembly language instruction. Addressing requires three bytes unless it is done in the DIRECT ADDRESSING mode. The three bytes consist of one byte for the opcode (instruction) and two bytes for the address. In DIRECT ADDRESSING, the address is found only within the first page (locations 0 to 255), so only one byte is needed to specify the address because the 0 page (high byte) is implied. Page 0 contains OS RAM, some BASIC RAM, some floating point RAM, and various other free bytes. Direct addressing to 0 page RAM can be done very quickly, but it is limited to only 0 page locations.

DIRECTORY — The disk DIRECTORY occupies sectors 361 to 368 on every DOS 2.0S disk. The DIRECTORY is read by the A function in the DUP.SYS menu. Each DIRECTORY entry uses 16 bytes. There are 128 bytes per sector, allowing eight DIRECTORY entries per sector. Eight entries per sector multiplied by eight sectors for the DIRECTORY allows a maximum of 64 files per disk (DOS 2.0S). The first 16 bytes of the entry are used to represent the name and status of the file.

BYTE 0	Bit 0	Open file flag
	Bit 1	DOS 1 flag
	Bit 2	?
	Bit 3	Unused
	Bit 4	Unused
	Bit 5	Locked File
	Bit 6	Valid file
	Bit 7	Deleted file flag

BYTE 1 BYTE 2		Total number of sectors in the file (LSB/MSB)
BYTE 3 BYTE 4		Number of the sector for the file (LSB/MSB)
BYTE 5	F	
BYTE 6	I	
BYTE 7	L	
BYTE 8	E	FILENAME
BYTE 9	N	
BYTE 10	A	
BYTE 11	M	
BYTE 12	E	
BYTE 13	E	
BYTE 14	X	EXTENSION
BYTE 15	T	



DISABLE KEYBOARD — POKE 16,255 to completely DISABLE the KEYBOARD. This will prevent mischief by those you wish to keep away from your programs.

DISABLE BREAK KEY — See BREAK KEY

DISK DETECTIVE — This package is one of the simplest disk utilities available. Essentially, the components are a sector viewer and writer. Also included is a disk map maker which checks to see if sectors are good or bad, used or empty. This map takes a full three minutes to build. There are better routines available. Produced by Datasoft.

DISK FIXER — DISK FIXER is distributed by Atari Program Exchange and was written for development purposes inside Atari. The documentation and cosmetics of the program reflect this. It is a very useful and efficient utility, but you should be aware of several points. All numbers used, file numbers, sector numbers, bytes, etc. are in hexadecimal. This may take some time for those of you who think only in decimal. The directory starts in $169, not 361. Secondly, you can do some major damage to a disk by changing directory entries, sector links, and bit maps. There is not much verification before the data is changed, so be sure you have a backup of your damaged disk before you start hacking on it. The options available on DISK FIXER are:

A. Directory entries — list entries, starting sector, number of sectors and status.

B. Trace Sector Chain — follows the links between sectors in a file which you specify. This is the way to find broken links.

C. Modify Directory Entry — change the name, status, start or number of sectors. You must be careful with this option.

D. Check Allocation Map — Builds a bit map from the disk files and compares it to the VTOC bit map. You can write the new one in if you want.

E. Modify Sector Link — You can reconnect broken files by this option. You will probably lose some data but the file will usually load and you can edit it.

F. Set Drive Number — for up to eight disk drives.

G. Exit to DOS — quits DISK FIXER.

H. Dump Sector — Writes the contents of a sector to the screen.

I. Edit Sector — You can edit the hex bytes of a sector and write the changes back to the disk. Without a search feature this has very limited value.

DISK, FLOPPY — The FLOPPY DISK is a type of magnetic recording media for semi-permanent data storage. The DISK stores data in the same manner as recording tape. An oxide material suspended in a resin binder is coated on a plastic substrate to form a smooth layer. The oxide material can be magnetized during the writing stage which polarizes the material magnetically. When the read/write head is passed over the spot again (in the READ mode), the head picks up the polarization switches and interprets them as highs or lows (0s or 1s).

DISK FILES — By looking at the first two bytes of an Atari DOS file, one can make an educated guess as to what type of file it is, such as BASIC LISTed, object code, etc. The following chart is only intended as a guideline; a data file may, for instance, have the same structure as a binary load object file.

FILE TYPE	FIRST BYTE	SECOND BYTE
Atari BASIC-SAVEd	0	0
Atari BASIC-LISTed	NOT 0	NOT 0
BASIC A+ - SAVEd	0	0
BASIC A+ - LISTed	NOT 0	NOT 0
Microsoft BASIC SAVE	0	NOT 0
Microsoft BASIC LIST	155	NOT 0
Assembler/Source	NOT 0	NOT 0
Machine/Object DOS I	134	9
Machine/Object DOS II	255	255
Machine/Object .COM	255	255

Machine language .COM files should run if binary loaded through the L option of DOS. This is because they have the RUN and INIT addresses appended to the end of the file. Otherwise, you would have to trace through to the end of the file to look for the addresses and even then expect some difficulty deciphering them from instructions. To examine bytes of a sector, use a short BASIC program such as: 1 OPEN #2,4,Ø, "D:FILESPEC" and 2 FOR X=1 TO 4:GET #2,A: PRINT A: NEXT X. This program will print out the first four bytes of the file named in FILESPEC.

The following program will analyze a binary file for the RUN and INIT addresses. There may be several modules to a program and these will be listed when you run the program. The second part of this program will convert HEX numbers to Decimal and vice versa. Do not forget to use D: for the filename.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** PROGRAM NAME: BINAN.BAS
30 REM ** BY PETE GOODEVE
40 REM
50 DIM NAME$(40)
60 TRAP 70:DIM HEX$(12),HXX$(1)
70 TRAP 32767
```

```
80 ? CHR$(125)
90 ? :? "FILE TO BE ANALYSED: "
100 ? " USE D: FOR FILENAME ";
110 INPUT NAME$
120 IF NAME$="" THEN END
130 TRAP 360
140 OPEN #1,4,0,NAME$
150 TRAP 390
160 GOSUB 300
170 ADDR1=HADR
180 GOSUB 300
190 ADDR2=HADR
200 BLKSIZ=ADDR2-ADDR1+1
210 IF ADDR1=736 THEN 280
220 IF ADDR1=738 THEN 290
230 FOR KK=1 TO BLKSIZ
240 GET #1,X
250 NEXT KK
260 ? "---","--- "
270 GOTO 160
280 ? "... RUN ADDR:..":GOSUB 300:GOTO
 160
290 ? "...INIT ADDR:..":GOSUB 300:GOTO
 160
300 GET #1,X:GET #1,Y
310 HADR=X+256*Y
320 GOSUB 430
330 IF HADR=65535 THEN ? "...HEADER...
":GOTO 300
340 ? HADR,HEX$(I,12)
350 RETURN
360 ? "BAD FILENAME"
370 CLOSE #1
380 GOTO 90
390 ? "_____________"
400 CLOSE #1
410 GOTO 540
420 GOTO 90
430 X=HADR:I=12
440 XD=X:X=INT(X/16):XD=XD-X*16
450 IF XD<10 THEN HXX$=STR$(XD)
460 IF XD>9 THEN HXX$=CHR$(ASC("A")+XD
-10)
470 HEX$(I,I)=HXX$:I=I-1
480 IF X=0 THEN 510
490 IF I=0 THEN HEX$(1,1)=">":GOTO 510
500 GOTO 440
510 I=I+1:REM ? ," = Hex: ";HEX$(I,12)
```

```
520 RETURN
530 TRAP 540:DIM HEX$(12),HXX$(1)
540 TRAP 550
550 ? "INPUT HEX OR DEC (OR QUIT)";:IN
PUT HXX$
560 IF HXX$="H" THEN 710
570 IF HXX$="D" THEN 600
580 IF HXX$="Q" THEN GOTO 90
590 GOTO 540
600 ? "DEC #:";:INPUT X:I=12
610 IF X=0 THEN 550
620 XD=X:X=INT(X/16):XD=XD-X*16
630 IF XD<10 THEN HXX$=STR$(XD)
640 IF XD>9 THEN HXX$=CHR$(ASC("A")+XD
-10)
650 HEX$(I,I)=HXX$:I=I-1
660 IF X=0 THEN 690
670 IF I=0 THEN HEX$(1,1)=">":GOTO 690
680 GOTO 620
690 I=I+1:? ," = Hex: ";HEX$(I,12)
700 GOTO 600
710 ? "HEX #:";:INPUT HEX$:X=0
720 IF LEN(HEX$)=0 OR HEX$="0" THEN 55
0
730 HXX$=HEX$(1,1):IF LEN(HEX$)=1 THEN
 HEX$="":GOTO 750
740 HEX$=HEX$(2)
750 IF HXX$<="9" THEN XD=VAL(HXX$)
760 IF HXX$>="A" THEN XD=ASC(HXX$)-ASC
("A")+10
770 X=X*16+XD
780 IF LEN(HEX$)>0 THEN 730
790 ? ,"= Dec: ";X
800 GOTO 710
```

DISK PEEPER — DISK PEEPER is a low cost utility sold by license from Home Software to user groups. This utility is written in Forth and is basically an editor of disk based data. You must know your way around the disk in order to make full use of this package.

DISK UTILITIES — DISK UTILITIES are programs required by Atari computer users and programmers to help correct or repair damage to disks which can result from extensive use, program bugs, hardware errors, or general abuse. DISK UTILITIES can also be used to modify machine language programs, or more specifically, to modify any data written on a floppy disk. Additional utilities included in some DISK UTILITIES are: various searches for ASCII or hexadecimal strings, machine language dis-

assemblers, directory rebuilders, VTOC rebuilders, file tracers, link and next sector utilities, and bad sector disk map builders. DISK UTILITIES can create much more damage than they can repair if one is not thoroughly familiar with the architecture of the Atari floppy disk. For this reason, one must work on an exact replica of the damaged disk. A sector copier can usually be used to make the back up replica. Descriptions of all DISK UTILITIES currently on the market are presented here.

DISK WIZARD — This package is a collection of utilities for disk repair and maintenance. The primary feature of this utility is the search routine which can find a series of up to 32 bytes on a disk. This might be useful for changing or personalizing a machine language program; for example, by putting your name on the title screen. DISK WIZARD also works on double density formats.

DISKED - v.1.12 — DISKED is a very comprehensive collection of disk utilities and it comes with excellent documentation. Many examples of how to correct real disk problems are included. Also included are a disassembler and a machine load address finder. You can show the addresses of a machine language file after you specify the origin address. Duped is a sector copier which will read sectors from one disk and write to another. CSBOOT and DSKBOOT allow transfer of files from non-DOS disk files to short IRG cassette files. DSKMAP builds a map of occupied, empty, and bad sectors. RECOVER is a unique utility that will rebuild a Variable Name table that has been altered to prevent listing. Amulet Enterprises.

DISKEDIT — DISKEDIT contains the usual disk tools such as a sector copier, sector editor, arithmetic mode, bad sector finder, and search routine. The most notable feature is the disassembler which works from memory. If you know the address of a binary load or boot file, you can disassemble the object code into assembly language (without labels, of course.) Soft Unlimited.

DISKEY — DISKEY is the most powerful collection of utilities available for the Atari on one disk. Unfortunately, the documentation is cryptic. If you can decipher the book which comes with the program, you probably will be able to do all of your disk maintenance and repair with DISKEY. The program is not menu driven so you will need the manual almost every time you use DISKEY. There are over 50 single commands; this makes memorization very difficult. Sometimes a single letter command takes on two meanings. X means exit in File mode and exclusive - or in non-File mode. The best features are the file tracing capability, the VTOC rebuilder, the search for up to 10 bytes and the calculation of the location in memory where a file will load. Some of the copy routines need two disk drives to operate. Adventure International.

DISKSCAN — DISKSCAN was one of the earliest disk utilities. It is basically a simple editor with search and disassembly subroutines. You can toggle the display of the sector between hexadecimal and ATASCII by hitting the T key. Also included is the capability of making a binary load file from sectors on a disk. The program is written in BASIC and documentation is included on the disk. David Young.

DISK WIZ II — DISK WIZ II offers extensive repair capabilities. You can find bad sectors, edit sectors, duplicate sectors, print, disassemble, and more. Allen Macroware.

DISPLAY LIST — The ANTIC is a custom microprocessor common only to Atari computers which controls output to the television or monitor. The ANTIC does not receive instructions via assembly language or BASIC, but rather from a program called the DISPLAY LIST. The DISPLAY LIST is written into RAM by the 6502 processor every time a GR. command is made. Only three things are specified in the DISPLAY LIST: where (in RAM) the displayed data can be found (the LMS or Load Memory Scan), which display modes to put the screen into (up to 16), and special options (interrupt, vertical scroll and horizontal scroll). The display list is a series of numbers which specifies all of the modes and data which the ANTIC processes. By changing graphics modes on the screen, some attractive title screens or game playfields can be designed. This is what is happening when you see large and small text mixed on the same screen. The memory address of the DISPLAY LIST is found at the addresses 560 and 561. (PEEK(560) + 256 * PEEK(561) will point you to the address of the DISPLAY LIST.) A custom DISPLAY LIST can be created and the address of the new DISPLAY LIST placed in locations low address 560 ($230) and high address 561 ($231).

DISPLAY MODE — The DISPLAY MODE is the technique used by the ANTIC processor for interpreting data from RAM for presentation on the screen. In BASIC there are are 12 modes called graphics modes (0 through 11). (The CTIA only supported nine modes.) The ANTIC supports 17 graphics modes. The modes are set by the bytes comprising the display list. ANTIC modes can be from 02 to 0F, which covers BASIC modes 0 through 8 (except for five modes which are not accessible through BASIC). GTIA modes 9, 10, and 11 are accessed by setting bits 6 and 7 in the priority register at 623 ($26F). In ANTIC mode F, setting bit 6 enables GR.9, setting bit 7 enables GR.10, and setting both 6 and 7 enables GR.11.

DISTORTION — In the SOUND statement in Atari BASIC, the DISTORTION of the voice is controlled by the third expression after the SOUND statement. The DISTORTION expression can be an even number between 0 and 14, with 10 and 14 producing pure tones. The eight values (0,2,4,6,8,10,12 or 14) set three bits in each of four hardware registers (called AUDC1 through AUDC4). These resistors are at locations 53761 ($D201), 53763 ($D203), 53765 ($D205) and 53767 ($D207). The three bits for the DISTORTION are bits 7, 6 and 5, the leftmost bits. The DISTORTION is actually a form of noise as certain pulses are deleted from the computer generated square waveform.

DLI — The Display List Interrupt gives the 6502 processor a signal during which some registers relating to the screen can be changed on a regularly timed basis. The DLI is only a tool by which the interrupt routine can be timed with the vertical blanking. The activity to be performed must be programmed by the user.

The signal to begin the DLI is found in the display list which is the set of instructions for the ANTIC chip. When an instruction with the high bit set (128 added) is found, the ANTIC continues processing the display until the last line is drawn. Then it checks NMIEN in location 54286 ($D40E) to make sure that it contains 192 for a DLI enable. If the DLI is not enabled, the ANTIC continues on with its regular business. If the DLI is enabled, the program looks at locations 512 and 513 ($200 and $201) to find the address of the routine to execute during the DLI. This routine could change a color register or the horizontal location of a player. Page 6 is a convenient place to locate the DLI

routine. Only display related registers should be changed in the DLI routine, or else all kinds of trouble could result. At the end of the routine the program should return control to the 6502 with an RTI.

DLI VECTOR — The Display List Interrupt VECTOR is located at 512 and 513 ($200 and $201). The address of the routine to be run during the interrupt must be placed in this vector. The non-maskable interrupt enable at 54286 ($D40E) must be poked with $C0 for an interrupt. Bit 7 of the display list instruction must be set (add 128) at the place where the interrupt is to happen.

DMA - Direct Memory Access — This is the technique whereby the ANTIC microprocessor steals memory cycles from the 6502 microprocessor in order to fetch information from memory. The types of information the ANTIC retrieves are: 1) its own display list instructions; 2) lists of characters or memory map graphics; 3) character graphics and moving object graphics (players and missiles) pointed at in PMBASE (54279 or $D407).

You can speed up the 6502 so that it is dedicated only to calculations, not graphics, by disenabling the DMA. Of course your TV screen will go black. You can do this by POKEing location 559 with a zero. Be sure to PEEK at 559 first to find out what is there before you change it or you will never be able to see the results of your calculations. Actually, you should save the contents of 559 in location 733 ($2DD) where it can remain undisturbed (at least in the XL series). Calculations will be sped up by about 30 percent.

DMACTL — This register at 54272 ($D400) controls the Direct Memory Access (DMA). The shadow for this register is 559 ($22F). All values should be written into this location. The DMA can be turned off by POKEing a zero (0) into 559. This allows the 6502 to devote all of its time to number crunching. The narrow playfield is enabled by setting bit 0 to zero in 559. The standard playfield is obtained by setting bit 1 in 559. Bit 5 must be one (1) for the ANTIC to fetch data through DMA.

DOCUMENTATION — Depending on the program type, the documentation can be the most important part of the package. Many types of application software programs are too complex or obscure to be easily comprehended. Buyers should look for clear, simple, yet very thorough documentation when spending money for new software. Unfortunately, much software is not well documented because of the lack of interest by the programmer (or lack of resources).

DOS (DO.) — In BASIC, the DOS command is used to load in the DUP.SYS package. DOS can be used in immediate mode or in a program.

DOS VECTOR — When you type DOS in BASIC, a pointer is followed to a routine which loads in the DUP.SYS package of utilities. You can borrow this vector for your own use. The location of the DOS vector is in RAM at locations 10 and 11 ($0A and $0B). Since they are in RAM in page 0, you can change them to point anywhere you want. You could point it at the start of BASIC (40960) or at a subroutine you loaded into memory. Remember, all you have to do to enter the routine once you have changed the vector is type DOS.

After you set 10 and 11 they will be reset when you press SYSTEM RESET unless you do the following. Locations 5446 and 5450 ($1546 and $154A) contain the value that the warmstart routine places back into 10 and 11. So if you POKE your DOS VECTOR location into 5446 and 5450 (LO-HI), you will keep your new pointer until you turn off the power.

DOS - SUPER — An enhanced version of Atari DOS 2.0S. Five additional features are included in the DUP.SYS package. A total of 76 sectors are required for the DUP.SYS. The additional functions are:

P. Copy Sectors. A range of sectors can be copied from one drive to another. Prompting is supplied.

Q. Check Sectors. Sectors can be examined for integrity within a specified range.

R. Radix Conversion. A built-in Hexadecimal to Decimal and Decimal to Hexadecimal converter is supplied.

S. Drive Speed. A disk drive speed checker.

T. Write Verify. This is a toggle to turn off the write verify feature of Atari DOS. This speeds up writing by a factor of 2.

The J option performs a sector copy as opposed to a linked file copy. The options do not require a <Return> after the letter is chosen.

DOS - HIDDEN FUNCTIONS — There are many undocumented tricks you can use in Atari DOS (actually DUP.SYS). These are several of the more useful functions which you can call up while you have the DOS menu on your screen:

1. Examine contents of a file. Use the "C." option of DOS and you can observe the contents of nearly any file with a directory entry. After you hit C you will be prompted with COPY-FROM, TO and then type in the filespec followed by ",E:" (do not type the quotes). This will send your file to the screen. If you want to print the file out, use ",P:" instead. If it is a machine language program, you may waste a lot of paper as there are probably many commands for TOP OF FORM which will reel off the paper.

2. Build a file. You can enter a BASIC program or build a text file without using BASIC. Use the C option, only this time send the output from the screen to the file. The format after you press C and get COPY-FROM, TO is E:, D:FILESPEC. After pressing Return, the drive will spin as it opens the file and makes an entry in the directory for the FILESPEC. You can now enter the file or program one line at a time. You must press Return at the end of each line, and lines must be correctly retyped once they are Returned. Use CONTROL-3 to exit this entry mode and your file will be complete on the disk. The BASIC program can be ENTERed from BASIC and then RUN.

3. Warmstart and Coldstart from DOS. Use the "M." option to simulate switching the machine off and back on (coldstart). When you get the prompt "RUN FROM WHAT ADDRESS", type F125 and Return. The system will clear memory and reboot. To do a warmstart, type F128 and your program will be saved. Remember though, all values will be reset and the cartridge in the left slot (if any) will take over. This is like hitting the "SYSTEM RESET" key.

DOUBLE DENSITY — The term DOUBLE DENSITY refers to the placement of data on a floppy disk. DOUBLE DENSITY disks are written with the same number of tracks and sectors (40 and 720) as single density disks, but the number of bytes placed in each sector is increased to 256 from 128. A modified DOS is required to read and write disks in DOUBLE DENSITY. Up to 180,000 bytes of data can be stored on a DOUBLE DENSITY disk. It may be necessary to use diskettes certified to DOUBLE DENSITY, but it is not a requirement as this is still a relatively low data density. The new line of Atari disk drives for use with DOS 3 are not DOUBLE DENSITY but are dual density. In this system the number of sectors is increased to 1,023 but the number of bytes per sector remains at the standard 128.

DOUBLE LINE RESOLUTION — Players are set up so that each byte is written with two horizontal scan lines on the television screen. To enable the single line resolution, bit 4 must be set in the DMACTL by adding 16 to the value in location 559.

DOWNLOAD — The communicating abilities of an Atari computer with a modem and proper terminal program allow you to copy programs from other computers into your own and save them to a disk or cassette. This process is called DOWNLOADing. A commercial terminal program will allow you to DOWNLOAD with relative ease. Jonesterm, which is a public domain terminal program, is capable of DOWNLOADing. The opposite of DOWNLOAD is UPLOAD, which is sending a program or file to another computer.

DOWNWARD COMPATIBILITY — Compatibility is important for computers. Software and hardware can be compatible or incompatible. If a new computer is compatible with old software, one can say that the software has DOWNWARD COMPATIBILITY.

DRAWTO (DR.) — In BASIC, DRAWTO is the command to draw a straight line from the last point PLOTted to the X,Y coordinate named in the DRAWTO statement.

```
EXAMPLE: PLOT Ø,Ø: DRAWTO 2Ø,3Ø
```

DRIVESPEC — This term refers to the disk drive involved in your input or output operation. This could be D1, D2, D3, or D4. If no number is specified after D, then the default drive is always D1. Drives are configured by the hardware switches on the back of the Atari 810 Disk Drive. Two switches, white and black, are positioned in one of four combinations to define the DRIVESPEC number of that drive.

DUP.SYS — DUP.SYS is the name of the file which contains the Disk Utilities Package. These utilities are the routines for locking files, copying disks, duplicating files, deleting files, etc. The DUP.SYS file must be present on a disk for you to load in these utilities, but it is not required for you to boot in DOS. The DOS.SYS file is the File Management System required for booting DOS. Without DOS.SYS in DOS 2.0, a BOOT ERROR message will appear on the screen unless a proper boot file exixts on the disk.

DURATION — In Atari Microsoft BASIC, the SOUND statement operates like the Atari BASIC statement with the addition of a DURATION expression. This expression is optional and it will specify the amount of time a tone will sound off in seconds/60. The expression is the fifth after the SOUND statement:

```
SOUND N,V,F,D,T
```

where N is the voice Number, V is the Volume, F is the Frequency, D is the Distortion, and T is the DURATION in 1/60ths of a second. With no DURATION expression, the sound remains on until an END or RUN statement is encountered.

DYNAMIC RAM — DYNAMIC RAM is a type of Metal-Oxide-Semiconductor (MOS) memory used for computer memories. DYNAMIC RAM is often abbreviated DRAM. This type of memory must be refreshed every few milliseconds by internal refresh circuitry. DRAMs are very inexpensive and prices are dropping even more as chip capacity increases and competition heats up.

Static RAMs, in contrast to dynamic RAMs, are not refreshed continuously and thus run cooler and require less power. They are more expensive and are not usually used in home computers.

E

EBCDIC — EBCDIC stands for Extended Binary Coded Decimal Interchange Code and it was developed by International Business Machines Corporation for use on their System/360 and 370. EBCDIC is an eight bit code for characters similar to ASCII and ATASCII, but it is not compatible with either. See ASCII.

EIGHT BIT MACHINE — All computers based on the 6502 microprocessor from MOS Technology, Inc., such as the Atari home computer line, are known as eight bit machines. This terminology derives from the way the data is handled by the processor and the way memory is allocated in the computer. Data is processed in bytes which consist of eight parallel bits. In other words, the smallest operation that can be performed, and the largest for that matter, is the handling of eight bits at a time. Eight bit words, or bytes, are loaded into the accumulator of the 6502 and worked on by the program. In order to address all 65,536 bytes of memory, two bytes are used to address each memory location. The fact that only eight bits can be processed while many operations call for 16 bit addresses limits the speed somewhat, because two bytes must be used to indicate every address. The largest number that can be operated upon in any one instruction is 255. Sixteen bit machines are generally faster than eight bit machines when doing number oriented operations. For text processing in general, eight bit machines are as good as 16 bit machines.

810 DISK DRIVE MODIFICATION — If you have an 810 Drive with an analog speed control board, it is possible that you have had some problems writing and formatting disks. The cause of these problems may be insufficient current to the read/

write head during write operations. This problem is probably the result of an improper component installed on the analog board. You could make the correction yourself if you are moderately handy with a soldering iron and the warranty has expired on your drive. It is likely that all drives with this problem are beyond warranty periods by now. You might consider the modification if you have frequent recurrence of the following symptoms:

1. Difficulty writing to disks formatted by another disk drive.
2. Trouble formatting your own disks.
3. Abundance of ERRORs 138, 144, 163, and 173.

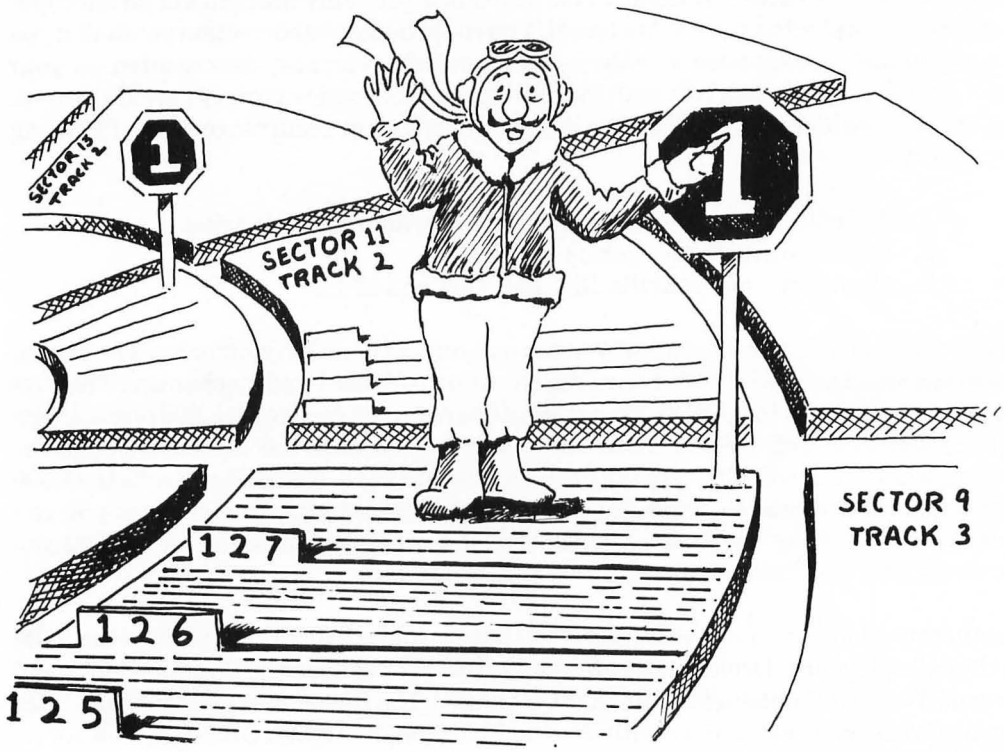
This modification is only possible if you have an analog board in your drive. The analog board is a small printed circuit board which sits up over the head mechanism. The cure is to replace one resistor (R218). The value of the installed resistor is 2,000 ohms. If you replace it with one of 1,400 or 1,500 ohms, you will increase the current through the writing head which will alleviate the problems listed above. Radio Shack sells the 1,500 ohm resistor for about 15 cents. Be careful with the soldering iron because you can overheat and destroy a transistor which is near the resistor. Also, be aware that you are doing this modification at your own risk, so be careful.

Another revision you should check for if you have an older 810 is the version of the ROM on the 810 sideboard. Drives made after May 1982 should have the C version, not the B version. The part number of the ROM is C011299C. The Revision B ROM is C011299B. The new version formats disks with the "fast" format. See SECTOR NUMBER for an explanation of fast format.

ELECTRONIC SPREADSHEET — VisiCalc (TM VisiCorp) was the first commercial implementation of an ELECTRONIC SPREADSHEET. This program sets up a matrix in the computer's memory. The matrix is made of variables and constants which can be interrelated. By changing one constant, all variables related to it will be automatically updated. Syncalc by Synapse Software, Inc. is another spreadsheet program for the Atari computers.

END — In BASIC, the END command is optional. Most programs will END themselves or will be ENDed when the machine is powered down. In the immediated mode, END is useful to turn off SOUND statements which will not be shut off with a BREAK. END also CLOSEs all channels which may have been OPENed.

END OF FILE — The END OF FILE message usually signifies that a disk read error has occurred. The end of a file is detected by several events. In the last byte of a sector, byte 127, the first seven bits are a representation of the number of bytes in the sector. This number in a full sector is normally 125 or 7D in hexadecimal. In the last sector of a file, this number is often less than 125 as the sector may not be full. In this case the last bit of byte 127 will be a 1, meaning that the sector is not filled up. The next sector link, which is 10 bits in bytes 126 and 125, should also be 0 if the sector truly contains the END OF FILE. If a mix-up occurs and the disk drive tries to read a different file during the reading of the current file, an END OF FILE error will occur. This usually signifies that some extensive damage has been done to the data disk and a disk utility must be used to correct it.



END OF LINE — See EOL.

ENTER(E.) — In Atari BASIC, ENTER is used to transfer a LISTed program from a disk or cassette tape into memory. The lines from the ENTERing program will be merged with any BASIC program resident in memory. This is handy for adding a subroutine utility to the end of a BASIC program, but it can make a mess if the two programs have common line numbers. Remember that the line numbers will be merged, so the ENTEREd program should have very high line numbers (like 32000) if it is to go on the end of the program in memory. Variables are added to the variable name table if any new ones are encountered.

ENTRY POINTS — An ENTRY POINT or VECTOR is the starting location in memory of a routine which does a certain task. The ENTRY POINT may be a fixed location in memory, or it may be specified by a program which was loaded in by the user. A USR statement can be used to jump to the address of the ENTRY POINT to begin the routine there. The routines may be for a floating point operation, a warm start, etc. Control is jumped over to the entry point address and things happen from there. The address must be between 0 and $FFFF in the Atari and most 8-bit computers. There are legal and illegal ENTRY POINTS. Legal ENTRY POINTS are recommended and documented by Atari documentation for programmers. Illegal ENTRY POINTS will often work until a revision is made in the Operating System, such as the Rev. A to Rev. B conversion which occurred recently. The Atari 1200XL Operat-

ing System and the XL series Operating Systems contain several changes which will render some previously existing software incompatible. An illegal ENTRY POINT is often in lower memory where a programmer finds some unused RAM and which may be interfered with by new programs written for a new OS.

EOF — In Microsoft BASIC, the EOF(n) command is used to detect an end of file byte transferred through the serial bus in the last reading through IOCB n. EOF(n) acts as a flag variable where true is 1 and false is 0.

EOL — The EOL character (End Of Line) is generated by hitting the RETURN key. This is ATASCII character 155 ($9B). The EOL character forces a carriage return and line feed.

EOR — Exclusive OR. EOR is a logical operator that compares two numbers and generates a result. EOR actually compares the byte (comprised of eight bits) which is the number in hexadecimal notation. In each respective bit, one number is compared to a bit in the other number. For each position in which ONLY a single 1 appears, a 1 is produced in the result.

Exclusive OR:	00111100	=	$3C
With:	01010101	=	$55
Results in:	01101001	=	$69

EPROM — Erasable Programmable Read Only Memory devices are used for low cost, usually prototype quantities of program storage. EPROMs are semiconductor devices which can be programmed with a special device. The program may be a game, application, device instruction, or other piece of code which may be changed by the writer at some time. EPROM programmers are relatively inexpensive as are the actual devices, especially when compared to custom integrated circuits. The device package has a clear quartz window in its lid which allows the chip inside to be exposed to high intensity ultraviolet light and erase the program in memory. The chip can then be reprogrammed and used again as if it were brand new.

ERL — In Microsoft BASIC II, ERL takes on the value of the line number where the last error occurred.

ERR — In Microsoft BASIC II, ERR takes on the value of the number of the last error generated.

ERROR — In Microsoft BASIC II, ERROR X causes the program to generate the ERROR designated by the variable X. Why would anyone want to include errors in the program? ERROR can be used for debugging or error handling investigation and they should be removed for the final program version.

ERRORS — You may generate an ERROR in the Atari computer system in a variety of ways. By checking the ERROR code against a table of explanations of the ERRORS, you can start to find the cause of the trouble. Codes between 2 and 21 refer to problems with a BASIC program or trying to load a BASIC program. These codes are generated

by the BASIC cartridge program. Program ERRORS coded between 128 and 171 are CIO STATUS ERRORS and are related to input and output faults to and from disk drives, printers, cassette recorders, or RS232 devices. These codes are actually generated by the CIO when an operation is not completed properly. Unless a $01 byte is returned from the device after a CIO call, an ERROR is generated. The following explanations will help in tracking ERRORS.

CODE	PROBLEM
2	Insufficient memory for statement, new variable, or string dimension. Go through program and free up memory by using memory conservation techniques. For example, use variables for numbers used more than three times, use multiple statements on each line, and use short variable names.
3	Value Error such as a result outside the range of +/–10 to the 97th power (or to the 127th power with the Newell Fastchip).
4	Too many variables. A maximum of 128 variables can be assigned. LIST program, type NEW, and ENTER program to remove unused variables. If you merely SAVE the program, all of the unused variable names will also be saved. This does not occur when you LIST and ENTER a program.
5	String Length Error. Check the string DIMension. You probably have a string which is longer than the DIM statement allows.
6	Out of DATA caused by a READ loop or statement trying to read past the end of a DATA list. Check the FOR/NEXT loop or see if you left our some data.
7	Number Greater than 32767. Integers must be less than 32767.
8	Input Statement Error. Make sure INPUTS for floating point numbers do not contain any alphabetic characters.
9	Array or String DIM Error. Make sure all strings or arrays are DIMensioned. DIMensions cannot exceed 32767. Variables cannot be DIMmensioned twice.
10	Argument Stack Overflow. Results from too many nested levels. It is very hard to generate this error unless you write a very complex program.
11	Floating Point Overflow. Usually results from trying to divide by zero.
12	Line not Found. GOSUB, GOTO or THEN must be followed by an existing line number. This error may occur if you renumber a program which has statements such as LINE=5ØØ: GOTO LINE.
13	Non-matching FOR. The variable in the FOR statement must have a matching variable in a NEXT statement.

CODE	PROBLEM
14	Line Too Long. Use colons or break up statement into shorter lines. You cannot use more than 128 characters per line. The editor will not let you type in more than 120, so you must abbreviate (LP., GOS., etc.) if you really want to pack in the statements.
15	GOSUB or FOR deleted. Cannot find the corresponding GOSUB or FOR for a RETURN or NEXT command.
16	RETURN Error. Program found a RETURN before processing a GOSUB command.
17	Syntax Error. BASIC found a defective line in the program. Since syntax is checked when the program is LOADed or ENTERed, this means either you POKEd some garbage into the BASIC program area of memory or you have some defective RAM in the program area.
18	VAL Function Error. The string in a VAL command must be a numeric value.
19	LOAD Program too Long. Shorten BASIC program, or if you do not have 48K, get a RAM upgrade. If you are LOADing a cassette program, try LOADing without DOS booted in.
20	Device Error Number. Device number must be between 1 and 7. Channel 0 is not available.
21	LOAD file Error. LOAD is used for tokenized BASIC files only. This error is generated when you try to LOAD a binary load file, a LISTed file, or a CSAVEd file. Try to use the conventional extender (.BAS) for SAVEd programs, and .LST for LISTed programs. This should cut down the number of errors.

The next group of errors (CIO errors) are caused by input/output faults to the printer, disk drive, cassette, or 850 interface.

CODE	PROBLEM
128	BREAK Abort. Caused by hitting the BREAK key during input/output operation. On Revision A Operating System computers, the system would often go to sleep during I/O. Hitting BREAK is a non-fatal cure for this symptom.
129	IOCB Already Open. The device channels can only be OPENed once or an error will result. Don't put OPENs in a loop. Graphics statements use 6. Channel 7 is also used to stick to 1 through 5.
130	Non-existent Device. The device (D:, P:, R:, E:, etc.) must be present in the handler table (located at 794-831 in RAM). New devices can be added. Note that R: is not added unless the 850 interface is on during start-up of the com-

CODE PROBLEM

puter. Also make sure there is a "D:" in front of the disk file you are trying to access.

131 IOCB Write Only. The channel through which the activity is being tried is OPENed for writing only. No reading is permitted (GET or INPUT in BASIC).

132 Illegal Handler Command. The CIO has received an invalid command. The XIO command structure is probably defective.

133 Device/File Not Open. Make sure the syntax of the OPEN statement is correct. An END statement will CLOSE all channels. BREAK or ERROR will not.

134 BAD IOCB Number. Use only channels 1 through 7, preferably 1 through 5.

135 OPENed for READ Only. The channel through which activity is being tried is OPENed for READing only. No writing (PRINT or PUT) is allowed.

136 End of File. The program either tried to input from a file which had run out of data or else started reading from a file that was not opened. This usually is a sign of a bad sector link. Use a disk utility, such as Diskey, to trace the file in question.

137 Truncated Record. This error occurs when the record being fetched is longer than the maximum length specified by the CIO call. This occurs when using a record based command (INPUT) on a file created with a byte oriented structure (PUT).

138 Device Timeout - Serial bus error. The device receiving the command did not respond within the time allowed (about four seconds). In most cases the disk drive or printer is not switched on and ready. Check all cable connections. Check device number, especially the black and white slide switches on the back of the 810.

139 Device NAK. (Serial Error) Device cannot respond due to bad set-up parameters (baud rate, word size, bad sector, etc.) Check the cables to the device being accessed.

140 Serial Frame Error. (Serial Error) Device is sending garbled data back. Bit 7 is SKSTAT (53775) is set. If it happens often have your computer checked out.

141 Cursor Out Of Range. Happens when you try to plot or draw off of the coordinates of the screen. Put a chopping routine to stop at the limits for your graphics mode.

142 Serial Bus Frame Overrun. (Serial Error) System did not respond fast enough to the input interrupt. Bit 5 in SKSTAT (53775) is set. If it happens often have your computer checked out.

CODE	PROBLEM
143	Checksum Error. (Serial Error) Checksum sent by the peripheral is different from that calculated by the computer. Recording media could be defective.
144	Device Done Error. (Serial Error) Device cannot complete a valid command. Usually means you have a write protect tab on a disk to which you are trying to write.
145	Illegal Screen Mode. Occurs when editor is OPENed in illegal graphics mode number. Also-Read After Write Compare Error. Disk drive notes difference between what was written and what came in for writing.
146	Function Not Supported By Handler. Occurs when you try to input to an output only device or vice versa. Keyboard is input only, printer is output only. Does not occur with D:.
147	Insufficient RAM. Graphics B requires over 8K just for the screen memory. Add memory or use a lower graphics mode to solve problem.
148	No Error Assigned.
149	No Error Assigned.
150	Port Already Open (RS232 Error). An attempt was made to open a serial port through another channel while it was already OPENed. An RS232 serial port can only be accessed through a single channnel at a time.
151	Concurrent Mode I/O Not Enabled (RS232 Error). A serial port must be OPENed for concurrent I/O with an XIO 40 command using an odd number task.
152	Illegal User Supplied Buffer (RS232 Error). Inconsistent user supplied buffer length and address were specified in the Start Concurrent Mode I/O command.
153	Active Concurrent Mode I/O Error (RS232 Error). Results from attempts to conduct I/O to RS232 port while concurrent mode I/O is active. System usually crashes and you never see the error message.
154	Concurrent Mode I/O not Active (RS232 Error). Concurrent mode I/O must be active to perform GET or INPUT commands over the R: device.
160	Drive Number Error. Drive numbers can only be in the range of 1 to 8. Check location 1809 ($710) to see how many buffers are allocated for drives.
161	Too Many OPEN Files. You can have a maximum of three files OPEN at any time. There are three sector buffers of 128 bytes each assigned by DOS. To OPEN more, POKE the number you want (up to 7) into location 1801 ($709) and write the DOS back to disk with the H. option.

CODE PROBLEM

162 Disk Full. Occurs when the VTOC indicates no free sectors are available. This may not be the case in reality due to errors in closing files or writing the map back to sector 360. You can usually just delete the filename and regain your sectors back. Sometimes the VTOC will get scrambled and all of the good files must be transferred to another disk.

163 Unrecoverable Systems I/O Error. DOS version or copy has a bug or defect. Use a new copy of DOS.

164 File Number Mismatch. Sector links between sectors of a file are inconsistent. This could happen in any number of ways and it usually does. If you save and delete many times to a single disk, you are bound to find this error sometime. It is very educational to try to correct this error. The forward sector reference points to a sector which is part of another file number. Sometimes it is a matter of using a disk utility to patch the forward references to the next sector on the disk. If data has been written after the original miswriting occurred the damage is probably extensive. Get out Diskey or Disk Fix and try to recover, otherwise, reformat the disk.

165 Filename Error. Illegal Characters in the filename. Probably put in by a disk utility program.

166 POINT Data Length Error. Byte count in the POINT call was greater than 125 (or 253 for double density disks).

167 File Locked. Occurs when you try to delete or change a filename, or write to a locked file. Unlock the file from DOS.

168 Device Command Invalid. Illegal XIO command.

169 Directory Full. Only 64 entries are allowed in the directory. This is a rare error.

170 File Not Found. Occurs during an attempt to access a file not listed in the directory.

171 Point Invalid. Attempt to POINT to a byte in a file not OPENed for an append. Look at the parameters of the IOCB which OPENed the file.

172 Illegal Append. Attempt to OPEN a DOS 1 file for append with DOS 2. File must be copied to a DOS 2 diskette for append operation.

173 Bad Sectors during Format. Drive could not format some sectors during FORMATting. If this occurs often, you may need work on the read/write part of the drive.

ESCAPE CODES — The ESCape key on the upper left hand corner of your keyboard is a remnant of the old Teletype terminal keyboards. ESCape is used by many applications programs (such as Letter Perfect) to return back to the menu of options. Some other key could have arbitrarily been chosen. In BASIC, ESCape is used to decontrol the editing keys on the Atari so that the ATASCII characters can be put into strings without doing the editing work on the screen. Suppose you wanted to place an UP ARROW or the ATASCII 28 character in a string. By hitting the ESC key, then CTRL, and then plus sign (+), the up arrow would be placed on the screen. By using ESC, one does not get the editing function of moving the cursor up one line.

EXP — In BASIC, EXP is the function which raises "e" to the number supplied in parentheses after the EXP expression. The "e" is Euler's number, which is 2.718281828 . . .

EXECUTE — A command or statement is EXECUTEd when it is sent through the E: device to the Central Input/Output utility. An entire logical line is sent for execution when the cursor is on the line and the RETURN key is pressed. If the line is set up so it is suitable for proper execution (no ERRORs in BASIC, for example), and the program is ready to accept input, then the line will be executed. If a line number is in front of the statement, BASIC will not EXECUTE the line immediately. If there is no line number, BASIC will assume the statement is for the immediate mode and will try to EXECUTE upon the pressing of RETURN.

EXPONENTIATION — EXPONENTIATION is the mathematical operation of raising a number to a power. Two to the power of three is 2 ^ 3 (2 × 2 × 2 or 8). To EXPONENTIATE a number, use the symbol " ^ ". Other languages use two asterisks to indicate EXPONENTIATION. To EXPONENTIATE e to the power of a number, use the BASIC comand EXP. The format for EXP is: GROWTH = EXP(VARIABLE). In Atari BASIC, EXPONENTIATION is very slow. You can speed up EXPONENTIATION by multiplying your numbers instead (A ^ 3 = A*A*A).

EXPRESSION — A mathematical EXPRESSION is a combination of operators and operands that calculates a new value. An operator is used to conduct some activity, like add (+), subtract (—), multiply (*), divide (/), exponentiate (^), etc. The operand is a variable or constant which is operated upon by the operator: 2+2, VAR1+VAR2, LOG(2), etc. The statement which may contain many operators and operands is an EXPRESSION. EXPRESSIONS which contain > and < symbols are called relational expressions because one operand is related or compared to the other.

F

FAN FOLD PAPER — Many of the low cost dot matrix printers for use with personal computers such as the Atari use FAN FOLD PAPER. There are advantages and disadvantages to this type of paper. The advantage is that the paper is very low cost, about $25 for 3,000 sheets. The paper is usually very light weight and all sheets are connected

on the top and bottom edges. There are perforations on the sides used for pin feeding through the printer's tractor mechanism. When these perforations are torn off, the paper is left with a ragged edge. This is fine for rough drafts and informal letters. Unless your printer has the tractor feed mechanism to keep the FAN FOLD PAPER aligned during printing, you will probably have some shifting and misalignment after printing more than a few pages.

FASTCHIP — Newell Industries has designed a ROM chip which fits into the 10K Operating System cartridge for the Atari 800. The chip has a revised floating point routine which speeds up multiplication and higher order math functions by about 30 percent. This chip expands the range available for scientific notation numbers from +/– 97 to +/– 127. This upgrade is worthwhile if you are doing extensive number crunching, plotting, or graphics with your Atari.

FETCH — This term is a verb meaning to go to a specific memory location and retrieve a value or data from that location. This is what the ANTIC chip does during DMA (Direct Memory Access). FETCH is a standard FORTH word which is written "@".

FIELD — A FIELD is an area within a larger collection of information. In a database manager, a FIELD is an entry within a record such as the address, last name, or ZIP code. In an assembly language source listing, a FIELD is the address, opcode, or comment part of a statement.

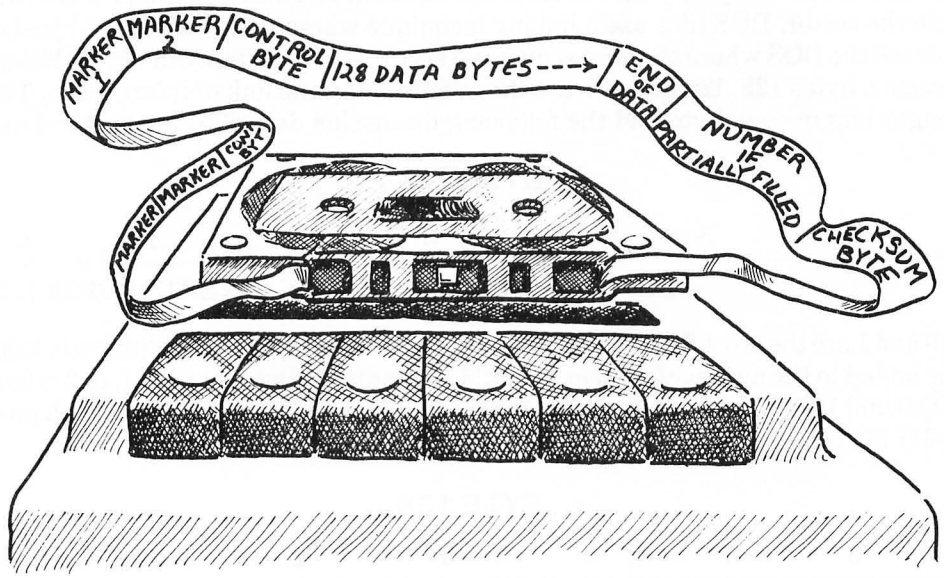
FILE — A FILE is a collection of data which can be saved on magnetic media (disk or tape) and loaded into RAM. A disk FILE is made of a series of sectors of 125 bytes of data and three bytes of linking information. An entry is made in the disk directory for each FILE on the disk. A cassette FILE has a 20 second leader of steady tone at 5,327 Hertz followed by a series of 132 byte records and then an End Of File (EOF) record. A cassette FILE record is comprised of two markers, a control byte, 128 data bytes, and a checksum byte.

FILE TYPES — There are several TYPES of FILEs used to save and load data into your Atari computer. There are cassette files and disk files. Cassette files can be SAVEd, LISTed, or autoboot type (see FILE-CASSETTES). Autoboot cassette files load directly into the computer when you turn on the switch with the START button depressed with the play button on the program recorder on. Disk files can be of many different types. See DISK FILES. The following list describes some of the disk file types.

NAME	DESCRIPTION
AUTORUN.SYS	Machine language program which is run on starting up.
BASIC-SAVEd	Tokenized BASIC file.
BASIC-LISTed	BASIC program which has been LISTed to a disk file.
BINARY	Machine language Binary Load file.
DATA	Strings of data for a database program.
DIF	Used by VisiCalc to format the spreadsheet data for interchange.
DOS.SYS	The Disk Operating System program.
DUP.SYS	The utilities package for DOS.SYS.
TEXT	Strings of alphanumeric data created by word processing programs.

FILE UTILITY — Letter Perfect uses a proprietary DOS which is not directly compatible with the Atari DOS. The directory starts in sector 363 in LJK DOS instead of in sector 361 as with Atari DOS. The VTOC starts in 362 instead of 361 as in Atari DOS. The FILE UTILITY by LJK is a translation utility which will allow you to make Atari DOS text files from Letter Perfect files. LJK Enterprises.

FILE - CASSETTE — A CASSETTE FILE is comprised of three parts: a 20 second leader, the data blocks, and the End Of File marks. The leader is a steady tone which gives the SIO time to synchronize with the data. The data blocks are made of 128 bytes each except for the last one which may or may not be shorter. The End Of File (EOF) block tells the system that all data has been transferred. A cassette data file must be OPENed in order to read or write data. In BASIC, a statement such as OPEN #2,8,0,"C:" will open IOCB #2 for output from the computer; in other words, write to the cassette. A 4 in place of the 8 in the preceding statement will READ data from the cassette. A cassette file cannot be OPENed for reading and writing at the same time (unlike a disk file), because you must push the RECORD button to write to the cassette. Every file which has been OPENed must be CLOSEd by a statement such as CLOSE #2, END, or RUN. If you try to OPEN an already OPEN channel, an error will occur.



FILE - DISK — DISK FILES, like cassette files, must be OPENed for input or output. The format for OPENing a disk file is OPEN #2,n,0,"D:". In addition to reading and writing, disk files can be opened for appending (using a 9 for n), read and write or updating (using a 12 for n), or to read the disk directory (using a 6 for n). Any OPENed channel (#2) must be CLOSEd in order to use the channel again. Data sent through an OPEN channel to the disk is sent through a buffer of 128 bytes. Only 125 bytes can be used and when the buffer is full, the contents are dumped out to the disk file. The contents of the disk buffer are also dumped when the channel is closed. Data can be printed to the disk file, just as to a printer using a statement such as PRINT #2;X. This

assumes that #2 has been OPENed as a disk file for writing. An End-Of-Line (EOL) character, which is ATASCII character 155 (inverse ESCape), is placed at the end of a string sent to a disk file. A comma separating the strings or variables will insert nine blank spaces with no EOL character. A semicolon will append the data with no spaces or EOL's. To force an EOL between data, use a CHR$(155) or separate print statements for each item. A PUT statement from BASIC can be used to send an individual ATASCII character (0 to 255) to a disk file. No EOL is generated after the PUT character. A GET statement from BASIC is the opposite of PUT. It is used to read data from a file, byte by byte.

When a file is OPENed for output, it must be rewritten entirely. To add data to the end of a file, or to append to it, the file must be OPENed for a number 9 mode operation. The file name must be represented in the directory or an ERROR 170 will be generated. All data sent out in mode 9 will be added to the end of the file. A file can be OPENed for updating (reading and writing) by using mode 12. Data is read and can be replaced within a sector when updated information is available.

There are two types of data files, random access and sequential. Random access files are controlled through NOTE and POINT commands in BASIC. These commands need an OPEN channel. Specify an absolute sector number on the disk and a character within the sector. DOS files use a linking technique whereby the last three bytes of a sector tell the DOS where the next sequential sector is to be found. Out of the 128 bytes per sector, bytes 125, 126 and 127 are those containing the link of information. There are eight bits in each byte and the following discussion describes the protocol used.

BYTE 125

7	6	5	4	3	2	1	0
FILE NUMBER						NEXT SECTOR (MSB)	

Bits 0 and 1 are the Most Significant Bits in the next sector reference. Either a 0, 128, or 256 is added to the number found in byte 126 depending on whether a 0, 1, or 2 is found in bits 0 and 1. The last six bits define the file number. Up to 2 raised to the 6th power (2^6=64) files can be on an Atari DOS 2.0S disk.

BYTE 126

7	6	5	4	3	2	1	0
NEXT SECTOR (LSB)							

The number in byte 126 is between 0 and 255 (decimal). This number is added to the number found in bits 0 and 1 in byte 125 times 256 to get the next sector reference. Byte 126 and the two bits in 125 are 0 if the sector is at the end of the file.

BYTE 127

7	6	5	4	3	2	1	0
NUMBER OF BYTES IN SECTOR							

Byte 127 is the number of bytes in the sector. Since most sectors have 125 (128 minus 3), we find a $7D in this place most of the time.

FILE MANAGEMENT SYSTEM - FMS — The part of Atari DOS which handles the reading, writing, and updating of files is the FMS. This part is booted in if the DOS.SYS file is present on the disk when the computer is turned on. The FMS boot record is in sector #1. The activities which are supported by the bootable FMS are: OPEN FILE, OPEN DIRECTORY, CLOSE, GET CHARACTERS, GET RECORD, PUT CHARACTERS, PUT RECORD, GET STATUS, NOTE, POINT, LOCK, UNLOCK, RENAME, and FORMAT. Up to four disk drives can be accessed with the FMS. DOS files of 128 bytes are numbered from 1 to 720 by the FMS.

FILE POINTER — In order to use random access to any data stored on an Atari DOS disk, a FILE POINTER is used. Random access is an information locating technique which finds a piece of information (sector and byte number), and goes directly to that data without reading every piece ahead of it. The NOTE and POINT commands in Atari BASIC allow one to go directly to a specific sector and read any byte from 0 to 124 within that sector. Bytes 125, 126, and 127 are used for linking sectors and not for writing user information. The NOTE and POINT commands work through an IOCB or channel which has been OPENed. The format for a NOTE command is:

NOTE #1, X, Y where #1 is an OPENed channel, X is the sector number from 1 to 719 and Y is the byte within the sector from 0 to 124.

NOTE checks the sector and byte locations for the current FILE POINTER. These values should be saved and associated with some record which can be recalled by POINTing back to this same location and INPUTting the data. The DOS maintains the values for NOTE and POINT in memory locations 44, 45 and 46 ($2C, $2D and $2E). Locations 44 and 45 contain the low and high bytes of the sector number. Location 46 contains the byte number within the sector.

POINT changes the FILE POINTER to a specified sector and byte by using the same format as in the NOTE statement. Data can then be written at any location on the disk.

FILENAME — The eight letter character string used for file identification in the Atari DOS directory is the FILENAME. Identical filenames can be used if the extender is different.

FILENAME ENHANCER — You can modify DOS by changing the range of character that will be accepted as valid filename characters. The normal usable characters are the numerical and uppercase characters (ATASCII 48 to 57 and 65 to 90). By doing a `POKE 3818,33` and `POKE 3822,123` in Atari BASIC, you can extend the range from the exclamation point (!) through all lowercase characters (ATASCII 33 through 122). If you go back to DOS and write the DOS out to a disk using the H option, your new DOS will be modified to accept this range in filenames. Files with these lowercase letters in their names will not be acknowledged by a normal, unmodified DOS.

FILENAME EXTENDER — Up to three characters can be appended to the eight character filename in Atari DOS files. A period (.) is used to separate the filename from the extender. A suggested standard format for uniformity is to use .BAS to identify BASIC programs, .SRC to identify source listings, and .OBJ to identify machine language code. Additional suggestions which have circulated through users' groups are: SAP-Saved BASIC A+, LAP-Listed BASIC A+, SMB-Saved Microsoft BASIC, LMB-Listed Microsoft BASIC, DAT-General DATa file, and COM-Object code with RUN and INIT addresses appended.

FILESPEC — The filename and the filename extension comprise the FILESPEC. Typical representation is FILENAME.EXT.

FILL — In Microsoft BASIC II, FILL is the BASIC implementation of the XIO 18 command. FILL works with a line scanning from left to right and fills the area inside a group of plotted lines.

FINE SCROLLING — The Atari computer has hardware capabilities for FINE SCROLLING. FINE SCROLLING is the movement of graphic images bit by bit across the screen. This contrasts with coarse scrolling which moves an entire character at a time across the screen.

There are two bits which must be enabled for FINE SCROLLING. Bit 5 of the Display List instruction is the vertical fine scroll enable bit (add 32 to the Display list instruction). Bit 4 is the horizontal fine scroll enable bit (add 16 to the Display list instruction.) Then the number of clocks or scan lines to be scrolled must be stored in the appropriate scrolling register. The horizontal scrolling register location 54276 ($D404). The vertical scrolling register is 54277 ($D405). FINE SCROLLING will only work for 16 color clocks or scan lines. Then coarse scrolling must be used to bring in new screen data. See section six of *De Re Atari* for the full treatment of scrolling.

In the XL series, FINE SCROLLING is enabled by POKE 622,255 and typing a GR.Ø. The GR.0 issues an OPEN command for the screen editor. This routine starts at $026E or FINE. Using POKE 622,0 will disable the fine scrolling and revert back to coarse scrolling. A new display list is generated when a GR.0 or OPEN screen editor command is given and the FINE location contains a 255.

FIRMWARE — This term indicates a computer program which has been stored on a medium such as a ROM (Read Only Memory) chip or an EPROM (Erasable Programmable Read Only Memory). It is identical in function to software but exists in a more durable form. FIRMWARE is programmed into the ROM and it will remain even when power is removed. It is available to the computer almost instantly as it is either loaded in during boot-up or is part of the computer's memory. Software stored on floppy disks is not called FIRMWARE.

FLAG — A FLAG is a simple marker in memory which is switched on or off according to some event occurring. A FLAG is usually one bit in a byte. For example, the last bit of the last byte of each sector on an Atari DOS file is a flag. If there is a 1 in this bit, then the FLAG is up and it signals that the sector contains less than the full number of bytes. If there is a 0 in the flag bit, then the sector contains the full 125 bytes. A flag can be programmed to indicate any kind of two state situation.

FLOATING POINT ARITHMETIC — Computers can operate on integers (numbers without decimal points) much easier than on FLOATING POINT numbers. With a decimal point, care must be taken to always put the point in the right column or the answer will be wrong. This problem does not exist when you deal only with integers. The Atari computer dedicates around 3K of the memory in the BASIC cartridge, the OS ROM, and in RAM, just to minding the decimal points for floating points. The SIN, COS, arctangent, and SQR functions are in the BASIC cartridge. Other routines are stored in the OS ROM in locations between 55296 and 57343 ($D800 to $DFFF). The FASTCHIP from Newell Industries has a rewritten version of the floating point mathematics routines which speeds up some of the higher level operations. FLOATING POINT numbers must be within the range of -9.99999999E+97 and 9.99999999E+97.

FLOWCHART — Some programmers use a schematic diagram of the operation of their program during the programming cycle. The schematic is a model of what the program will do at various branching points and loops. The FLOWCHART is drawn on paper using well defined symbols for decision points, processes, activities and events, such as input, output, IF/THEN, compare, calculate, and END. While programming with a flowchart is still popular with large programs for large computers, very few hobbyists actually take the time to develop a FLOWCHART.

FMS — See FILE MANAGEMENT SYSTEM.

FONT — The design of a character is referred to as a FONT. A FONT is formed by individual pixels within a block. The Atari character FONTs are formed in an 8×8 block of pixels. For example, the built in FONT set can even be redesigned into fancy designs for adventure games.

FOR — In BASIC, FOR is the mandatory first part of a FOR/NEXT loop. FOR sets the variable and the range for the loop. If NEXT is not found with the same variable, an ERROR 15 will be generated.

FORCED READ MODE — It is possible to GET characters from the screen without using the RETURN key to initiate a reading. The FORCED READ mode will get all of the characters from the logical line where the cursor resides. The cursor will then be moved to the start of the next logical line. The FORCED READ mode is enabled by setting bit 0 of the AUX1 byte (right after the Device ID and Command byte sent in the serial bus command frame). You can do this by doing a POKE 842,13 and putting the cursor above the screen characters to read. You will need to return the command back to normal with a POKE 842,12.

FOREGROUND — The playfield is sometimes called the FOREGROUND. Characters or mapped data can be used to construct the FOREGROUND. See PLAYFIELD.

FORMAT — The Atari 810 disk drive uses soft sectored diskettes. This means that all of the sector identification and timing marks are set up randomly around the tracks. There is no timing hole requirement as the sector marks are set up during the FORMATting process. When a diskette is FORMATted, several data are written to the disk. The boot file is written to sector 1. The VTOC (Volume Table Of Contents) is set up in sector 360, and the Directory is started in sector 361. Zeros are written to all other sec-

tors. A disk can be FORMATted through the "I" option of the DOS menu. From Atari BASIC, FORMATting can be done by a XIO,#254,#1,4,0,"D:"<return> in the immediate mode.

Besides the obvious initial data which the FORMATting process writes on the diskette, there are other items which are important but are not visible nor accessible. These are the intersector identification marks. The 810 disk drive uses these marks to find sectors on a track, but there is no way to change or read these marks. In addition to the 128 byte sectors, there are three other fields of data: a pre-sector gap (six bytes of zeros-$00), an ID field and a post-sector gap (nine zeros-$00 and three ones-$FF). The ID field contains: one ID address mark, a $00 denoting sector size equal to 128, two cyclical redundancy checks, seventeen zeros, and a one byte data address mark. The Western Digital Corporation FD1771 floppy disk formatter/controller chip takes care of putting all of these marks between sectors. (See SECTOR NUMBER for a description of the sector layout written during the FORMAT operation.)

FORMATTED DISK — See FORMAT. A FORMATTED DISK is a disk that has been prepared to accept data for storage by a series of markers and control information. A FORMATTED DISK does not necessarily have the DOS file written on it, so even though a disk has been FORMATTED, it may not be able to boot the system. You must first prepare a blank disk for writing by using the master Atari DOS diskette and formatting with the I option. After formatting, you may want to put another copy of DOS on your new disk by using the H option to write new DOS files.

FORTH — FORTH is an interactive, threaded, interpreted language. Interaction is the job of the "outer interpreter." It takes commands, called words, from the input stream (usually the keyboard) and executes these commands. The words (or programs or commands) which are available for execution are contained in the FORTH dictionary. Most of the memory taken by a FORTH system is used by the dictionary.

Programming in FORTH is done by defining new words in terms of existing words. The FORTH words : and ; (colon and semicolon) are used to define new words.

```
:SAMPLE action1 action2 ;
```

The above definition adds SAMPLE to the dictionary. This new dictionary entry contains pointers to the dictionary entries of the FORTH words action1 and action2. When SAMPLE is executed, the words action1 and action2 will be executed. The word SAMPLE is now available to use in defining other words. It is now part of the FORTH system. Notice that this aspect is quite different from BASIC, or any other language for that matter.

Threading is the job of the "inner interpreter." When a word is executed, the inner interpreter threads through the pointers which make up that word. In SAMPLE (above), the first pointer points to action1. This word may also be made up of pointers to other words as action1 may have been defined in the same manner as SAMPLE. Eventually the inner interpreter reaches a primitive word. A primitive is not made up of pointers, but of directly executable machine code. After the primitive is executed, control is returned to the inner interpreter which continues threading until it reaches the end of SAMPLE. The CONTROL is then passed to the outer interpreter.

Several characteristics distinguish FORTH from other programming languages. Flexibility is the most important of these. FORTH is written in FORTH, so the user is free to redesign even the most fundamental features to suit a particular application. There are standards called FIG 79 and FIG 83 for FORTH Interest Group 1979 and 1983. These standards help users with different types of computers to transfer programs among themselves by using standard words. This is always an issue since any user can define new words.

FORTH is compact and runs fast. When pure FORTH is not fast enough, a FORTH assembler can be used to write machine language words. These words will still be compatible with the rest of the FORTH system.

FORTH's unified dictionary means that new programs can be built upon, and share code with, other applications. One major use of this feature is use as an operating system. The user can borrow OS routines, resulting in shorter and more unified code.

FORTH is in the public domain and several free versions are available through users' groups and FORTH groups. Commercial versions are enhanced and include custom utilities. Caverns of Mars was written in FORTH. PNS FORTH by Pink Noise Studios, Val-FORTH by Valpar, Extended fig-FORTH by APX, QS FORTH by Quality Software, FORTH by Elcomp, ECS/MVP-FORTH by Mountain View Press. Thanks to Stephen Malinowski and John Peters.

FRAMING ERROR — A FRAMING ERROR (ERRORS 140, 142, or 143) occurs when the timing of the response to data input or output is not correct. These errors only occur on devices connected to the serial bus (the disk drive, cassette, or interface). After data is sent in or out of the computer, it expects to see a COMPLETE signal. If something interferes with the data transmission and the operation is not complete, a FRAMING ERROR message will be generated. This is usually an indication of a hardware problem.

FRE — The BASIC command FRE returns the amount of free memory available in bytes. Free memory can be used for construction of BASIC programs or storage of strings in memory. FRE takes the value stored in MEMLO at locations 743 and 744 ($2E7 and $2E8) and subtracts this value from that of MEMTOP {locations 741 and 742 ($2E5 and $2E6)}. This difference is the amount of free RAM available for BASIC programming minus any buffers or handlers you have loaded in. The amount will change depending on the graphics mode, whether or not DOS is booted, or the amount of RAM you have installed in your system.

FROGGER GENRE — A multitude of games have appeared based on the theme of a small animal trying to cross a busy thoroughfare without getting crushed. The original game was called Frogger and the theme was a frog crossing a freeway. Some other games based on this theme are: Chicken-public domain, Preppie I and II -Adventure International, and Pacific Coast Highway-Datasoft.

FSK — Frequency Shift Keying — Another term for FSK is Frequency Shift Signaling. This is the technique used in modem communications. It is a type of frequency modulation where the frequency of the transmission is changed from instant to instant

according to whether 0 or 1 is being transmitted. During transmission, 0 will be represented by a signal at 1070 Hz in the originate mode and a 2025 Hz signal in the answer mode. During transmission, a 1 is represented by a signal at 1270 Hz in the originate mode and a 2225 Hz signal in the answer mode. By shifting between signals at 1070 and 1270 Hz, the modem sends a stream of ones and zeros through the telephone line to the other modem and computer.

FULL DUPLEX — FULL DUPLEX refers to simultaneous two-way independent transmission of data in both directions. Modems are usually set up for FULL DUPLEX transmission. Half duplex transmission works in both directions, but not simultaneously. Use the FULL DUPLEX made when communicating with another modem to echo characters back to your own screen.

G

GET (GE.) — The Atari BASIC command GET is used to retrieve one byte of data through an OPENed channel. GET can be used as an alternative to INPUT when no question mark (?) prompt is desired. When a program encounters a GET statement OPENed for the keyboard, it will halt execution until a key is pressed. An example of the use of GET is:

```
10 OPEN #1,4,0,"K:"
20 GET #1,KEY
30 PRINT KEY,CHR$(KEY)
40 CLOSE #1
```

GET retrieves the ATASCII value code from the key pressed. You must use an ATASCII conversion chart to find the value if you want to branch out from the GET input. GET can also be used to read single bytes from the cassette or disk devices. The cassette or disk buffer is filled with data bytes (128 or 125), with the first GET, and then the series of bytes in the buffer is read sequentially. At the end of the buffer the next sector or block is read in. GET can also be used to read a character from the location of the cursor on the screen (E: or S: device).

GLITCH — A GLITCH is a little bug which interferes with the intended operation of some software or hardware.

GOSUB — A BASIC program can contain subroutines which perform some activities out of the main flow of the program. The subroutine is identified by a line number as are all BASIC statements and it is ended by a RETURN statement. To enter a subroutine, include the GOSUB XXX statement in the main program and control will switch to line number XXX. Unless there is a RETURN, the program will continue processing sequential line numbers until the end of the program or until an ERROR results. A RETURN without a GOSUB will always cause an ERROR 16.

GOTO — In Atari BASIC, a GOTO causes the program to begin execution at the line named in the GOTO command. An ERROR 12 will result if the line number called does not exist. The GOTO command can use a variable as its line number expression. This

makes reading and debugging much easier but it will usually cause problems in compiling or renumbering by using a utility program. An example of a variable used as a line number is:

```
10 START=10
20 PRINT :PRINT "ENTER 1, 2 OR 3"
30 INPUT A
40 IF A=1 OR A=2 OR A=3 THEN GOTO A*100
50 GOTO START
100 PRINT "ONE":GOTO START
200 PRINT "TWO":GOTO START
300 PRINT "THREE":GOTO START
```

GOTOs and GOSUBs can easily make a program very complicated and are scorned by most hard-core hackers.

GRAPHICS CHARACTERS — The ATASCII characters between numbers 0 and 27 comprise the graphics set. These characters are generated by pressing the CTRL key plus an alphabetic character key. To see a list of the GRAPHICS CHARACTERS on your screen, try to RUN the following BASIC program.

```
10 FOR X=0 TO 26
20 PRINT X,CHR$(X)
30 NEXT X
```

GRAPHICS — The Atari BASIC GRAPHICS (GR.) command sets the GRAPHICS mode in any mode from 0 to 11. The format for the statement is GR.X, where X is a number from 0 to 11. When the GRAPHICS command is executed, the screen is normally cleared. Modes 1 through 8 also have a four line text window at the bottom of the screen. The screen clearing and text window options can be turned off. By adding 16 to the number after the GRAPHICS command, the text window is eliminated and the screen is extended by 64 scan lines. By adding 32, the clear screen feature is eliminated when the GR. command is executed. By adding 48 to the number after the GRAPHICS command, both the text window and clear screen are eliminated.

GRAPHICS UTILITIES — One of the most popular software genres is the GRAPHICS UTILITY. These programs allow you to draw on the TV screen using a joystick or tablet. Some programs allow you to save the images onto a disk file or to a printer. Some of the popular graphics packages are:

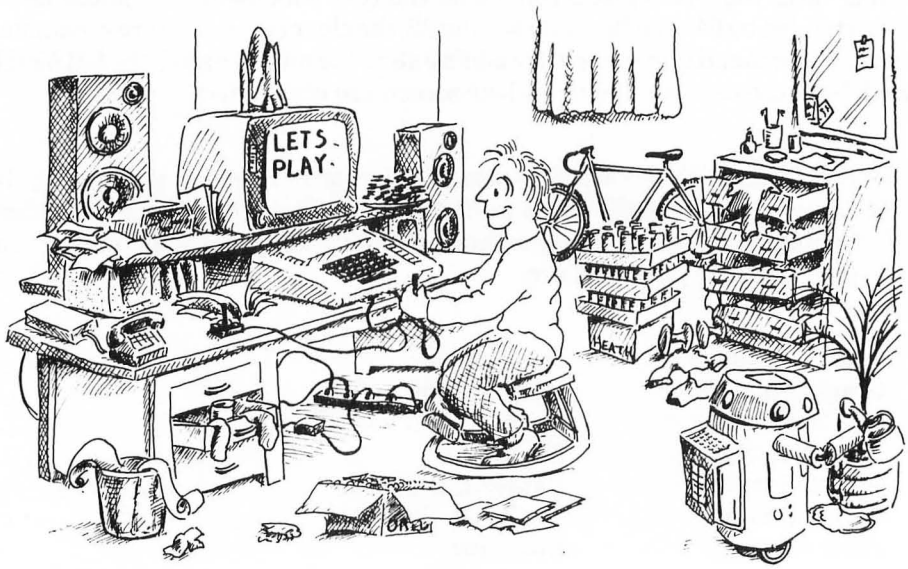
Graphics Master	Datasoft, Inc.
Graphic Generator	Datasoft, Inc.
Fun With Art	Epyx
Microillustrator	Koala
Micropainter	Datasoft, Inc.
PM Animator	Don't Ask Software
Paint	Atari, Inc.

GTIA — The GTIA processor replaces the older CTIA chip which was installed in most machines built before 1982. The GTIA is the Graphics Television Interface Adaptor (also called George's TIA after George McLeod who designed the chip for Atari). It handles graphics processing, specifically in these areas: priority control-overlapping objects on the screen, color-luminance control of all objects on the screen, players and missiles size control and horizontal position, collision detection between objects, and the keyboard switches and joystick triggers. The GTIA converts digital signals from the ANTIC to analog television signals. The registers for the GTIA are located between 53248 and 53279 ($D000 and $D01F) in the 400 and 800. They are in the range of 53248 to 53503 ($D000 to $D0FF) in the XL series.

The GTIA functions the same as the older CTIA except for the addition of three new graphics modes. These modes are called modes 9, 10 and 11, or GTIA modes 1, 2, and 3. All three of these modes have the same resolution, 80 pixels across by 192 pixels down. In Atari BASIC, the SETCOLOR and COLOR commands are used to implement graphics designs in modes 9, 10, and 11. In mode 9, the luminance value must be zero and the COLOR command is used to vary luminances. The value in the COLOR statement in mode 9 must be between 0 and 15 inclusive giving a total of 16 luminance possibilities. In mode 10, there are a possible nine colors in any hue and luminance combination. The COLOR command is used for switching colors (between 0 and 8). Mode 11 is like mode 9 with a possible sixteen hues of one luminance.

H

HACKER — Slang term for a person who has given up most of their normal interaction with society to play around with computer hardware or software for fun.



HALF DUPLEX — In modem communications, HALF DUPLEX refers to two way transmission of data which does not occur simultaneously. Most bulletin boards are set up for full duplex operation. If you modem or modem program is set up for HALF DUPLEX and you call a system which is on full duplex, you will see two copies of each key you hit; one from your computer and one which is bounced back from the remote system. Just switch your modem or program to full duplex to fix this situation.

HANDLERS — Device handlers are used to perform input or output operations through the Atari peripheral devices. Your program must call for some input or output through an IOCB (e.g., OPEN #1,4,0,"D:"). This call goes through the Central Input/Output (CIO) utility. The CIO then looks in a table of the available devices. The table has the device name and a list of addresses where the various utilities required for I/O are found. If necessary, the Serial bus I/O (SIO) is then used to communicate with the device. The table is called HATABS and is located at 794 ($31A). The table is 38 bytes long and can have up to 12 devices listed. Five device entries are loaded in from the OS ROM on power-up. These devices are: Cassette (C:), Display Editor (E:), Keyboard (K:), Printer (P:), and Screen (S:). If the 850 interface is on-line during start-up, the RS-232-C (R:) entry will be loaded into the table. Other devices can be loaded optionally by the user. HATABS is comprised of three bytes per device entry. The first byte is the ATASCII letter for the device name (P for printer, C for cassette, etc.). The next two bytes are the address pointing to the table of I/O routines. The routines listed in the table are: OPEN, CLOSE, GETBYTE, PUTBYTE, GETSTAT, SPECIAL, and JMP (initializing code). HATABS and the vector table look like this:

HATABS

ADDRESS	BYTE	NOTE
031A	45	(E:)
031B	00	Address
031C	E4	E400
031D	43	(C:)
031E	40	
031F	E4	
.	.	
.	.	
.	.	
.	.	

Vector Table

ADDRESS	BYTE	FUNCTION
E400	LO	OPEN Vector
E401	HI	
E402	LO	CLOSE Vector
E403	HI	
E404	LO	GETBYTE Vector
E405	HI	
E406	LO	PUTBYTE Vector
E407	HI	
E408	LO	GETSTAT Vector
E409	HI	

The Video80 program in the April 1983 issue of *Compute!* magazine adds a new handler device called "V:" to the table. This entry points to a vector table which gets characters from a table different than the standard Atari character set table and puts them on a graphics 8 screen. Since you are not using the E: (editor), you do not have full use of the cursor keys and other editing functions. When System Reset is pressed, the entire table is reentered from ROM and the new handler entry in the HATABS is lost.

HAPPY ENHANCEMENT — The HAPPY ENHANCEMENT is a hardware modification of the Atari 810 disk drive which upgrades the original drive in four ways. A track buffer is used whereby an entire disk track (18 sectors) is read in just 1.05 revolutions. This translates to about 0.2 seconds. The original 810 requires about three or four revolutions in order to read an entire track because no data is buffered. A new ROM program is provided which improves upon the new FAST FORMAT CHIP. This involves reading interleaved versus contiguous sectors for faster reading. An enhanced data/clock separator circuit is provided eliminating many of the errors found on inner tracks where data in compressed. Finally, the HAPPY ENHANCEMENT is guaranteed for five years to back up any commercially available software package sold on floppy disk for your Atari computer.

Several accompanying software packages provide additional features to the HAPPY ENHANCEMENT. A diagnostic program checks the circuitry of the enhancement board. A "Slow It Down" program is provided to convert your enhanced 810 back to the identical condition of an unmodified 810. This prevents any software manufacturer from trying to mischieviously lock out Happy modified drives. A compactor program allows packing of multiple single boot disks on one disk. All protection is maintained and the compacted programs will only run on a Happy modified drive. A customizer program is available to allow you to create your own custom protection schemes using bad sectors, missing sectors, bad CRC sectors, unreadable tracks, tracks which contain more than one sector with the same number but having different data and non-standard data marks. A multidrive system is available to permit fast copying by virtually simultaneous reading and writing from one drive to another. A "Warp Speed" DOS is available which speeds up the SIO transfer rate by about three times to over 40,-000 baud. Happy Computers.

HARD DISK — A HARD DISK (also called a Winchester drive) is a mass storage device used to store and rapidly retrieve many millions of bytes in memory. The recording medium is a ferromagnetic material coated very smoothly on a polished aluminum plate. The medium is very sensitive to contamination so the entire assembly is sealed in an air filtered container. The medium is usually not removable from the drive. It must be backed up for reliability on tape or floppy disks. A small HARD DISK about the size of an Atari 810 may hold 10 million bytes of data. You can use a HARD DISK with your Atari 800 by buying a Corvus HARD DISK and using the Integrater board as an interface. Corvus Systems and ADS.

HARD SECTORED DISK — Some disk drives (not the Atari types) use a hard sector format. These disks are identified by the 13 or 16 holes punched around the center hub which are visible through the tiny timing hole in the floppy disk cover. These holes are used to mark sectors as they are read by a light source shining through the holes. Atari computer based drives do not use these holes, but these disks can be used with Atari drives with no problems.

HATABS — This label refers to the locations in memory of the Handler Address Table. This table contains the device letter (ATASCII code) and the address vector to the handler routine. The table consists of 38 bytes located between locations 794 and 831 ($31A and $33F). The devices are loaded in from the OS ROM on start-up and the P:, C:, E:, S:, and K: devices are represented. Other devices can be loaded. See HANDLERS.

HEADS — The part of the cassette recorder/player and the disk drive which reads and writes the data to the magnetic medium. The HEAD is capable of generating a fluctuating field which magnetizes the recording medium during writing. It can also detect the fluctuations when it is passed over the medium during reading. The media (floppy disks and cassette tapes) contain a small amount of lubricant to reduce friction as the HEAD rubs over the media. After many years, the HEADs will wear and require replacement. Proper cleaning and maintainence will delay replacement beyond the life of your computer.

HELP — In the XL series of computers, the HELP key can be used as another console key, as well as to perform some rudimentary internal diagnostics during power-on. The HELP key is like the BREAK key in that when it is typed, no ATASCII character is produced, but a database variable is changed. The HELP key uses locations 732 ($02DC or HELPFG) for a database variable and 764 or $02FC to look for the last key pressed. In BASIC, you can check (or poll) location 732 by using a PEEK(732). The normal default value is 0, or 17 ($11) if the HELP key alone is pressed. To clear the HELP flag, type POKE 732,0. To check if a Shift and HELP key are simultaneously pressed, look for an 81 ($51) in 732. To check for a CTRL and HELP key, look for a 145 ($91) in 732. After the HELP flag location is used, it must be cleared by writing a zero to it. This is not done automatically. HELP is used in the internal diagnostic programs as a menu choice.

HERRINGBONE PATTERN — The HERRINGBONE PATTERN on the screen sometimes results from RF interference with the television signal. The source of the interference may be the computer itself if there is a leak or defect in the RF modulator. The interference can usually be cured by rearranging the RF cable to the television so that the interference cancels itself.

HEXADECIMAL — The HEXADECIMAL numbering system is based on 16 characters: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, and F. It is customary to represent HEX numbers with a dollar sign ($) preceeding the number to distinguish it from a decimal (base 10 number). Each column from left to right is a higher power of 16. The columns are valued: 1, 16, 256, 4096, etc.. All bytes stored in memory and on disk and cassettes are HEXADECIMAL numbers. Two HEX numbers make up a byte. It only takes four bits to represent any of the 16 HEX digits, so an eight bit byte comprises two HEX digits.

HI RES — HI RES is short for HIGH RESOLUTION. This term refers to the graphics modes, usually of some software based game. HI RES is usually considered GRAPHICS 8 or ANTIC mode "F". Most BASIC games are not written in HI RES because of the enormous amount of data which must be manipulated for animation. Machine language programs are required to do fast action in HI RES.

HI-RES — HI-RES is a magazine published by Compupress, Inc. with articles of special interest to all Atari computer owners. Magazines such as this one offer some of the best means of learning to use your computer.

HOME — To HOME the cursor is to send the cursor to the upper left corner of the screen. This occurs when the CLEAR screen operation is done.

HORIZONTAL BLANK — The time interval starting when the electron beam turns off on the right side of the screen until it turns on and begins writing on the left side of the screen is the HORIZONTAL BLANK. The amount of time required for HORIZONTAL BLANKing is 14 microseconds.

HORIZONTAL SCAN LINE — A television set operates by scanning a beam of electrons inside a tube over the inside face of the front plate. The beam starts at the top left and moves across to the right. This line is called a HORIZONTAL SCAN LINE. As the beam travels across the horizontal line, it is turned off and on, brighter and dimmer according to the information it receives from the control circuits. At the end of the screen, the beam is stepped down one trace and another line is drawn. The entire screen is drawn in less than 1/30 of a second. The beam then turns off and starts at the top left again. This time interval when the beam is off is the vertical blanking interval. There are 525 lines generated on an interlaced television screen (NTSC) every 1/30 of a second, but the computer generates 262 non-interlaced lines in 1/60 of a second. The ANTIC hardware uses 22 lines for the vertical blank leaving only 240 for display. The OS graphics firmware uses 192 to leave a neater margin on the top and bottom of the screen. The display list is used to control the structure of the screen.

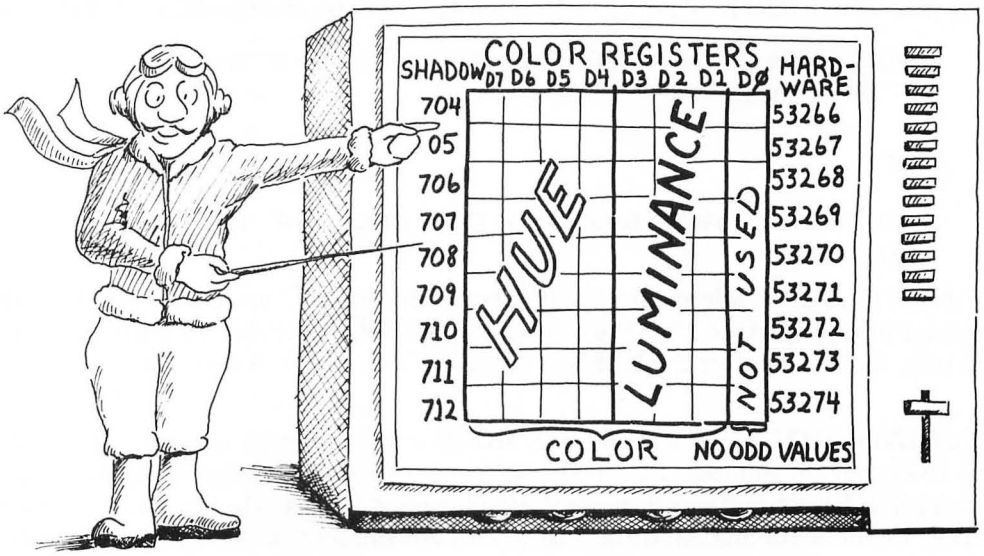
HORIZONTAL SCROLLING — Scrolling is the movement of data across the screen. In order to scroll horizontally, one must organize the data for the screen memory such that the screen area can be moved over it. The display list is then written to load the screen data from the reorganized data. To begin scrolling, the address of the first byte of screen memory is changed incrementally to move horizontally. Notice that this technique is different than moving the data past the screen. The screen is moving over the data. See FINE SCROLLING and COARSE SCROLLING.

HORIZONTAL SCROLL ENABLE — The register called HSCROL, located at 54276 ($D404), is the HORIZONTAL SCROLL ENABLE byte. The number of color clocks you want to scroll is put into this register. The number ranges from 0 to 16 (16 for coarse scrolling). In order to do smooth scrolling, you must also set bit D4 of the display list instruction (LMS, etc.). This means adding 16 to the instruction.

HORIZONTAL POSITION REGISTER — There are four memory locations which hold the value of the horizontal player position on the screen. These registers (HPOSP0-3) are located at 53248 through 53251, corresponding to players 0 through 3. Similar registers exist for missiles at locations 53252 through 53255. The HPOSP register can take a value from 0 to 227 and the corresponding player will appear on the screen at that location. The value of the register must be rewritten to move the player. It is reset to 0 immediately after it is written. Some positions will be too far off the screen and typically values between 40 ($28) and 220 ($DC) will make a player visible. See PLAYER.

HUE — HUE is half of the COLOR definition. The luminance is the other half. There are 16 HUES (they are often thought of as colors — orange, green, blue, etc.) and each HUE can can have eight luminances or brightnesses. This makes 128 possible colors. In GTIA GR. 9 with a few display list interrupts, it is possible to get 16 HUES with 16 luminances for a total of 256 colors.

The HUE is set by the left four bits of the color register. The luminance is set by the next three bits, and the last bit (bit 0) is unused. Color registers (the shadows) are located from 704 to 712 ($2C0 to $2C8). The registers are actually at 53266 to 53274 ($D012 to $D01A).



I

I/O — Input/Output — This term refers to the process or results of communicating with the computer. Without I/O the computer would serve no purpose. All peripheral devices are used for I/O. These include disk drives, cassette recorders, CRTs, modems, joysticks, paddles, trackballs, amplifiers, printers, graphics tablets, and anything else you can get to communicate with your Atari computer. The main external method of I/O is through the SIO (Serial Input Output). This is the trapezoidal plug which connects most peripherals. The controller jacks and the connectors for the RAM cards and OS boards also facilitate I/O. Internal I/O is handled by the Central I/O (CIO), which is a program in the Operating System.

IC — Integrated Circuit — The IC is the basis for all microcomputers. The IC is a device made from silicon which processes signals. The term "integrated" is used because each IC (chip) combines transistors, capacitors, and resistors in one tiny device instead of having each device in its own discrete package. Older four function calculators made from non-integrated electronics were several times larger than an Atari computer and only did basic arithmetic. The 6502, ANTIC, GTIA, Pokey, and all of the memory chips in the Atari computer are Integrated Circuits.

IF — IF is a part of a BASIC statement used for testing (IF/THEN or IF/THEN/ELSE statements).

IF/THEN — BASIC conditional branching statement. IF tests an expression, and IF it is true, the expression or command after the THEN is executed. If a number is supplied after the THEN, it is assumed to be a BASIC line number. Multiple IF/THENs can be placed on a line. Boolean operators are easier to use than multiple IF/THENs.

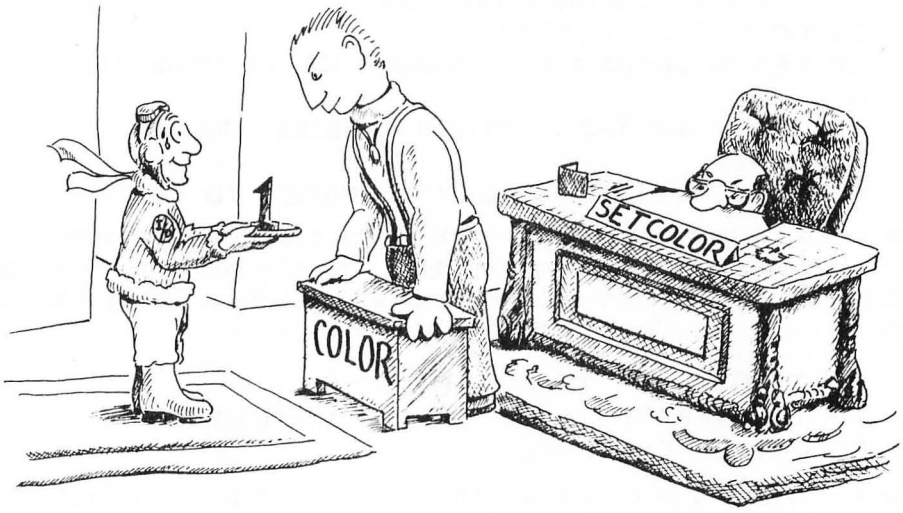
```
10 IF A=1 THEN IF B=2 THEN IF C=3 THEN I
F D=4 THEN 100

or

10 IF A=1 AND B=2 AND C=3 AND D=4 THEN 1
00
```

IF/THEN/ELSE — Microsoft BASIC Term. This BASIC statement looks at an expression and IF it is true, then the clause after THEN is executed. IF the expression is not true, the clause after the ELSE is executed. This can be a conditional branch.

IMMEDIATE MODE — IMMEDIATE MODE is a term which refers to the way in which the program counter (PC) in the 6502 processor points directly to data. IMMEDIATE MODE operations do not involve addresses of data to be loaded in or stored out. They use direct or immediate numbers. This is different than the other addressing modes, absolute and indirect. In the absolute or zero page mode, the PC points to the address of the data. In the indirect mode, the PC points to the address of the address of the data.

IMMEDIATE MODE — In BASIC, IMMEDIATE MODE refers to the processing of a statement without a line number. A valid BASIC statement is typed on the screen and the RETURN key is pressed. If the statement is valid, it will be processed in the IMMEDIATE MODE. You can use BASIC as a calculator if you are in a pinch. Get the READY prompt on the screen and type ?123 + 456 <Return>. You should see the result (579) displayed below the IMMEDIATE statement.

INDIRECTION — This term refers to the way a microprocessor obtains data to process. One way which is very direct is to just feed it a number in the immediate mode, LDA #$FF. This loads $FF (255) into the accumulator. This is very simple. A more indirect way to load the accumulator is to specify an address and go out to a memory location and grab a byte from that location. See ABSOLUTE MODE. Every time you do this there may be a different number in that location. LDA $0600 goes to the start of page 6 ($0600 or 1536) to find the byte to load into the accumulator. The most indirect way (and the big advantage Atari computers have over most competitors) is to use a register to POINT to the address of the real data you are concerned with. This is called the indirect mode. This is done in the COLOR registers and the character set. The CHBAS pointer in location 756 points to the location in memory that is to be the base of the character set data. By changing this pointer you can change the entire character set in an instant by pointing to a different set of data.



INDIRECT ADDRESSING — Using the program counter (PC) of the 6502 processor to point to the address of the address where the data is found.

INITIALIZE — Setting up all of the color registers, character set data, and subroutines needed to run a program. Initialization may also consist of writing zeros to every position in a string in order to "erase" the old data. If hundreds of bytes are changed at the start of a BASIC program, and the changes are done by POKEing the data, the INITIALIZATION process can require several minutes of time. This is usually dead time and one may wonder if the program has crashed.

**INKEY$** — Microsoft BASIC II command. INKEY$ returns the character corresponding to the last keystroke. Similar to a CHR$(PEEK(764)).

INPUT . . AT — Microsoft BASIC II command. This command is used to read a byte from a sector through a channel OPENed for input. It assigns the byte to a variable name. It can also read bytes from the screen.

INPUT (I.) — BASIC command. INPUT halts a program while it waits for a string or numerical variable to be assigned a value through the keyboard. You must press RETURN to enter the value. A question mark (?) prompt is generated by the INPUT command. In Microsoft BASIC II, the prompt can be supplied after the INPUT command. This makes for easier programming. Alternatives to INPUT for one byte entries are GET and PEEK(764) routines.

INPUT ALTERNATIVES — In Atari BASIC you can cause your program to stop and wait for input by using the INPUT command. INPUT will present you with the question mark (?) prompt whether you are asking for a numeric or string input. You must also press RETURN after entering the value. There are alternatives to INPUT. A very neat way is to use the GET command. GET waits for a single character. You must know the ATASCII value of the character you are looking for and also you must OPEN the keyboard for input. The BASIC subroutine is this:

```
10 ? "HIT 'C' TO CONTINUE!"
20 OPEN #1,4,0,"K:"
30 GET #1,A:REM ** PROGRAM WAITS HERE FO
R A KEY
40 IF A<>67 THEN 30:REM ** LOOP UNTIL 'C
' IS HIT
50 CLOSE #1:REM ** DON'T FORGET TO CLOSE
```

Another method is to PEEK at location 764 to find out the last key struck on the keyboard. This technique is very simple but again it requires you to know the keycode for each key. The keycode is not the same as the ATASCII value. See the KEYCODE table for the keycode to ATASCII conversions.

```
10 ? "HIT 'C' TO CONTINUE!"
20 IF PEEK(764)<>18 THEN 20:REM CYCLE HE
RE UNTIL 'C' IS HIT
```

Another form of INPUT is to use an INPUT through an IOCB. This will not give the question mark (?) prompt nor will it print the variable you type on the screen, unless you want it to. Again, you must OPEN the IOCB and you must also DIM the variable you decide upon.

```
5 DIM X$(30)
10 ? "TYPE A LINE, PRESS 'RETURN'"
20 OPEN #1,4,0,"K:":REM ** OPEN IOCB 1 F
OR THE KEYBOARD READ
30 INPUT #1,X$:REM ** WAIT HERE FOR X$ T
O BE ASSIGNED
40 CLOSE #1
50 PRINT X$
```

INPUT LINE BUFFER — The INPUT LINE BUFFER is used by BASIC as the place where it goes to get data to tokenize for interpretation. The buffer is the 128 bytes between 1408 and 1535 ($580-$5FF). The data comes into this buffer through the CIO from a handler device.

INSTR — This is a Microsoft BASIC II command derived from the term INSTRing. INSTR uses three arguments in the format: INSTR(n,X$,Y$), where n is a number (integer), X$ is a short string, and Y$ is a longer string. INSTR searches the longer string (Y$) for the shorter string (X$) starting from the nth position in the string. The value of the starting position of the short string is returned.

INSTRUCTION SET — Every microprocessor has an INSTRUCTION SET to tell it what to do. The 6502 has 151 different opcodes (operation codes) in its instruction set. These are listed in the code conversion table. The ANTIC chip has only four opcodes in its instruction set. The opcodes control the flow of bytes in and out of the memory and registers of the 6502, as well as conversion or shifting of bytes in the processor registers.

INT — BASIC function. INT provides the greatest integer value that is less than the argument given. INT(123.456) becomes 123 and INT (–7.23) becomes –8.

INTERFACE MODULE — The 850 INTERFACE MODULE is a conversion device. The INTERFACE MODULE takes the high speed serial data from the computer and converts it to a parallel bit signal for printers and also a controllable speed serial bit signal for modems and other devices (RS 232). In order to use the 850 interface, you must have it turned on and connected WHILE you boot up your computer. Note that it must be on BEFORE the computer is turned on. The reason is that a handler program contained in ROM in the 850 is loaded into the Atari computer. You will hear a loud and steady tone letting you know it is successfully loaded. See the RS232 booter program listing in the RS232 section. This program starts the process of booting the RS232 handler through an AUTORUN.SYS program. This file must be on any disk which loads in a modem program, such as AMODEM.

INTERNAL CHARACTER SET — The Atari computer hardware does not use ATASCII characters for its internal operations. ATASCII characters are translated according to the following chart as data is input or output.

ATASCII CODE RANGE	INTERNAL CHARACTER SET RANGE
$00 to 1F	$40 to 5F
20 to 3F	00 to 1F
40 to 5F	20 to 3F
60 to 7F	60 to 75
80 to 9F	C0 to DF
A0 to BF	80 to 9F
C0 to DF	A0 to BF
E0 to FF	E0 to FF

You can make your own transformation table by taking the ranges of codes in the ATASCII range and transforming them to the INTERNAL CHARACTER SET range as listed above. When you use the WINDOW.BAS program in the MEMORY section, you are viewing the INTERNAL CHARACTER SET codes.

INTERNATIONAL CHARACTER SET — The XL series computers contain an additional set of characters to allow use of Atari computers with most western European languages. To enable the INTERNATIONAL SET, type POKE 756,204. To use the characters, you must use CTRL in combination with the key chart layout provided in the owners' guide. Note that these characters will NOT print out on your printer just because they appear on your TV screen. The printer uses its own character set.

The data for the INTERNATIONAL CHARACTER SET is stored in ROM beginning at location 52,224 ($CC00). Poking 756 with 204 sets the character set base pointer (CHBAS) to page $CC (204).

To print the INTERNATIONAL CHARACTER SET on the Atari 1025 and 1027 printers, send an ESCape CTRL-W (27 23 ATASCII) to the printer. International characters are represented by the 26 CTRL keys (RANGING form ATASCII 0 to 25). See the XL owners' manual for the keychart defining the characters. An ESCape CTRL-X turns off the international mode.

INTERPRETER — The Atari BASIC cartridge contains a program which is an interpreter. This program takes words and numbers you compose as a BASIC program and converts or interprets them into a tokenized code which is used as instructions for the 6502 microprocessor.

INTERRUPTS — An INTERRUPT is an event which causes a microprocessor to stop what it is doing and begin something else. When the other task is complete, the microprocessor can take up where it left off on the original task. INTERRUPTS can be used to make sound, move players, change colors, and change data. Tasks are programmed in machine language and can be executed very quickly during the INTERRUPT time. The vertical blanking time can be used for an INTERRUPT of up to 1400 microseconds in length. The vertical blanking occurs every 1/60th second and many exciting activities can be performed during this time. The vertical blank INTERRUPT (VBI) can be performed before the Operating System begins its updating of all of the registers, or it can be done afterwards. If it is done before, it is an immediate mode VBI, and can consist of a program up to 840 machine cycles in length. If it is done afterward, it is a deferred mode VBI and can consist of up to 1,470 machine cycles. See *De Re Atari* for an advanced treatment of INTERRUPTS.

INVERSE CHARACTERS — The Atari character set resides in ROM starting at address 57344 ($E000) and takes up 1,024 bytes. There are only 128 characters in the table, or eight bytes per character. You see the inverse character by pressing the Atari Key (or inverse key) which sets the highest bit (D7). This essentially adds 128 to the ATASCII value. *Note: This key is often hit accidentally which causes any program designed to accept alphanumeric characters to be disrupted.*

IRG — InterRecord Gap — A record in a cassette tape file consists of 132 bytes. There are 128 data bytes, two markers for speed, a control byte, and a checksum. Between records there is a gap called the InterRecord Gap. This gap can be short or long depending on the mode set when a file is opened. In the normal IRG mode the tape stops after each record to process the data. This stop is usually so short that the speed fluctuation is not observable. In the short IRG mode, the data is loaded straight into memory (RAM) without stopping. CSAVE and CLOAD use the short IRG mode.

IRQ — Interrupt ReQuest. Location 53774 is the Interrupt Request Enable register. Setting various bits in this register enable interrupts; for example, the BREAK key interrupt.

IOCB — Input/Output Control Block. IOCB's are sometimes referred to as channels. All input from and output to devices is done through the IOCB's which allows you to use the CIO (Central Input Output). The eight channels are defined 0 through 7 and are available through Atari BASIC in a format such as OPEN #X,4,0,"D:" where X is the channel number from 0 to 7. Channel 0 is permanently reserved for the E: device, the Editor, and channels 6 and 7 are used for some graphics outputs. Channels 1 through 5 are always available to use in BASIC programs. A channel must be OPENed for use and then CLOSEd by a CLOSE, END, or RUN statement.

An IOCB is a 16 byte long block of data. They start at $340 for #0 and repeat at $350, $360, $370, $380, $390, $3A0, and $3B0. The 16 bytes which hold the information to direct I/O are specified as follows. Note that the OPEN command from BASIC fills in the data for these locations.

NAME	OFFSET	CONTENTS
ICCOM	IOCB+2	Command
ICBAL	IOCB+4	Buffer address, low
ICBAH	IOCB+5	Buffer address, high
ICBLL	IOCB+8	Buffer length, low byte
ICBLH	IOCB+9	Buffer length, high byte
ICAX1	IOCB+10	Aux 1 byte (read/write)
ICAX2	IOCB+11	Aux 2 byte (GR. mode)

After all of the parameters are filled in, the call can be executed. The offset of the start of the IOCB ($00, $10, $20, etc.) which you are using must be put into the X register, and the CIOV ($E456) must be entered.

J

JIFFY — JIFFY is a measure of time equal to about 1/60th of a second. This time interval is derived from the timing of the vertical blanking interval for the CRT rewriting. The duration of the SOUND statement in Microsoft BASIC II is controlled in JIFFIES by the optional fifth parameter given after the SOUND statement.

JONESTERM — JONESTERM is a public domain modem program with many powerful and easy to use features. It was written by Frank Jones. JONESTERM (JTERM) is available through most bulletin boards as a download program, or you can type it in from *COMPUTE!*, January 1983, p. 202. JTERM has undergone extensive revision; version 35 is popular at this writing. Many users groups have enhanced JTERM to the point of it being of commercial quality. JTERM uses a series of menus to set up the terminal parameters. The console switches are also used in the following ways: OPTION is used to select full or half duplex; SELECT is used to turn on and off the MEMSTORE feature (this is like a recorder which dumps all transmissions into available memory.); START is used to exit the terminal mode and bring back the menu. The menu (of which there are many versions) usually asks whether you want to upload, download, or automatically dial. If you hit U for upload, you will need to supply a filespec to load in for uploading. If you hit D for download, you will be asked about translation and parity checking. Translation can be either Light or None. Light translation interprets some characters for more aesthetic viewing on the screen. No changes are made to stored or received characters. Parity checking is a method for verifying the integrity of each transmitted byte by adding up all of the bits. Most bulletin boards do not support parity checking, so none is needed.

JUSTIFICATION — JUSTIFICATION is the process of aligning text on a page. The left edge of text is normally justified or aligned. The right-hand side can be aligned with most word processing programs. This feature is implemented in software which looks at the determined length of the line and divides the number of spaces among the individual words. A printer must have proportional spacing capability if it is to print evenly spaced, right justified text. The display screen in GR.0 will not support proportional spacing, but it will support right hand JUSTIFICATION.

K

K — From KILO. One of the most commonly used buzzwords in computerdom is K. Common usage of the term K refers to a thousand. Engineers often refer to their annual salary in terms of K dollars. In computers, K is defined as 2 to the 10th power, or 1,024. This is 24 more than the typical interpretation of K, and this often results in confusion. 64K is actually 64 times 1024 (65,536), not 64,000.

KDOS — A powerful, feature-packed disk operating system for those who want to look deeper into the Atari architecture. KDOS resides in memory and takes up 14K. To be useful, some of the utilities require a 32K system. A machine language monitor allows you to Run, Load, Save, Go, Proceed, Examine, Alter, and Register. A map of records loaded can be displayed on the screen as a file is loading. A file can be loaded and not run while the monitor tells you what the INIT address is. Memory locations can be altered and the memory can be dumped back out to a disk file. This can help you cheat on those high scores if you can find the location of the score display. KBYTE.

KEY CLICK — In the XL series, you can turn off the KEY CLICK. This click is delivered through the TV speaker as opposed to through an internal speaker (as it is on the 400 and 800). To disable the KEY CLICK, POKE 731,255. To enable the KEY CLICK, POKE 731,Ø.

KEY REDEFINITION — You may REDEFINE the KEYboard of the XL series computers. There is a pointer to a table of characters which will be generated when you strike a key. By changing the pointer to a custom table which you generate, you can REDEFINE the KEYboard. The pointer to the table is in locations $79 and $7A (121 and 122, LO byte AND HI byte). The table is 192 bytes long with the lowest 64 ($40) bytes for the lowercase, the middle 64 bytes for the shifted (uppercase) and the top 64 bytes for the CTRL keys. Any keys between $80 and $91 (128 and 145) will be treated as a special editing key. The following table shows the structure of the three 64 byte blocks described above. These are hardware generated codes.

KEYBOARD CODES

	$0-F 1-15 X	$10-1F 16-31 1X	$20-2F 32-47 2X	$30-3F 48-63 3X
0	L	V	,	9
1	J	HELP	SPACE	
2	;	C	.	0
3	F1	F3	N	7
4	F2	F4		BACK S
5	K	B	M	8
6	+	X	/	<
7	*	Z	⅄	>
8	O	4	R	F
9				H

	\$0-F 1-15 X	\$10-1F 16-31 1X	\$20-2F 32-47 2X	\$30-3F 48-63 3X
A	P	3	E	D
B	U	6	Y	
C	RETURN	ESC	TAB	CAPS
D	I	5	T	G
E	-	2	W	S
F	=	1	Q	A

KEY REPEAT — In the XL series Operating System, you can control the KEY REPEAT DELAY and RATE by changing two OS variables. These options are not available on the 400 and 800 computers. To change the KEY REPEAT DELAY, POKE location 729 (\$02D9) with the number of jiffies (1/60 second intervals) you want the keyboard to wait before it starts repeating the typed key. The default value is 48 (48/60) or .8 seconds.

To change the KEY REPEAT RATE or the number of jiffies, the keyboard waits to repeat characters after the DELAY is passed (POKE 730 (\$02DA)) with the number of jiffies. The default value is 6 (.1) seconds between repeats.

KILL — KILL is used in Microsoft BASIC II to delete a file, usually one stored on a disk.

KLUDGE — This term (which rhymes with huge) is used to denote, in a somewhat cynical manner, an intricate assembly of hardware or software which has the appearance of being pieced together by a hobbyist. You would not normally buy a KLUDGE, but you might make one from spare parts in your spare time.

KOALA PAD — The KOALA PAD is an input device for the Atari and other brands of computers. It is a touch sensitive board that uses the paddle ports to generate values from 0 to 255 according to the x and y position of a stylus or finger pressing on the surface. Though not really useful for most games, it is a nice graphics tool. The software provided with the KOALA PAD is Micro Illustrator. Screens created with the Micro Illustrator can be saved to disk files and used for other activites. The files are stored in a non-standard format, but you can make the program write them in a standard format by pressing INSERT while the picture is on the screen. The image will be stored as a file called D:PICTURE. Any image file called PICTURE can be recalled by pressing CLEAR.

L

LABEL — A LABEL is used in an assembly language statement as a marker for a particular line number. The LABEL is the second field in a statement and it must be separated from the statement number and opcode mnemonic by one space. If no LABEL is used, two spaces must be placed between the number and mnemonic. Disassembled source code usually does not reconstruct the LABEL.

LADDER GENRE GAMES — During 1982 and 1983, a series of games using the ladder climbing scenario were introduced. Some of these games were: Miner 2049'er, Canyon Climber, Donkey Kong, Jumpman, Jumpman Jr., Mr. Robot, Hardhat Mack, and Mountain King.

LANGUAGES — Although BASIC is now built in to the Atari computers, you have your choice of many different computer languages. BASIC is fine for composing quick utilities and demonstrations, but assembly language is required for high performance games and financial programs. FORTH is available to give you total control over your computer. A list of languages available for Atari computers follows.

LANGUAGE	NAME	PUBLISHER
Action!	Action!	OSS, Inc.
Assembler	Atari Assembler/Ed.	Atari, Inc.
Assembler	Atari Macro Assembler	Atari, Inc.
Assembler	MAC/65	OSS, Inc.
Assembler	EASMD	OSS, Inc.
Assembler	ATAS (ed/assem)	Elcomp Publishing
Assembler	ATMAS (macro)	Elcomp Publishing
Assembler	Synassembler	Synapse
Assembler	MAE	Eastern House Software
Assembler	ASSM/TED	Eastern House Software
Assembler	Edit 6502	LJK Enterprises
Assembler	DATASM	Datasoft, Inc.
Assembler	Cassette Assembler	Quality
BASIC	Atari BASIC	Atari, Inc.
BASIC	BASIC A+	OSS, Inc.
BASIC	BASIC XL	OSS, Inc.
BASIC	Microsoft BASIC II	Atari, Inc.
BASIC	OSS BASIC 400	OSS, Inc.
BASIC	The BASIC Compiler	Datasoft, Inc.
C	C/65	OSS, Inc.
C	Deep Blue C	APX
C	Tiny C	OSS, Inc.
FORTH	FORTH	Elcomp Publishing
FORTH	Team FORTH	Public Domain
FORTH	PNS FORTH	Pink Noise Studios
FORTH	Extended fig-FORTH	APX

LANGUAGE	NAME	PUBLISHER
FORTH	valFORTH	ALPAR International
FORTH	ECS/MVP-FORTH	Mountain View Press
FORTH	QS FORTH	Quality Software
LISP	InterLISP	Datasoft, Inc.
LOGO	Atari LOGO	Atari, Inc.
Pascal	Atari Pascal	APX
PILOT	Atari PILOT	Atari, Inc.
WSFN	Which Stands For Nothing	APX

**LEFT$** — In Microsoft BASIC II, LEFT$(X$,n) returns n characters from the left side of the string X$.

LEFT-HANDED JOYSTICK — You can convert an ordinary Atari joystick to a lefty model by merely unscrewing the base and transposing a few connectors. The button will be on the top right side when you are finished and all of the direction labels on the front should be changed for consistency. The top will become the right side. When you take the bottom off the case, you will see a column of colored connectors. Use the chart below to transpose the wires and put your lefty model back together.

Right	Left
brown	blue
white	brown
black	black
blue	green
green	white
orange	orange

LEGAL — A LEGAL operation is one that will not generate an error during interpretation by the BASIC interpreter.

LEN — In BASIC, LEN returns the value of the number of characters in a string. The format is LEN(STRING$).

LET — This BASIC command is usually superfluous. It is used for the assignment of values to variables, as in LET X=1. The same assignment can be done by X=1.

LETTER PERFECT — LETTER PERFECT is a high powered word processor for use on Atari computers. This entire book was written on the LETTER PERFECT text editor. LETTER PERFECT uses its own version of DOS and is not compatible with Atari DOS 2.0S. There is a special utility program for conversion of files between the two. The directory begins in sector 362 in LJK DOS, as opposed to 361 in Atari DOS 2.0S. Version 3.X of LETTER PERFECT offers substantial improvements over Versions 1 and 2. Version 1 actually had several fatal bugs which are entirely remedied in Version 3. There is an 80 column version of 3.X on the flip side of the program disk, and this allows 80 character by 24 line editing with the Bit 3 80-column board and a monitor. The program is also available on a ROM cartridge for a premium. File merging into letters is possible with LETTER PERFECT.

Database files prepared with LJK's Data Perfect database manager, or lines entered manually with LETTER PERFECT can be merged with form letters. A separate printer editor allows easy custom modification of printer codes so that any printer can be used with LETTER PERFECT. This feature makes it the most versatile text editor on the market as far as printer interfacing is concerned. The printer codes are stored in sector 1 and can be copied to any LJK formatted disk with a sector copier such as SCOPY. A large variety of control characters are used for word processing commands. A system of crude mnemonics is used to help memorization. Control-E is used to move the cursor to the end to the text file in memory. Format lines are used in the body of the text to control printing of the text. A Control-F followed by w60 will set the WIDTH of the printed page to 60 columns. See LJK DOS for a program to list the directory of LJK disks. LJK Enterprises.

LIGHT PEN — A LIGHT PEN is a peripheral device that can tell the computer where the user is pointing on the CRT screen. Since it is usually shaped like a pen or pencil, you can literally point to images on the screen, push a button, and make changes, provided you have the proper software. The Atari OS has two registers designed for a light pen through the joystick ports. These are LPENH for the horizontal position value in location 564 ($234) which is a shadow from location 54284 ($D40C). The horizontal value is determined by the triggering on of the phototransistor in the LIGHT PEN which in turn puts the horizontal color clock count into location 564. This ranges over 228 possible values. You can use a PEEK(564) to identify the actual numbers for your TV. The vertical position register is in location 565 ($235) which is a shadow for 54285 ($D40D). Only 96 vertical lines are resolvable. The resolution of a LIGHT PEN is often degraded by extraneous light which enters the lens on the phototransistor. Careful design will eliminate much of this optical noise.

LINE INPUT — In Microsoft BASIC II, LINE INPUT reads an entire line from the keyboard into a string variable, up to the carriage return. This differs from INPUT in Microsoft BASIC II, which ignores any data after a comma or semicolon.

LINE INPUT..AT — In Microsoft BASIC II, LINE INPUT..AT is used to enter a line of data from a disk providing that the device has been OPENed for LINE INPUT.

LINE — There are two types of LINEs in BASIC. A physical LINE is the 38 or 40 characters you see across the screen in normal default mode. The 40 character LINE buffer for a LINE is located between 583 and 662 ($247 and $26E). A logical LINE consists of up to 120 characters. A logical LINE is what follows a line number and is terminated by a RETURN (EOL or ATASCII 155). The BASIC LINE must fit into the 128 byte input buffer and also in the 256 byte BASIC output buffer after tokenization.

LINE FEED — LINE FEED is the scrolling of the screen data or paper in the printer by one character line. A LINE FEED is executed by sending an ATASCII character number 10 (CTRL-J) to the handler device. A RETURN is a combination carriage return and LINE FEED.

LINE NUMBERS — BASIC programs use line numbers to order the processing of lines and statements. Lines can be entered in any order but they will be numerically

arranged after they are tokenized for interpretation. A renumber utility may be used to spread out lines logically if there is no room left to insert new lines.

LINE MAKER — You can generate DATA statements automatically by writing the statement on the screen and letting the program PRESS RETURN. The following program will do this for you. LIST the line numbers you want to save to a file by using the format LIST"D:TEMP.LST",1,100. This example saves the first 100 DATA statements to a temporary file.

```
20000 REM ** ABCS OF ATARI COMPUTERS
20010 REM ** PROGRAM: DATAGEN.BAS
20020 REM ** GENERATES DATA STATEMENTS
20030 REM ** AUTOMATICALLY.  YOU JUST
20040 REM ** TYPE IN NUMBERS AND COMMA
S.
20050 REM **
20060 CLR :POKE 82,2:POKE 83,39:POKE 7
52,0:? CHR$(125):RESTORE 20180
20070 FOR A=1536 TO 1589:READ B:POKE A
,B:NEXT A
20080 ? CHR$(125):? :? "ENTER FIRST LI
NE #, INCREMENT":INPUT B,A
20090 B=INT(B):A=INT(A):IF B<1 OR B>29
999 OR A<1 OR A>9999 THEN 20080
20100 CLOSE #1:POKE 702,0:OPEN #1,4,0,
"K"
20110 ? CHR$(125):? :? :? B;" DATA ";
20120 GET #1,C:IF C=155 THEN 20150
20130 IF C=127 THEN POKE 702,64:GOTO 2
0080
20140 D=USR(1536,C):? CHR$(D);:GOTO 20
120
20150 ? :? "CONT":POSITION 2,0:POKE 84
2,13:STOP
20160 POKE 842,12:B=B+A:IF B<30000 THE
N 20110
20170 CLOSE #1:STOP
20180 DATA 162,13,104,104,104,221,26,6
,240,10,202,16,248,133,212,169,0,133,2
13,96,189,40,6,76,13,6
20190 DATA 32,109,106,107,108,117,105,
111,98,110,103,104,116,121,48,48,49,50
,51,52,53,54,65,66,67,68,69,70
20200 CLOSE #1:? CHR$(125):? :? :FOR C
=31000 TO 31013:? C:NEXT C:? 32000:? 3
2001
20210 ? "CLR:POKE 842,12:?CHR$(125)":P
OSITION 2,0:POKE 842,13:STOP
```

LINK — Disk files created under Atari DOS 2.0S and OS/A+ 2.X are linked files. At the end of each sector in the file are three bytes which are used for control information as to where to find the next sector in the file, the number of bytes in this particular sector, and the file number of this file according to the directory. The link is found in two bytes of any sector, byte 126 and part of byte 125. There are 720 numbered sectors on the disk so 10 bits are required to represent the sector number (eight in byte 126 and two in byte 125). Nine bits would allow only 512 sectors to be addressed (2 to the 9th power). Ten bits allow up to 1,024 sectors to be addressed (2 to the 10th power). Since each byte is composed of eight bits, we must use two bits from another byte to get a total of 10 bits. Of the 128 bytes per sector (256 in double density), three are used for housekeeping; these are 125, 126, and 127. Bytes 125 and 126, located at the end of every DOS file sector, are used for the link.

BYTE 125

BIT #							
7	6	5	4	3	2	1	0
FILE NUMBER (0 to 64)						MSB	

BYTE 126

BIT #							
7	6	5	4	3	2	1	0
NEXT SECTOR - LSB							

The next sector pointer is broken into two parts residing in two separate bytes. The LSB (Least Significant Byte) is byte 126. This is a number between 0 and 255. An ATASCII character representing the number will be found in this location when you use a sector viewing utility such as Diskey or Diskscan. The MSB (Most Significant Byte) is found in the two rightmost bits of byte 125. These two bits can represent the numbers 0, 1, 2, or 3 by holding a 00, 01, 10, or 11. The number in these bits is multiplied by 256 resulting in 0, 256, 512, or 768. This number is added to the LSB number in byte 126 to get the next sector. Actually, there is no need for a 11 or 768 in the MSB because the largest sector number is 720. Byte 125 still contains six other bits which are used for a file number.

LIST (L.) — In BASIC, LIST is the command to print the program from memory to the screen in the order of line numbers. Programs can be LISTed to other devices such as the printer by using the format LIST"P:". A range of line numbers can be LISTed by typing LIST"P:",1,1ØØ. The range of line numbers can be saved to another file by using LIST"D:TEST",1,1ØØ. The LISTed file is not saved as a tokenized file, but rather, as a text file. The variable table is NOT saved when you LIST a program so this is a good way to discard unused variable names.

LJK DOS — Letter Perfect application software uses a DOS which is not compatible with Atari DOS 2.0S. The disk layout is different which means that a conversion must be done to interchange LJK files with Atari compatible files (Text Wizard, Atari Writer, LISTed BASIC files, etc.). LJK DOS uses 128 bytes per sector with no sector links. Sector 1 is used for the printer set up information. You can set this file up on one disk and make multiple copies to other disks by copying only sector 1. Sectors 8 to 55 are ignored on a disk. The VTOC on an LJK file disk is on sector 362 as opposed to 360 on an Atari DOS disk. The directory is located between sectors 363 and 371. LJK has a conversion program file utility which allows you to load files from either Atari DOS disks or LJK DOS disks and translate them into the other type.

The following program will print out a listing of the directory of an LJK DOS disk.

```
100 REM ** ABCS OF ATARI COMPUTERS
110 REM ** BY DAVE MENTLEY * JAN.1984
120 REM ** PROGRAM * LJKDIR.BAS
130 REM
140 DIM SECT$(44),A$(128),DIR$(512),Q$
(1)
150 TRAP 200
160 FOR B=1 TO 44
170 READ C
180 SECT$(B,B)=CHR$(C)
190 NEXT B
200 ? CHR$(125):? "LETTER PERFECT DIRE
CTORY LISTER"
210 ? :? "INSERT DISK AND PRESS <RETUR
N>";:INPUT Q$
220 W=0:DIR$(1)=""
230 FOR SECT=363 TO 368
240 REM
250 FOR D=1 TO 128:A$(D,D)=" ":NEXT D
260 REM ++ READ EXAMPLE ++
270 X=USR(ADR(SECT$),ADR(A$),SECT,0)
280 REM
290 DIR$(1+W)=A$(6,13):DIR$(9+W)=A$(22
,29):DIR$(17+W)=A$(38,45):DIR$(25+W)=A
$(54,61):DIR$(33+W)=A$(70,77)
300 DIR$(41+W)=A$(86,93):DIR$(49+W)=A$
(102,109):DIR$(57+W)=A$(118,125)
310 REM **
320 REM ** PRINT
330 REM **
335 TRAP 395
340 W=W+64:NEXT SECT
350 LPRINT "      LETTER PERFECT FILE D
ISK":LPRINT
360 FOR I=0 TO 511 STEP 32
370 LPRINT DIR$(1+I,8+I),DIR$(9+I,16+I
),DIR$(17+I,24+I),DIR$(25+I,32+I)
380 IF DIR$(26+I,28+I)="   " THEN 200
390 NEXT I
395 RUN
410 DATA 104,104,141,5,3,104,141,4,3,1
04
420 DATA 141,11,3,104,141,10,3,104,104
,201
430 DATA 1,208,7,169,87,141,2,3,208,5
440 DATA 169,82,141,2,3,169,1,141,1,3
450 DATA 32,83,228,96
```

LOAD MEMORY SCAN — One of the important registers in the ANTIC chip is the "Memory Scan register." This register points to lists of characters or bit mapped graphics data in memory. This register is written only by the ANTIC's own DMA. Very simply, the LMS tells the ANTIC where in memory to go to find the data to write to the screen. The LMS instruction requires two more bytes to specify the memory starting point. In the display list, the LMS is enabled by setting the sixth bit of the LMS instruction. This is equivalent to adding 64 (decimal) to the instruction or specifying a $40 as the first nibble of the instruction byte. The second nibble is used to specify the graphics mode. For example, the portion of display list instructions listed below:

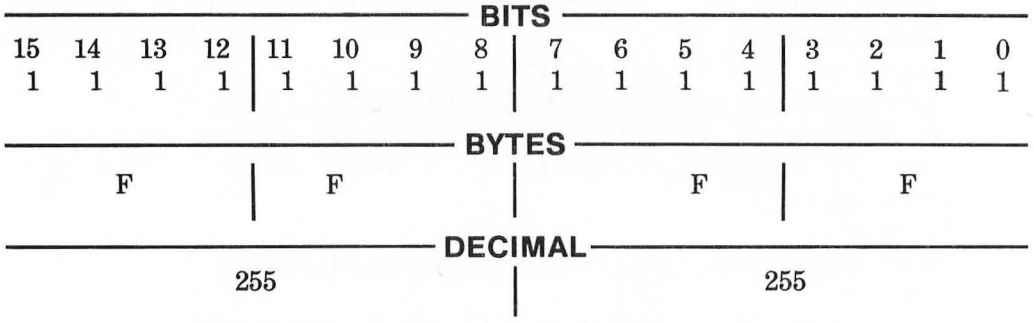
42, 00, 80 (HEX) or
66, 0, 128 (DECIMAL)

means LOAD MEMORY SCAN with the address that follows, and set up for ANTIC mode 2 (BASIC GR.0). Get screen data from memory location starting at $8000 (32768).

LOAD (LO.) — In BASIC, LOAD is the command used to transfer tokenized (SAVEd) BASIC programs into the computer's memory. Programs can be LOADed from the disk drive or cassette recorder. The format for using LOAD is LOAD"D:FILE" or LOAD"C:". The long IRG mode is used with cassette LOADs.

LO BYTE/HI BYTE — The LO BYTE/HI BYTE technique for specifying memory locations is sometimes confusing to the new user. The 6502 processes addresses in the LO/HI format but the natural way of looking at a number is HI/LO. To understand the difference, we must look at the way hexadecimal numbers are used to denote addresses in memory.

By using two 8 bit bytes for an address location we have a possible 65,536 different locations. This number is derived from 2 to the 16th power. Since each bit can be 0 or 1, and the address is composed of a series of 16 bits, 2 to the 16th is the number of different locations available. This number, 65,536, can also be represented by 256 x 256. A byte (eight bits) can represent from 0 to 255 numbers for a total of 256. If we use two bytes (16 bits), we can again address 65,536 different locations.



A simple example of the HI/LO format is as follows. Page 6 is the area of memory where BASIC will not usually erase or overwrite your data. Page 6 is the sixth block of 256 bytes from the bottom of memory. This area starts at location 1536 (6 x 256). In hex-

adecimal, this location is $0600. The two bytes comprising the address are 06 and 00. The first byte (06) is the HI BYTE and the second byte (00) is the LO BYTE. To get the decimal value, multiply the HI BYTE by 256 and then add the LO BYTE. For page numbers, the LO BYTE is always 00 or 0, so there is no addition. The confusion comes with the way the 6502 processes addresses. Instead of 06 00, the addresses are processed in the order 00 06, or LO BYTE/HI BYTE.

LOCATE — In Atari BASIC, LOCATE is the functional equivalent of a POSITION followed by a GET command. LOCATE positions the cursor at a specified x,y coordinate on the screen and assigns the value of the byte for that data to a specified variable. The format is LOCATE X,Y,var where X is the horizontal pixel location on the screen, Y is the vertical pixel location, and "var" is the variable name you assign. A number between 0 and 255 is returned in GR. 0, 1, and 2. Getting data from the screen in this way can change the data you are examining. Another way of getting data from the screen is to find the beginning of screen memory (just below PEEK(106)*256), calculate the position of the character position you want to get, and PEEK it.

LOCK — In Microsoft BASIC II, the LOCK command performs the same function as the Lock File command in Atari DOS. The LOCKed file cannot be deleted unless of course the disk were erased or formatted.

LOCKED FILE — The F option in Atari DOS is used to LOCK a file. The LOCK inhibits the DOS from deleting or rewriting data to the file. You cannot SAVE a file with the same name as a LOCKED FILE. In Microsoft BASIC II, the LOCK command will lock a file. Even a LOCKED FILE will be destroyed during the formatting of a disk. From Atari BASIC, the XIO command XIO 35,#1,0,0,"D:FILE" will LOCK the file called FILE.

LOG — In BASIC, LOG returns the natural logarithm of a positive number.

LOGICAL LINE — A LOGICAL LINE in BASIC consists of up to 128 characters which fill an input buffer used by the BASIC cartridge. Around three physical lines comprise a LOGICAL LINE. By using abbreviations (such as SE. for SETCOLOR), you can exceed three physical lines (120 characters) in a program and still safely construct a very long LOGICAL LINE.

LOMEM — The locations 128 and 129 ($80 and $81) comprise a pointer to an area of memory at the bottom of BASIC's memory and at the top of the OS RAM. This area is a token output buffer of 256 bytes in length. Without DOS booted, LOMEM contains 1792 ($0700). With DOS booted, the value is 7420 ($1CFC). Other buffers and handlers increase the value in LOMEM. You can lift up LOMEM and put your own programs here so that they will be safe from destruction. The RS232 device handler and the disk drive buffers raise LOMEM and take away from free memory.

LOMEM — In BASIC XL, LOMEM allows you to reserve memory below the normal BASIC program area. In most cases, it is more desirable to use the top of memory as a reserved area. The LOMEM command will wipe out any program in memory.

LOUDNESS — The volume of a sound generated by the SOUND statement is specified in the fourth argument in the statement. SOUND V,F,D,L is the format where L is the LOUDNESS value which ranges from 0 to 15. If the sum of all four VOLUME values for the four voices is above 32, distortion will result. The LOUDNESS is controlled by the low four bits of the AUDC1 through AUDC4 registers located from 53761 to 53767 ($D201 to $D207).

BIT NUMBER

7	6	5	4	3	2	1	0
DISTORTION			VOL	LOUDNESS			

LPRINT (LP.) — LPRINT is used to send results or variable values to the P: device (the printer). If no printer is attached and working when an LPRINT command is executed, an ERROR 138 will result. An LPRINT statement will clear the cassette buffer before the cassette recorder is used. Ignore the error which is generated.

When you use LPRINT, you send out a full 40 characters, even if only one character is being sent. The rest of the characters are blanks. In order to send or PUT just one character, you must OPEN an IOCB and PRINT the character to the channel. For example, OPEN #7,4,0,"P:":PRINT #7,X will send the value of X to the printer (P: device).

LSB — Least Significant Byte. Memory addresses need two bytes to include all available locations in the Atari system. Just as decimal numbers have more significant digits and less significant digits, hexadecimal addresses have more and less significant parts. In a decimal number such as 123, the 3 is the least significant digit. Increasing the 3 to a 4 only adds 1 to the value of the number. The 1 in the number 123 is the most significant digit. Increasing the 1 to a 2 adds 100 to the number (223).

Hexadecimal addresses are comprised of two bytes, a high byte and a low byte. The high byte is the most significant part. The low byte is the least significant part. Let us look at a number like $0600. This commonly seen in reference to page 6. The high byte or MSB is 06 and the low byte or LSB is 00. To convert to decimal we must multiply the high byte by 256 and add the result to the low byte. $0600 is 06 × 256 plus 00 or 1,536 + 0 = 1,536. Each byte can be expressed as a number from 0 to 255 by using hexadecimal digits 00 to FF. A confusing issue regarding hexadecimal addresses is that we look at them and read them "high byte, low byte," but most programs and assemblers call for the addresses by "low byte, high byte."

LSI — Large Scale Integration. This term refers to the number of devices that are built into a single silicon chip, such as a microprocessor. LSI refers to chips with between 100 and many thousands of logic gates per chip. Higher degrees of integration are referred to as Wafer Scale Integration (WSI) or Ultra Scale Integration (USI). Medium Scale Integration (MSI) and Small Scale Integration (SSI) were the precursors of LSI.

LUMINANCE — LUMINANCE or brightness of a playfield is determined by the four rightmost bits of the color registers, 708 - 712 ($2C4 - $2C8). Brightness, technically, is the way your eye and brain interpret LUMINANCE, however the terms are often used interchangeably. LUMINANCE is a part of color and the other part, the hue is determined by the four leftmost bits. LUMINANCE can be any even number between 0 and 14 (eight choices), with 14 being the brightest. The value of LUMINANCE can be added to the value for the hue multiplied by 16, and the result POKEd into the color register. The SETCOLOR command uses a value for LUMINANCE in its third argument (SETCOLOR R,H,L where R is the color register from 0 to 4, H is the hue from 0 to 15, and L is the LUMINANCE, an even number from 0 to 14.

COLOR REGISTERS

D7	D6	D5	D4	D3	D2	D1	D0
HUE				LUMINANCE			NA

LVAR — In BASIC XL, LVAR generates a list of all variables in a BASIC program. The line number on which the variable is used is also listed. The output can be sent to any device or filespec (P:, E:, D:filename). This is similar to the V function in the Monkey Wrench cartridge by Eastern House and also to the variable lister program in the V section of this book.

M

MACE — Michigan Atari Computer Enthusiasts. MACE is one of the largest Atari Computer users groups in the world. MACE publishes a newsletter/journal which is somewhat like a magazine. The MACE Journal is usually over 30 pages of critical hardware reviews, programs, software reviews, and Atari world news. The MACE library has many disks of public domain software available only to members (journal subscribers). The address is: Michigan Atari Computer Enthusiasts, P.O. Box 2785, Southfield, MI 48037.

MACHINE CODE — The 6502 processor in the Atari computer takes its instructions in machine code. Machine code is usually generated by an assembler, such as the Assembler/Editor cartridge, but you can program directly in machine code if you like. The instructions are comprised of an opcode (operations code) and an address. The opcodes work on the registers which are called the Accumulator, the X register, and the Y register. Programs simply load numbers into the registers, jump to certain locations in memory, change the value in the register, or compare two numbers. The following summary describes some of the basic opcodes and their actions.

OPCODE	FORM
20 LO HI	JSR oper -Jump to SubRoutine at location HI LO and start execution. Once at the subroutine at HI LO, occurrence of the opcode 60 will return to the main program. Uses three bytes and six machine cycles.

OPCODE	FORM
60	RTS - ReTurn from Subroutine described above. Uses one byte and six machine cycles.
CA	DEX - Decrement the value in the X register by 1. Uses one byte and two machine cycles.
E8	INX - Increment the value that is in the X register by 1. Uses one byte and two machine cycles.
various	LDA - LoaD the Accumulator with memory. The number which is to be loaded into the Accumulator can be an immediate number or it can come from a memory location. If LDA is used from memory, it can be from the Zero Page, absolute locations, or from a location indicated by pointers. Uses two or three bytes and two to six machine cycles.
various	LDX - LoaD the X register with the memory. All numbers loaded into the registers must be in hexadecimal. Works like LDA, but without the indirect modes. Uses two or three bytes and two to four machine cycles.
4C LO HI	JMP - JuMP to new location HI LO and start program execution there. There must be a workable program located at HI LO for this JuMP to work. JMP can also be used in the indirect mode where the location is pointed at. This mode must be used for relocatable code. Uses three bytes and three machine cycles (five in indirect mode).
various LO HI	STA - STore the contents of Accumulator into memory location HI LO. HI LO can be any writable location in the computer in zero page, absolute, or indirect mode. Uses two or three bytes and three to six machine cycles.
AA	TAX - Transfer the Accumulator to the X register. Uses one byte and two machine cycles.
8A	TXA - Transfer the value which is in the X register to the Accumulator. Uses one byte and two machine cycles.
various	CPX - ComPare the value of the number in the X register with memory. Memory can be an immediate number, a zero page byte, or an absolute memory location. The result of the comparison will be that the value is either equal or not equal to the compared number.
D0 YY	BNE - Look at the previous test (CPX). Branch back YY locations if the results were not equal. Uses two bytes and two machine cycles. If branch is to the same page, add one extra machine cycle. If branch is to another page, add two machine cycles.

MACHINE CYCLES — The 6502 processor in the Atari computer executes its instructions one step at a time. The processor is timed by a clock which pulses 1.79 million cycles (two color clocks) per second. Depending on the type of addressing required, each instruction requires several cycles of the processor to complete. Two cycles are used if no branch is taken (if the instruction is in the load IMMEDIATE mode). Three cycles are used if a branch is taken to fetch or store data in the ZERO page. Four cycles are used if the instuction is an ABSOLUTE mode requiring a two byte address and use of the X or Y register. Five cycles or more are used for an INDIRECT mode instruction in which an address for read/write is found in another address. The steps occur as instructions in the microprocessor are executed. See MACHINE CODE. There are less than 28,558 machine cycles per frame displayed on the screen. At least another 1,130 cycles are used for DMA refresh (getting the data called for by the display list, getting player or missile data, etc.) This gives a total of 29,868 cycles per frame. In GR.0, 8,672 cycles are needed to fetch the screen data to put the character data up.

MAPPING THE ATARI by Ian Chadwick. This memory map and guide to the 400 and 800 computers is a very complete and accurate source for the curious programmer. Nearly every memory location is described, and many applications are provided to experiment with changes in the system. COMPUTE! Books.

MARK — In modem telecommunications, the term MARK is used to designate a low signal. The low (as opposed to high) is interpreted as a logical "1". The lows and highs are derived from the shifting of frequencies by the modem in response to data coming from the computer.

MASK — A MASK is a technique for inputting data into a program in which you are only allowed to type in certain fields on the screen or in which you can only type certain characters, such as numbers, etc. MASKing helps make a better user interface between the program and the operator. You can build a screen MASK by using POSITION and INPUT statements.

MATRIX — A matrix is a multidimensional array of numbers. Atari BASIC supports only a two dimensional matrix. Variables in a two dimensional matrix are arranged logically in rows and columns. The value of the element stored at each row and column intersection is identified by a double subscript, such as ITEM(A,B). The following is an example of a hypothetical matrix of numbers:

		MACHINE NUMBER						
		1	2	3	4	5	6	7
DOLLARS	1	23	34	34	45	34	34	55
	2	33	23	44	34	23	43	44
	3	33	54	43	43	33	22	43
	4	11	22	32	33	43	11	21
	5	54	43	32	23	11	23	44

5 X 7 MATRIX

MAZE GAMES — One of the most popular fads to hit the home computer market was the maze game. Pac-man, Serpentine, Jawbreaker, Taxman, Ms. Pacman, and Ghost Hunter are a few examples of these games in which the theme is to chase around a maze while "eating" dots.

MEDIA — In computer terms, MEDIA usually refers to magnetic storage MEDIA. These are floppy disks, cassette tapes, or stringy floppies. The MEDIA are plastic films coated with a film of iron oxide particles which can be magnetized, thus enabling them to store data. The head of the storage device (disk drive or cassette deck) reads and writes the data as the MEDIA is moved past the head. Magnetic MEDIA will retain data for many years if they are not subjected to intense magnetic fields, such as from a speaker magnet or motor, but they will eventually wear down from abrasion and thus should be backed up.

MEGAHERTZ — One MEGAHERTZ is a unit of measurement for frequency equal to one million cycles per second. A signal, such as the clock signal to the microprocessor or the signal to the television, is made to vary at several MEGAHERTZ.

MEM.SAV — The MEM.SAV (pronounced mem-save or mem-dot-save) file is a file name reserved for disk files which store the contents of the Atari program memory when the DOS utility or DUP.SYS package is called in. The MEM.SAV file must be on the disk (listed in the directory) in order to use this utility. The N option in DOS 2.0 creates MEM.SAV on the disk. The data in the MEM.SAV file will be transferred back into computer memory when you leave the DUP menu. This is a time consuming process.

MEMO PAD MODE — The Atari 400 and 800 computers have a MEMO PAD MODE which allows you to type on the keyboard and leave messages on the screen. No other input or output is allowed. This feature is useful for testing the editor or for a demonstration for small children. The screendump utility called Printwiz (from Allen Macroware) makes use of the MEMO PAD MODE by allowing screens to be built-up without the ubiquitous ERROR messages from BASIC, and then permitting a dot for dot dump of the screen. Typing BYE (while in BASIC) gets you into the MEMO PAD MODE. The XL series uses BYE to enter the diagnostics mode, and does not have the MEMO PAD MODE.

MEMORY — MEMORY, or RAM, is the part of the computer which stores programs and allows them to RUN. Much of the computer's memory is used for things such as procedures for what to do when a key is pressed, how to add numbers, how to take the square root, etc. Other parts are used for handling disk drives and modems. The interesting part is the FREE RAM which varies with the amount of memory chips you have installed. If you have 48K installed, you will get a chunk of memory about 37K bytes long into which you can load and run your programs. Memory is assigned by two-byte values from 0000 to FFFF (hex) or 0 to 65,535 (decimal). The lower and upper memory locations remain the same for all computers but some parts in the middle differ depending upon: how much RAM you have installed, which graphics mode you are in, whether or not DOS is loaded in, or whether or not the RS-232 handler is booted in. See a memory map to get an idea of the structure of the Atari memory.

Memory is organized into "pages" which are blocks of 256 bytes each. There are 256 pages of 256 bytes. The page number is the first byte of the address. For example, the location $08 FF would be located in page 8. Location $9F 33 would be located in page 9F. Page 6 which is located between locations $06 00 and $07 00 (1536 and 1792 decimal) is one fairly protected area where the user can place strings and short machine language routines. The following program is designed to allow you to PEEK at the contents of memory. Use the joystick to page through. The trigger resets you back to page 0. Moving right or left moves one byte at a time. The keyboard can also be used to move through memory in this program. Use the period (.) to reset to page 0. The less than (<) and greater than (>) keys move through the pages and the hyphen (-) and equal sign (=) keys move a byte at a time.

```
10 REM ********************************
20 REM ***    MEMORY WINDOW          ***
30 REM ***    ABCS  OF  ATARI        ***
40 REM ***       COMPUTERS           ***
50 REM ***  PROGRAM: WINDOW.BAS      ***
60 REM ********************************
70 GRAPHICS 0:TRAP 270:POKE 16,64:POKE
 53774,64:POKE 82,0:POKE 83,30
80 POKE 752,1:POKE 559,33
90 FOR A=1536 TO 1563:READ B:POKE A,B:
NEXT A
100 DLST5=1540:DLST6=1541
110 SCRLO=PEEK(88):SCRHI=PEEK(89)
120 POKE DLST5,SCRLO:POKE DLST6,SCRHI
130 POKE 1552,SCRLO:POKE 1553,SCRHI
140 MEM=SCRHI*256+SCRLO
150 POKE 560,0:POKE 561,6
160 ? CHR$(125):? "          current mem
ory=":? MEM;" TO ";MEM+256
170 ? "          PAGE= ";INT(MEM/256)
180 IF STRIG(0)=0 OR PEEK(764)=34 THEN
 MEM=0:POKE 764,0
190 IF STICK(0)=14 OR PEEK(764)=55 THE
N MEM=MEM+256:POKE 764,0
200 IF STICK(0)=13 OR PEEK(764)=54 THE
N MEM=MEM-256:POKE 764,0
210 IF STICK(0)=7 OR PEEK(764)=14 THEN
 MEM=MEM+1:POKE 764,0
220 IF STICK(0)=11 OR PEEK(764)=15 THE
N MEM=MEM-1:POKE 764,0
230 IF MEM<0 OR MEM>65280 THEN MEM=652
80
240 MEMHI=INT(MEM/256):MEMLO=MEM-256*M
EMHI
250 POKE DLST5,MEMLO:POKE DLST6,MEMHI
260 GOTO 160
```

```
270 RUN
280 DATA 112,112,112,66,0,0,2,2,2,2,2,
2,2,0,0,71,0,0,0,2,7,7,7,2,6,65,0,6
```

MEMORY MAP— A MEMORY MAP is a listing of important memory location and their functions. Atari memory is laid out in a linear format with 65,536 locations numbered from 0 to 65,535. Some of the locations are not used, especially if you have less than 48K of RAM installed in your computer. The locations are grouped into certain blocks for very specific functions. Each location within the computer is important. Atari, Inc. has not published a comprehensive memory map, but several maps are available from outside sources. The best by far is a book called MAPPING THE ATARI, by Ian Chadwick (published by Compute! Books). You must have a good MEMORY MAP in order to write machine language programs. The BASIC language relieves you of most of the trouble of understanding the memory layout but it also prohibits you from taking advantage of some of the excellent features of the Atari computer.

MEMORY TESTER— A MEMORY TESTER is a program which writes some data into each available bit of free memory and reads and checks it to make sure that it will hold the correct data. While semiconductors such as the RAM chips in personal computers are very rarely known to fail if they pass their first burn-in, it is a good idea to test your memory if you get persistent program failures. The following program is a simple MEMORY TESTER you can type in to check your free memory. This program runs very slowly, so let it run while you go and do something else.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** PROGRAM: MEMTEST.BAS
30 REM ** FROM PORTLAND ATARI CLUB.
40 REM
50 DIM HEX$(2):APPMHI=256*PEEK(15)+PEE
K(14)+32:SDLSTL=256*PEEK(561)+PEEK(560
)
60 ? CHR$(125);"MEMORY TEST":?
70 ? "YOU CAN TEST BETWEEN LOCATIONS "
:? APPMHI;" AND ";SDLSTL:?
80 ? "ENTER LOW ADDRESS, HIGH ADDRESS"
:? :INPUT LO,HI:SEED=-85
90 IF (LO<APPMHI) OR (HI>SDLSTL) THEN
?
100 FOR TEST=1 TO 4:SEED=SEED+85:A=SEE
D:GOSUB 210:? :? "WRITE ALL $";HEX$
110 FOR ADDR=LO TO HI:POKE ADDR,SEED:N
EXT ADDR:? "READ AND VERIFY . . .";
120 FOR ADDR=LO TO HI:A=PEEK(ADDR):IF
A<>SEED THEN GOSUB 210:? ADDR,HEX$;CHR
$(253)
130 NEXT ADDR:? " OK":NEXT TEST
140 ? :? " WRITE $55 AND $AA..."
```

```
150 FOR ADDR=LO TO HI STEP 2:POKE ADDR
,85:POKE ADDR+1,170:NEXT ADDR:? "WAIT.
..";:FOR I=1 TO 10000:NEXT I
160 FOR ADDR=LO TO HI STEP 2:A=PEEK(AD
DR):IF A<>85 THEN GOSUB 210:? ADDR,85,
HEX$;CHR$(253)
170 A=PEEK(ADDR+1):IF A<>170 THEN GOSU
B 210:? ADDR+1,170,HEX$;CHR$(253)
180 NEXT ADDR:? "MEMORY OK":? CHR$(253
);:FOR I=1 TO 200:NEXT I:? CHR$(253)
190 ? "MORE";:INPUT HEX$:IF HEX$="Y" T
HEN 60
200 END
210 B=INT(A/16)+48:IF B>57 THEN B=B+7
220 C=A-INT(A/16)*16+48:IF C>57 THEN C
=C+7
230 HEX$(1,1)=CHR$(B):HEX$(2,2)=CHR$(C
):RETURN
```

MEMTOP — MEMTOP is the name of the memory pointer which holds the value of the top of the BASIC memory area. MEMTOP is located at $90 and $91 (144 and 145 decimal). Try PRINT PEEK[144] + PEEK[145]*256 to find the value of MEMTOP after you have loaded in a program.

MENU — A MENU is a program which gives a user several choices of action, each chosen by either a number, a letter, or a small number of keypresses.The following MENU is a very fast running program which you can use for your Atari DOS disks. It will load and run a program from disk if it is a BASIC SAVEd file. Use the AUTORUN.SYS maker program to make an auto-booting loader for the MENU by calling this file MENU and typing D:MENU as the file to run when prompted.

```
200 REM ** PROGRAM NAME: MENUFAST
210 REM ** ABCS OF ATARI COMPUTERS
220 REM ** PUBLIC DOMAIN PROGRAM
230 REM
240 GRAPHICS 0:OPEN #2,4,0,"K"
250 DIM FILENAME$(23*17),FILE$(17),F$(
20)
260 OPEN #1,6,0,"D:*.*"
270 TRAP 440
280 FOR X=1 TO 64
290 INPUT #1,FILE$
300 IF FILE$(5,16)="FREE SECTORS" THEN
 330
310 ? CHR$(64+X);"  ";FILE$:FILENAME$(
(X-1)*16+1,(X-1)*16+16)=FILE$
320 NEXT X
330 ? :? "TYPE LETTER"
```

```
340 GET #2,A:A=A-64
350 FILE$=FILENAME$((A-1)*16+3,(A-1)*1
6+13)
360 F$="D:"
370 FOR X=1 TO 8
380 IF FILE$(X,X)=" " THEN 410
390 F$(LEN(F$)+1)=FILE$(X,X)
400 NEXT X
410 F$(LEN(F$)+1)="."
420 F$(LEN(F$)+1)=FILE$(9,11)
430 RUN F$
440 END
```

MERGE — In Microsoft BASIC, MERGE is the equivalent of ENTER in Atari BASIC. Only LISTed programs can be called from disk or tape to be MERGEd with the program in memory. Existing line numbers will be overwritten. New line numbers will be placed in order in the new program listing.

MERGE (MAIL MERGE) — MAIL MERGE is a term borrowed from MicroPro, creator of Wordstar, the top selling word processing program for CP/M computers. MAIL MERGE refers to the process of taking a mailing list of names and addresses and creating a series of form letters with customized addressees from the list. LJK's Letter Perfect and Data Perfect are capable of doing this on the Atari computer.

MERGE FILES — In BASIC, files can be merged by LOADing, ENTERing, or typing in one program, and then ENTERing another BASIC program on top of it. Duplicate line numbers will be replaced and new one will be ordered. In LJK'S Letter Perfect, the MERGE function actually appends a file to the one in memory. It is not a true MERGE utility.

MICRODOS — Short (two sector) utility which resides in page 6 and allows many functions of the DUP.SYS without loading in DUP.SYS. It was originally published in the July 1982 edition of *Compute!*. DOS is modified so that DOSVEC points to page 6 instead of its normal location. Anytime you type DOS from BASIC, you will get the MicroDOS without loading a file from disk. Utilities included are Lock, Unlock, Delete, Rename, Format, Menu (Directory), DOS (real DOS), and BASIC.

MICROSOFT BASIC — The first implementation of BASIC on a microcomputer was done by William Gates of Microsoft, Inc. Microsoft has become a widespread standard on TRS-80s, Apples, and other popular micros. Atari has licensed a version of Microsoft for Atari Home Computers called Microsoft BASIC II. While there are some very useful features in Microsoft BASIC, there are some drawbacks. On the positive side, strings are handled very easily. Some of the DOS functions are implemented by keyword commands. Some utilities, such as renumbering, are implemented. On the negative side, the Microsoft BASIC II cartridge does not check for syntax errors when you press RETURN as does Atari BASIC; you have to wait until you RUN the program or else trace it for errors; the 16K cartridge is not large enough to handle all of the features necessary, so an additional disk is needed to load in some routines.

Microsoft BASIC will allow you to type in and run programs (with little modification) written for other computers. If the program uses extensive custom graphics screens, it will be a difficult job to make the modifications, but if the program is a financial or business program, there should be a minimal number of discrepancies.

**MID$** — In Microsoft BASIC, MID$ pulls out a substring from a larger string and assigns its value to a new variable. The original string, the first character position, and the number of characters desired must be specified. EXAMPLE: X$=MID$(Y$,4,6) will pull out the fourth through the ninth characters of string Y$ and assign them to a string called X$.

MISSILE — A missile is a two bit wide image in memory which can be controlled and made to collide with the playfield and players for animation, particularly for shoot 'em-up games. There are four missiles which can be combined to form a fifth player by setting bit 4 (adding 16) to memory location $26F (623). Missiles have their own position, collision, and color registers, just as the players do. The main difference is that instead of being eight bits wide as the players are, missiles are only two bits wide.

MISSILE — In BASIC XL, the MISSILE statement sets up the position and height of a missile. The format for the statement is MISSILE player#, exp1, exp2, where player# is the number of the parent player which is to shoot (1 to 4), exp1 is the vertical position from the top of the screen for the missile, and exp2, is the vertical height of the missile.

MODE LINE — A MODE LINE is a group of scan lines on the TV screen. Depending upon the graphics mode, a mode line varies in composition from 1 to 16 scan lines per mode line. The display list sets up the composition of the screen.

ANTIC MODE	BASIC MODE	Scan Lines/ Mode Line
2	0	8
3	–	10
4	–	8
5	–	16
6	1	8
7	2	16
8	3	8
9	4	4
A	5	4
B	6	2
C	–	1
D	7	2
E	–	1
F	8	1

MOIRE PATTERN — In high resolution graphics modes 7 and 8, a pattern of curved interference lines is generated from closely drawn lines on the screen. Sometimes extra colors from artifacting will be seen. This pattern is called a MOIRE PATTERN.

MONITOR — A television monitor is a CRT which displays the image generated by the computer for your eye to receive and your brain to interpret. A monitor is a television without the receiver, but it is typically able to accept much more information, more quickly than a TV. The 5 pin DIN plug on the Atari 800 allows you to connect the composite video signal to a color monitor (such as the Commodore 1702). The XL series also has the 5 pin DIN plug, but the chrominance signal is not connected, so your color monitor can only be used in a black and white mode.

MONITOR — A machine language MONITOR is a program which can display and modify memory locations. A MONITOR can often disassemble a program resident in memory, sometimes even while the program is running (or very recently shut off). The Monkey Wrench by Eastern House, Atmona by Elcomp, Omnimon by David Young, KDOS by KByte, Atari Assembler/Editor by Atari, Synassembler by Synapse, MAC 65 by OSS, and Action! by OSS are, or contain, MONITOR routines.

MOVE — In Microsoft BASIC, MOVE is used to transfer the contents of a block of memory to a new location in memory. The format to use MOVE is:

```
MOVE FIRSTADR,2NDADR,NUMBYTES
```

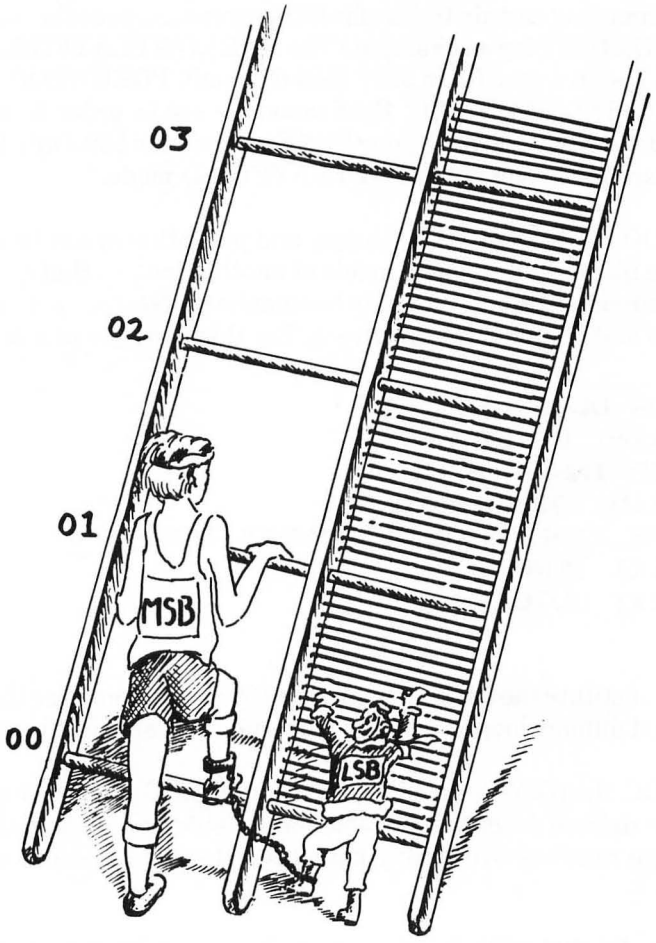
where FIRSTADR is the decimal address of the start of the block to be moved, 2NDADR is the address to which the block is to be MOVEd, and NUMBYTES is the number of bytes to be moved. This command is designed to move players vertically on the screen.

MOVE — In BASIC XL, the MOVE statement will move any number of bytes to any address at machine language speed. You could move important data into an area where it will be no good or you could mess up the Operating System memory and crash the system. The format for MOVE is:

```
MOVE STARTFROM, STARTTO, NUMBYTES
```

where startfrom is the starting address of the block you want to move, startto is the starting address of the place you want to move to, and numbytes is the length of the block to move. If numbytes is negative, the block will be transferred from the top down as opposed to from the bottom up.

MSB — Most Significant Byte — The MSB of an address is the first byte or the page number. This corresponds to the leftmost part of the address. For example, in the address $0700, the MSB is 07. In the address $FFAB, the MSB is FF. Many assemblers and machine language routines use the format LSB, MSB which may be confusing to the beginning programmer. This is intuitively opposite of the way we normally think of numbers. The notation LSB/MSB or LO/HI is often used to remind you of the order.



MUSIC PROGRAMS — One of the superior features of the Atari computer is its music making capabiltity. The first software to demonstrate this power was the Music Composer cartridge from Atari. It has complex editing commands and produces only square wave tones. This limits its attractiveness. The Advanced Music System from APX is truly more advanced. Its range is 5½ octaves, and the attack and decay of the notes can be adjusted. The Music Construction Set from Electronic Arts promises to be the easiest music system to use to date.

N

NAME . . . TO — In Microsoft BASIC, the NAME..TO function is the equivalent of the rename option in Atari DOS. It is available only on the extension disk.

NARROW PLAYFIELD — The NARROW PLAYFIELD is only 32 characters wide. It is useful for formatting certain types of utilities or games. (See the Memory Window program under MEMORY for an example.) The NARROW PLAYFIELD is enabled by setting bit 0 and 1 with a zero (0) in $22F (559 decimal). POKE 559,33 will enable the NARROW PLAYFIELD in BASIC. Bit 5 must be set in order to see the screen, therefore you must always add 32 to your POKEs to location 559. Only 128 color clocks are used on a scan line in the NARROW PLAYFIELD mode.

NESTING — DO loops, FOR-NEXT loops, and parentheses can be nested. NESTING refers to the placement of a loop inside of another loop so that completion of the inner loop is required for the outer loops to be completed. Nested loops slow down even simple programs and can be used for timers. Try this example of a nested loop:

```
10 FOR OUTER=1 TO 50
20 PRINT OUTER
30 FOR INNER=1 TO 50
40 PRINT OUTER+INNER
50 FOR TIME=1 TO 1000:NEXT TIME
60 NEXT INNER
70 NEXT OUTER
```

Lines 10 and 70 constitute the OUTER loop. Lines 30 and 60 comprise the INNER loop. Line 50 is a nested dummy loop which just burns up time, slowing down the program.

NEW — In BASIC, the NEW command deletes any BASIC program resident in RAM. All variables are deleted from the variable name table and all sounds are shut off. Machine language routines which may have been stored in memory are usually not affected.

NEXT (N.) — In BASIC, NEXT is the required second half of the FOR/NEXT loop. NEXT is used to increment the variable named in the FOR statement. For every FOR there must be a NEXT and vice versa. An ERROR 13 will be generated if there is no FOR to pair with every NEXT.

NMI — Non Maskable Interrupt. A Non Maskable Interrupt is an interrupt call which must be processed by the 6502. The only way the 6502 can ignore an NMI is if the non-maskable interrupt enable bit (NMIEN) is 0 (see NMIEN). In the Atari 800, the NMI pin on the 6502 processor is connected to the ANTIC chip and it can be pulled by the NMIEN bit on the ANTIC. ANTIC generates Vertical Blank and Display Instruction NMI's and the SYSTEM RESET button on the console feeds through ANTIC to this pin as well when a WARMSTART is forced. The Display List Interrupt and the Vertical Blank Interrupt are NMIs; that is, they must be executed when the 6502 encounters a call to the interrupt. See *De Re Atari* for a good explanation of interrupts.

NMIEN — Location $D40E (54286 decimal) is the Non-Maskable Interrupt Enable byte. Only bits 5, 6, and 7 are used in this byte. Setting bit 7 (adding 128) will enable the Display List Interrupt. Setting bit 6 (adding 64) enables the Vertical Blank Interrupt.

Setting bit 5 (adding 32) enables the RESET key interrupt. The DLI and VBI bits must be set to 1 after each reading for the interrupt to occur.

NORMAL PLAYFIELD — Location 559 ($22F) controls the width of the playfield. Setting bit D0 to zero (0) and bit D1 to one (1) will return to the normal 40 character playfield. The default value of location 559 is 34. That is 32 for enabling the DMA fetch and 2 for NORMAL PLAYFIELD. The sum is 34, which is the default value on power-up.

Try these POKEs:

POKE 559,33 for a Narrow Playfield
POKE 559,34 for a Normal Playfield
POKE 559,35 for a Wide Playfield

```
10 IF PEEK(53279)=3 THEN POKE 559,33
20 IF PEEK(53279)=7 THEN POKE 559,34
30 IF PEEK(53279)=6 THEN POKE 559,35
40 GOTO 10
```

Type in this short program and then type RUN. Press the START, SELECT, and OPTION console keys to see the different playfield widths. PEEK(53279) looks for a console key depression.

NOT — NOT is a Boolean operator in BASIC. Boolean algebra works only on 1s and 0s. NOT 0 will produce a 1. NOT (A=B) will produce a 0 if A=B (if it is true that A equals B). The same expression will produce a 1 if A is not equal to B. There are several bugs in Atari BASIC associated with NOT. PRINT A=NOT B will send your computer to Never-Never land.

NOTE (NO.) — In Atari BASIC, NOTE is used to retrieve the sector and byte numbers pointed to by a POINT statement. This information can be used to go to a specific location (sector) on a disk and retrieve specific bytes (byte count). NOTE works through a channel called an IOCB which must be OPENed for use.

NOTE — In Microsoft BASIC, NOTE is used to store the sector and byte values for the location of the file pointer into a variable named in the command. The NOTE routine must be loaded in from the extension disk. No POINT statement is used in conjunction.

NOTE AND POINT — The NOTE AND POINT scheme is a method of manipulating a pointer to a specific sector and byte on a disk. The position of the pointer is the place where the next byte or bytes will be read from or written to. POINT is used to position the pointer and NOTE is used to find out where it was positioned by the last POINT statement. NOTE AND POINT allows true random access to a disk. If the values for NOTE AND POINT are stored along with a keyword index, you can build a powerful data base manager.

NTSC — National Television Standards Committee. NTSC refers to a standard for sending data to a television receiver. It defines the frequencies and signal stucture of the data used to produce color images on a television. NTSC is used for all broadcasts in the United States, Canada, Mexico, the Western part of South America, Japan, and Korea. The Atari computers sold in the U.S. are NTSC versions. NTSC specifies 525 scan lines per screen and 30 frames per second. Western European Ataris are designed to work with the PAL (Phase Alternation by Line) standard and these standards use 625 lines per screen at 25 frames per second. The clock for the 6502 runs faster so that PAL systems' machine cycles are at a 2.217 MHz rate. France and Eastern Europe use a SECAM standard which is also different. In the XL series, location 98 ($62) is used as a flag to signal NTSC or PAL operation. In the 400 and 800, location 53268 ($D014) is the read only register to determine if is computer is set up for PAL or NTSC. IF bits 1, 2, and 3 are zero (0), then the computer is a PAL system. When these bits are one (1), the computer is an NTSC system.

MEMORY LOCATION 53268 ($D014)

D7	D6	D5	D4	D3	D2	D1	D0	BIT #
0	0	0	0	1	1	1	1	NTSC
0	0	0	0	0	0	0	1	PAL

Also check location $FFF8(65528) and $FFF9(65529) in Atari 400s and 800s. Compare against the following chart.

Value in $FFF8	Value in $FFF9	Revision and TV type
$DD (221)	$57 (87)	NTSC Revision A
$D6 (214)	$57 (87)	PAL Revision A
$F3 (243)	$E6 (230)	NTSC Revision B
$22 (34)	$58 (88)	PAL Revision B

The XL series uses a routine to check the type of GTIA (which adjusts the display list according to the type installed).

NULL STRING — A string variable with nothing assigned to it is a NULL STRING. It is possible to check for a NULL STRING by comparing it to an empty quote (""). Note that there is not even a space between the quotation marks. A NULL STRING results if no value has been assigned to a string variable; for instance, if no key has been hit during an operation.

NUM — In BASIC XL, the NUM statement calls the automatic line numbering routine. The format for using NUM is:

```
NUM start, inc
```

where start is the starting line number and inc is the increment between line numbers.

O

OBJECT CODE — OBJECT CODE is produced from an assembly language file. OBJECT CODE is machine language comprised of a series of eight bit bytes. The bytes are instruction codes for the 6502 microprocessor to perform specific activites. A disassembler can reproduce some of the assembly language program which is used to produce the OBJECT CODE. See MACHINE CODE.

OCTAL — An OCTAL number system is one based on eight symbols (three bits). The OCTAL system uses only digits 0 through 7. Each column from right to left is a higher power of eight. The first column is the 1 s column, then 8 s, 64 s, 512 s, etc. The OCTAL system is very rarely used in eight bit computer systems like the Atari. The hexadecimal system, which is based on sixteen symbols, is much more convenient.

Decimal	Octal	Binary	Hexadecimal
0	0	000	0
1	1	001	1
2	2	010	2
3	3	011	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	10	1000	8
16	20	10000	10
32	40	100000	20
128	200	10000000	80
255	377	11111111	FF

OFFSET — OFFSETs are used by the BASIC cartridge program to tokenize statements in a line. The OFFSET is the number of bytes into the line where a new statement begins. This occurs after a colon (:) is encountered within a BASIC statement. See TOKENIZATION.

ON . . . GOTO — In BASIC, ON . . . GOTO is a conditional branching expression in which control is transferred to another line number depending on the value expression. The statement is used like this:

```
ON RESULT GOTO 100,110,120,130,140,150
```

If the variable RESULT has a value of 1 then the program goes to line 100. If RESULT equals 2 then the program goes to 110, and so on. ON . . . GOTO is often used to transfer control from a menu to a subroutine in a program.

OPCODE — In assembly language, the OPCODE is the one byte code generated by using a mnemonic command (JSR, LDA, etc.) in one of the various addressing modes. There are 151 OPCODEs for the 6502 processor used in the Atari computer. See the

MCS6500 microcomputer family Programming Manual by MOS Technology, Inc. for the ultimate resource for writing in 6502 machine language. Some examples are:

MNEMONIC	OPCODE
BNE	D0
DEX	CA
INX	E8
JSR	20
RTI	40

See KEYCODES for the full list of OPCODEs.

OPEN (O.) — OPEN is the BASIC command used to prepare an IOCB (channel) for input or output. Without an OPEN command, the channel cannot be used. See IOCB.

OPERAND — An OPERAND is an object upon which an operator operates. In the operation SQR(49) which produces the square root of 49 in BASIC, SQR is the operator and 49 is the OPERAND.

OPERATOR — An OPERATOR is a device which performs some function on another object. There are arithmetic, trigonometric, Boolean, logical, and assembly language OPERATORs. See OPERAND.

OPERATOR STACK — Memory between locations 256 and 511 (page 1) is known as the STACK. OPERATORs are placed in this memory during processing and the results are returned to variables and to the screen if requested. See STACK.

OPERATING SYSTEM — The OPERATING SYSTEM (OS) is the program which controls all of the functions of the computer. The OS is physically located in a 10K byte Read Only Memory (ROM) cartridge in the 400 and 800 systems. The XL series has a 14K byte OS. The OS loads or transfers some of its program information into the Random Access Memory (RAM) which can be read and written. The OS uses most of the bottom part of page zero RAM which is located from 0 to 128. Most of the memory located above 53247 is also used by the OS.

The OS handles all of the serial input and output activites to disk drives, printer, cassette, and RS232 devices. It also handles the interrupts and Central Input/Output (CIO). A detailed manual describing the OS is available in the Atari Technical Reference Notes package. The source code for the OS is also available. At least four revisions of the OS have been produced. To check the version OS you have, use the following PEEKs. In BASIC, type:

	NTSC Version		PAL Version	
	400/800 REV. A	400/800 REV.B	400/800 REV. A	400/800 REV. B
PRINT PEEK[65527]	221 ($DD)	243 ($F3)	230 ($E6)	34 ($22)
PRINT PEEK[65528]	87 ($57)	230 ($E6)	87 ($57)	88 ($58)

	1200XL	600XL/800XL
PRINT PEEK(65521)	1	X (X=PRODUCT CODE)
PRINT PEEK(65527)	Y	Y (Y=Internal Revision for OS)

This information is important for software developers. By examining these locations, the programmer can determine what type of Atari computer the user has and can make changes or jumps to different routines accordingly. See REVISION B OS.

OPTION BASE — In Microsoft BASIC, OPTION BASE is used to specify the first subscript in an array. The value can be either a zero (0) or a one (1). The command OPTION BASE 0 will start numbering at 0, while OPTION BASE 1 will start at 1.

OPTION CHR — In Microsoft BASIC, the OPTION CHR command reserves or frees up memory needed by the character set data. OPTION CHR1 reserves the normal 1,024 bytes for character set data. OPTION CHR2 reserves 512 bytes for the uppercase only character set as in text modes 1 and 2. OPTION CHR0 leaves no memory reserved for the character set. If you use OPTION CHR0 you must load in a new character set and point at it in location $2F4 (756 decimal).

OPTION PLM — In Microsoft BASIC, OPTION PLM0-2 reserves 0, 640 or 1,280 bytes respectively for players and missiles construction. You must still POKE 53277 with 3 to enable the player-missile graphics and also POKE 559 with 46 or 62 for double or single line player resolution.

OPTION RESERVE — In Microsoft BASIC, OPTION RESERVE X will reserve X bytes for your own machine language routine. Use VARPTR(RESERVE) to return the starting address of the X bytes which you have reserved.

OPTION — The OPTION key is located right below the SYSTEM RESET key on the Atari 400, 800 and XL series, and above the 7 key on the 1200 XL. Location $D01F (53279 decimal) is the console key register which is scanned to detect the pressing of a console key. Bit 2 is assigned to the OPTION KEY. Bit 0 is for the START key and bit 1 is for the SELECT key. By PEEKing at the value of 53279, one can determine which key or combination of keys has been pressed. The values are not stored in this register and thus should be scanned regularly for fast action.

VALUE IN 53279 ($D01F)	CONSOLE KEYS PRESSED
0 00000000	OPTION, SELECT and START
1 00000001	OPTION and SELECT
2 00000010	OPTION and START
3 00000011	OPTION

ORIGINATE MODE — A modem in the ORIGINATE MODE is ready to begin talking to another modem in the answer mode. The modem in the answer mode will begin making a high pitched signal. The modem in the ORIGINATE MODE will begin transmitting when it hears the 2025 to 2225 Hz tone at the other end. Most modems switch automatically between answer and originate.

OS/A+ — Disk Operating System for Atari computers (also for Apple) published by Optimized Systems Software, Inc. OSS wrote Atari DOS 2.0S and Atari BASIC. OS/A+ comes in Version 2 and Version 4. Version 2 is suitable for drives accessing less than 256K bytes per disk. This includes Atari 810's and Percom double density drives. Sectors are linked via two bytes at the end of each sector. A file may start at sector 100, go to 110, jump to 200, continue through 205, jump to 300, and then end. Version 4 uses a file mapped system similar to CP/M. Operation of CP/A+ V.4 is in fact functionally very close to CP/M by Digital Research, Inc. Sectors are linked only when a previously defined block is filled up.

OS/A+ uses a Console Processor (CP) after it is booted from disk. CP is similar to the File Management System (FMS) used by Atari DOS 2.0S. A wide variety of utilities are supplied with the system, but many must be loaded from disk. Resident commands available from the CP when you see the D1: prompt are: DIRectory, PROtect, UNProtect, ERAse, RENname, LOAd, SAVe, RUN, and CARtridge. Optimized Systems Software, Inc.

OUTPOST ATARI — This monthly column in Creative Computing magazine is a good source of information about the Atari computer. The topics are aimed at beginners and intermediate users. Unfortunately for us Atari users, this column is often the only Atari specific information in the magazine.

OVERLAYS — A technique borrowed from the old days when computer memory was very expensive, can be used to run large programs in very limited memory. The technique is called OVERLAYing. It is used by Wordstar to pack many more features than would normally fit into a 64K computer. OVERLAYing in BASIC can be done by organizing your program into discrete blocks which perform different functions and have grouped line numbers. Lines 0 to 100 could be initialization, 101 to 1000 could be data, 1001 to 5000 could be the main area, 5001 to 6000 function #1, 6001 to 7000 function #2, and so on. The OVERLAYS will be brought in by ENTERing the OVERLAY subroutine from the main program. This means that the OVERLAY program must be in the LIST format, not SAVEd, and there must be a non-numbered statement at the end of the OVERLAY program. The non-numbered statement will prevent the program from stopping when the OVERLAY is loaded in because it will be executed in the immediate mode. The non-numbered statement should be something like GOTO 100 or some type of jumping statement. An easy way to construct this type of file is to use an Atari DOS word processor such as Text Wizard or Atari Writer. Other OVERLAYS can be swapped in and out of the same line number area allowing a large program to be simulated in as little as 16K of RAM.

OVERSCAN — The beam of a television tube is scanned beyond the edges of the tube face. This OVERSCAN leaves no margin on the screen. Televisions and receivers operate differently and some software may actually show some very untidy behind-the-scenes operation on the sides of the screen. The Operating System sets the margins in BASIC to start at column 2 in order to compensate for any overscan effects. The register to control this margin is in location 82. Try POKE 82,0 to set the left margin to 0. POKE 83,37 will set the right margin to 37.

OVERSTRIKE — On a printer, the ability to back up and print another character over one which has been already printed is called OVERSTRIKE. This technique is often used on legal documents, but it has limited uses otherwise. Some foreign languages require overstrike on English characters to print properly.

P

PADDLE — The PADDLE controllers are hand held devices with rotatable knobs for input to the Atari computers. The controllers are actually potentiometers which can be adjusted from about 3 to 750,000 ohms by turning the knob. An analog to digital converter in the Atari computer translates the knob position into a number and puts it in the paddle registers. There are eight PADDLES. They can be read by using the BASIC command PADDLE(X) where X is a number for the paddle between zero and seven. The XL series has only four paddle ports because PORT B is taken over for Operating System duties. The value read from the paddle results in an integer between 1 and 228 depending upon the rotated position of the knob.

The PADDLE registers, called POT0 to POT7, are located from 53760 ($D200) to 53767 ($D207). These registers, which are shadowed in locations 624 to 631 ($0270 to $0277), can be read with a PEEK to find the PADDLE position between 0 and 228. Hook up a pair of paddles to port 1. Type and run the following program.

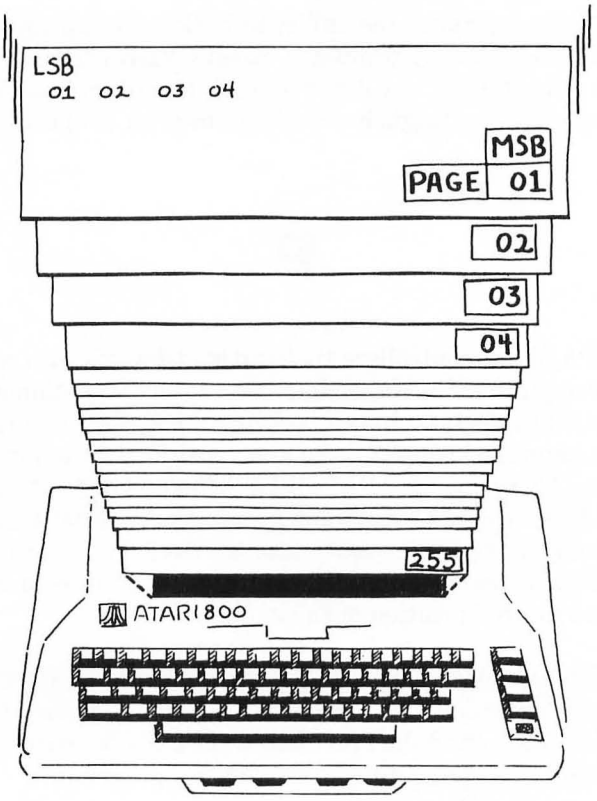
```
10 ?PEEK(624),PEEK(625):GOTO 10
```

Turn the knobs and watch the values change.

PAGE — A PAGE is the term used to describe a 256 byte block of memory in the Atari computer. There are 256 PAGES of 256 bytes each. The most important ones are: PAGE 0 — which is used for the operating system; PAGE 1 — which is the stack; PAGES 2, 3, and 4 — which are used by the Operating System; PAGE 6 — which is user RAM; and PAGES C0 through CF — which are unused by the current Operating System.

PAGE 0 — The area of memory from 0 to 255 ($00 to $FF) is special to the 6502 processor. All instructions which reference PAGE 0 (zero page instructions) are faster (require fewer machine cycles). The lower half, 0 to 127 ($00 to $7F), is used by the OS. The upper half is available for use except for $D4 to $FF, which is reserved for the floating point processing routines.

PAGE 1 — This area of memory from 256 to 511 ($100 to $1FF) is the 6502 hardware stack. These 256 bytes are used by instructions and interrupts which must store some data temporarily to which they must come back. The assembler instruction JSR, for example, writes to this stack the location to which it must return. When the RTI instruction is encountered, the address is removed from the stack. On power up, the stack pointer points at the top of the stack (511 or $1FF). As it is filled, the pointer moves down towards the bottom (256 or $100).



PAGE 6 — The area of memory called PAGE 6 is generally useable for short machine language routines which you may call from BASIC programs. This area is located between $0600 and $0700 or 1536 and 1792 in decimal. Programs placed here will not be deleted when SYSTEM RESET is pushed nor will BASIC write over them. The area is not invulnerable, however. Any input from disk, cassette or keyboard which exceeds the usual buffer length of 128 bytes will overflow into the bottom half of PAGE 6 and mess up your program or data. Thus, memory between $0600 and $067F is NOT safe but the area from $0680 to $06FF is safe to use. This is your best bet for scrounging 128 free bytes which are protected from system use.

PAGINATION — PAGINATION refers to the breaking up of a text file into pages during the printing or formatting stage. If you are using tractor feed paper with perforations, it is very important that your word processing software be able to do PAGINATION.

PAL — Phase Alternation by Line is a television receiver standard used in Europe, China, Australia, and South America. The NTSC standard is used in the United States. For computer users, the type of television receiver is important. NTSC televisions in the U.S. use 525 scan lines per frame with 30 frames per second. PAL televisions use 625 scan lines per frame at 25 frames per second. PAL sets do not have tint controls. The operating system must know these details for timers and interface requirements. You can check which version of Atari computer you have, PAL or NTSC, by looking for

a label stuck on the top. It will be labeled DOM or P for PAL. If the label is not there, check the OS cartridge. It will be labeled CX810-P if it is a PAL version. The "-P" will be missing if is an NTSC version. You can also try ?PEEK(58383); if you get a 249, you have a PAL version of the Operating System. Location 53268 ($D014) is also an ID for PAL systems. If the value in this location is not 14 ($E) or higher, it is a PAL system. See NTSC.

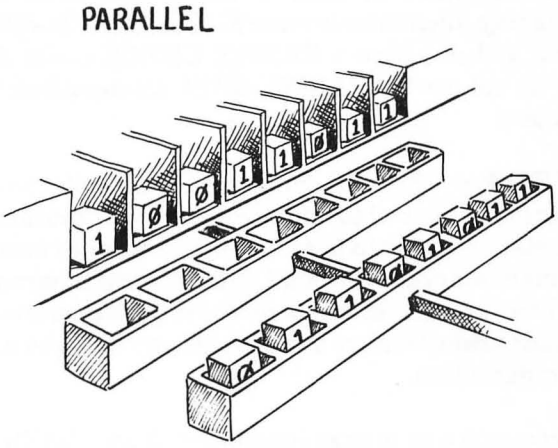
PARALLEL PORT — The 850 interface module has three different kinds of sockets: the SIO sockets (2) which connect to the computer, disk drive, or cassette drive; the serial ports of which there are four; and the parallel port. The parallel port differs from the serial port in the way that the data is transmitted. In a serial port, data is sent in the form of an eight bit byte in a series of high and low signals representing the appropriate ASCII characters. In the parallel port the data is sent over eight lines simultaneously. This is the way most printers receive data. See the figure.

DATA → 1 → 0 → 1 → 0 → 1 → 0 → 0 → 1

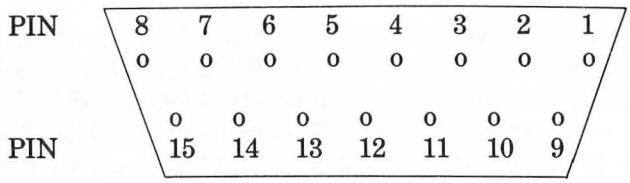
Serial Transmission - Single Data Line

DATA → 1 Line 1
0 Line 2
0 Line 3
1 Line 4
0 Line 5
1 Line 6
0 Line 7
1 Line 8

Parallel Transmission — 8 lines



A strobe signal is used to time the pulses so that all are sent on time and regularly. The 850 needs a +5V signal on Pin 12 to operate. This voltage must be supplied from the printer or the interface will not work. If no +5V source is available, Pin 12 can be connected to Pin 9 to supply a constant +5V source. The pins of the parallel port have the following functions:



Parallel Port Connector on 850 Interface Module

1	Data Strobe	9	+5V Pull up
2	Data Bit 0	10	Not used
3	Data Bit 1	11	Signal Ground
4	Data Bit 2	12	Fault
5	Data Bit 3	13	Busy
6	Data Bit 4	14	Not Used
7	Data Bit 5	15	Data Bit 7
8	Data Bit 6		

You can make your own cable for your printer by using the proper DB15 male connector and multi-conductor wire. See CABLES.

PARAMETER— A PARAMETER is a piece of information, usually a number, which is required by a function or utility so that it can do its job.

PARITY CHECK — In modem communications, a PARITY CHECK is a routine which adds all seven of the bits which makes up an ASCII character (byte) and compares the result to a known outcome. The sum of the bits will be either 1 or 0, depending upon the character. One bit may be reserved to check whether the sum is truly the 1 or 0 that it is supposed to be. An error due to line noise may have caused a bit to be transmitted incorrectly and a PARITY CHECK can help identify such errors. A parity error can be signaled with a flag, such as an inverse X. Note that it is still possible to have an error in transmission and not have a PARITY CHECK error. Atari to Atari communications usually do not use the PARITY CHECK since all eight bits may be used for ATASCII characters.

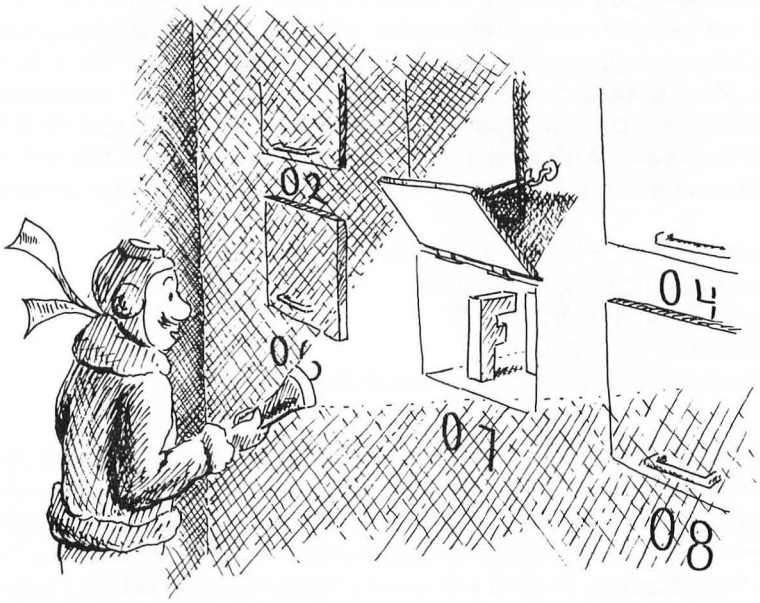
PASS PARAMETERS — The USR function, for example, has facilities to PASS PARAMETERS to transfer control to another program or routine. The USR function needs several parameters in order to pass them. The memory location of the machine language routine is mandatory for use of a USR. After that, parameters such as variable values, strings, line numbers, or whatever the routine operates on, are listed in the USR statement. These parameters are then passed onto an area of memory called the STACK and used for execution.

PEEK— The most common way to examine memory from a BASIC program is to use a PEEK statement. PEEK is just what it sounds like. It is a look into computer memory. There are 65,536 locations to examine, and to look at one just type PRINT PEEK(X), where X is the number of the location you want to examine. To look at all of the memory, try this:

```
FOR X=Ø TO 65535:PRINT PEEK(X):NEXT X <Return>
```

This will print out a list of the decimal values of all memory locations, many of which will be 0 (See POKE). To see what the characters are in these locations, change the program to:

```
FOR X=Ø TO 65535:PRINT CHR$(PEEK(X));:NEXT X <Return>
```



PEEK (7)

PEN — In BASIC XL, the PEN function reads the contents of the light pen registers and returns the value. PEN(0) reads the horizontal position register and PEN(1) reads the vertical position register.

PERCOM DRIVES — The Atari 810 disk drive is not the only alternative for your Atari computer. Percom makes drives which can operate in single or double density mode. Double density allows storage of up to 178K bytes per floppy disk. PERCOM DRIVES come with a different controller than the 810. You may get a master drive which will cost $100 to $200 more than the slave, but it will control four drives. Atari 810 s come with a controller in each drive and are all priced the same. Also, it is not possible to flip Percom disks to use the back side for storage. This is possible on the 810 s because the timing hole near the center of the disk is not used. The PERCOM DRIVES use these timing holes. Since the holes are asymmetrically placed, the disks cannot be flipped successfully. PERCOM DRIVES are shipped with a disk speed of 300 RPMs. They should be adjusted down to the normal 288 RPMs for use on Atari computers.

PERSONAL FINANCE PROGRAMS — One of the classic uses for a personal computer is to maintain home finances. Several commercial programs are available to do this for you. Generally, using a program only to balance your checkbook is not worth the effort of loading the program, entering the data, running the program, and trans-

ferring the results. Addition and subtraction are more efficiently done on a calculator. Input errors and data collection often make checkbook maintenance a self-defeating effort.

Budgeting and recordkeeping are legitimate and useful activities for the Atari computer. Be aware that you will have to carefully plan to set up your own categories for expense and income with every program you buy. No software can anticipate everyone's income from babysitting, lemonade sales, and options trading, nor the even wider range of expenses. Most packages are limited to a certain number of transactions due to memory limitations of the computer. For most households, a 48K system will handle all transactions for a year. With a microcomputer, small businesses will begin to feel the limitations of memory and the number of options. Some of the programs available are:

Money Processor by Luck Software
Financial Wizard by Computari
Family Cash Flow by APEX
Bookkeeper by Atari, Inc.
Home Accountant by Continental Software

PERSONALITY BOARD — The PERSONALITY BOARD contains the Operating System on ROM chips. This is the cartridge which sits in the front slot inside the case of the 800. The PERSONALITY BOARD takes up memory between locations $D800 and $FFFF in the 400 and 800 models. Various upgrades or enhancements to the board are available. A Fastchip from Newell will speed up some floating point operations but will prevent some software from executing. A monitor chip from DCY Computing, called Omnimon, gives you a machine language monitor, but several programs will not run if the monitor chip is activated. Revision B ROMs will cure the early problem of serially connected peripherals going to sleep.

PIA — The Peripheral Interface Adaptor chip handles controllers connected to the joystick ports. The PIA occupies memory locations between $D300 and $D3FF (54016 and 54271 decimal). Several printer and modem interfaces are now available to plug into these ports, thus eliminating the need for the 850 interface module.

PINBALL GENRE — Quite a few pinball games are available for Atari computers. The following list is merely a compilation of those offered at this time.

David's Midnight Magic - Broderbund Software
Raster Blaster - Budge Co.
Pinball Construction Set - Electronic Arts
Night Mission - Sublogic Corporation
Bulldog - Hayden
Zero Gravity Pinball - Avant Garde Creations, Inc.

PITCH — PITCH refers to the number of dots or characters per inch. Standard typewriters type at 12 PITCH for elite and 10 PITCH for pica type. PITCH can also refer to the number of pixels per inch on your television display.

PIXEL — PIXEL is a derivation of two words – PICTURE and ELEMENT. A PIXEL is the smallest controllable object or resolution element on a display screen. The limiting factor may be the computer or the display. In the highest resolution graphics mode (mode 8), a PIXEL is one scan line high by one-half color clock wide. This is essentially the resolution of a home television receiver. Each PIXEL requires one bit of memory to control it in monochrome mode.

PLAYERS — The PLAYER is Atari's solution to high speed animation of objects on the screen. In many other types of computers, it is necessary to move large amounts of data through the area of RAM called screen memory in order to achieve animation. The image must be drawn, erased, changed, and redrawn for each movement. Since memory is linear and the image is a two-dimensional map, keeping track of the image while it is moving and changing is a big job, even for a microprocessor operating at 2 to 3 mHz. PLAYERS are a simple solution to animation. An image is created in a map 128 bytes long (for each of the four PLAYERS). The map can be 256 bytes long if single line resolution instead of double line resolution is used. (Resolution is set by bit D4 in register 54272.) The map is put directly onto the screen over whatever else may be present. To move the PLAYER vertically, the image is shifted — in one dimension — up or down in the PLAYER RAM area. Horizontal motion is controlled by a hardware register called HPOSP0 (53248 or $D000). There are actually four of everything discussed here to handle up to four PLAYERS. A number between 0 and 255 is placed in the HPOSP register and the PLAYER is moved to that horizontal location on the screen. Vertical and horizontal motions can be combined. PLAYERS have their own independent color registers COLPM0-4 (53266-70 or $D012-$D015) into which color values for the PLAYERS can be placed.

PLAYERS can have different priorities with respect to each other and the playfield. Priority refers to the visiblity of one image over another when one passes through the same part of the screen. The priorities are set in register GPRIOR at 623 ($26F). PLAYERS can also collide with other PLAYERS, missiles or playfields. There are 15 registers in the GTIA area which are read-only locations in which you can detect various collision combinations. See COLLISIONS and MISSILES.

For more detailed information see *De Re Atari*, Section 4; *Compute!*, Oct. 81, Dec. 81, Feb. 82 and May 82.

PLAYFIELD — The PLAYFIELD is the largest part of the display screen. It is the part on which the text is printed and players and missiles are maneuvered. The PLAYFIELD can be made narrower or wider by POKEing location 559 ($22F) with a 33 or 35. The color of the PLAYFIELD is controlled by the color registers (708 to 712 or $2C4 to $2C8). While the large playfield is usually referred to as the PLAYFIELD, the background, text window, and characters are also technically PLAYFIELDs.

PLOT — PLOT is the BASIC command which puts a dot (pixel) on the screen at the row and column specified in the PLOT command. PLOT X,Y will put a pixel at column X and row Y. The color of the dot is specified by the COLOR statement which looks at the corresponding color register to get the color to plot. If none is specified, the background color register will be used.

PLOTTER — A PLOTTER is an output device which will draw lines based on software control from the computer. PLOTTERs can draw in color if they pick up different colored pens as commanded by the software.

PMADR — In BASIC XL, the PMADR function returns the address in memory of any of the players or missiles. The format is PMADR(exp), where exp is the player number (0 through 3) or missile number (4 through 7).

PMBASE — PMBASE is the register used to hold the page number of the start of the player missile data. PMBASE is located at $D407 (54279). This data is used to hold the bit mapped image of the players. One or two Kbytes of memory must be reserved, usually by lowering RAMTOP, depending upon whether you are in single line or double line resolution.

PMCLR — In BASIC XL, the PMCLR statement clears (writes 0s to) the player missile data area specified. The format is: PMCLR playnum where playnum is the number of the player area to clear. If playnum is from 4 to 7, all of the missile data is cleared.

PMCOLOR — In BASIC XL, the PMCOLOR statement works just like the SETCOLOR command, but for the players and missiles.

PMGRAPHICS — In BASIC XL, the PMGRAPHICS (PMG.) enables or disables Player Missile GRAPHICS, presumably by reserving memory for the data. PMG.0 shuts off the PM graphics, PMG.1 enables single line resolution, and PMG.2 enables double line resolution.

PMMOVE — In BASIC XL, PMMOVE allows movement of a player to a location on the screen. The format is:

```
PMMOVE playnum,horiz;vert
```

where playnum is the number of the player to move, horiz is the horizontal position on the screen to which to move (0 to 255), and vert is the vertical position (–255 to 255).

Note that this statement is much easier to implement than the subroutines required in Atari BASIC. Horizontal movement is facilitated by a hardware register, but vertical movement requires shifting data through memory.

PMWIDTH — In BASIC XL, PMWIDTH allows the horizontal resolution of the players and missiles to be changed. The format is PMWIDTH playnum,exp where playnum is the player or missile number and exp is 1, 2, or 4 for the number of color clocks per bit in the horizontal direction.

POINT — In Atari BASIC, POINT is the command used to set the pointer in a buffer for reading or writing data from a disk. In the POINT statement, you must specify the absolute sector number and the byte within the sector. A channel (IOCB) must be opened in order to use POINT. See NOTE and IOCB.

POINTER— A POINTER is a location in memory, a register, or a variable which tells the computer WHERE to find another piece of information or a subroutine. It contains a number which is an address, table position, or a location usually in two bytes. Why use a POINTER? In order to change an activity, such as what happens when you press the BREAK key or SYSTEM RESET, you would just have to change the contents of the POINTER, not the entire program which executes after the key is pressed. It is just like rotating a sign on a street, pointing you in a different direction; for instance, to a party on your block instead of on someone else's. This is the basis of the term INDIRECTION.

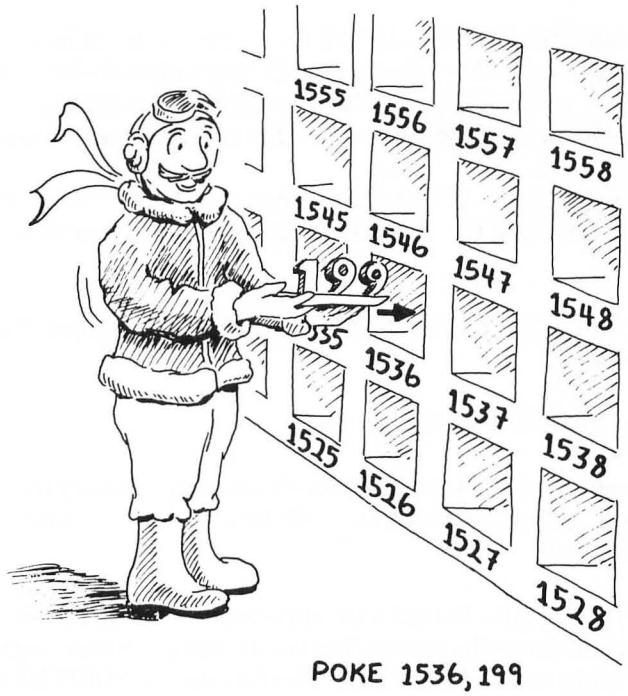
POKE — POKE is the way to change the contents of memory from BASIC. Unlike PEEK, which only allows you to look at memory, POKE allows you to write as you wish. You can POKE safely at any of the free RAM areas. Try this:

```
POKE 1536,199 <Return>
```

You have just changed location 1536 from 0 to 199. If you do not believe it, type:

```
? PEEK(1536) <Return>
```

and see that there is really a 199 in 1536. If you try to POKE a location which is assigned to ROM, you will not be able to change it. Try to POKE a number into 65535. If you then PEEK at 65535, you will find a 192 regardless of the value you POKEd. That is because it is a ROM address. If you POKE some critical RAM area which the computer needs for operation, you can crash the system and you will have to restart the computer to correct it. Machine language programs perform many POKEs during loading.



POKE 1536, 199

POKEY — The POKEY chip is a custom integrated circuit used by Atari computers to handle various input and output activities. Access to the POKEY routines is gained through registers located between 53760 and 54015 ($D200 and $D2FF). POKEY handles the keyboard scanning, Serial I/O, interrupt requests, random number generation, paddle controllers, and sound.

POP — POP is used to clear the stack of the address used during a loop or branching statement. It is an abort to get out of a loop or branch without causing an error. See STACK.

PORT A — The hardware register in the PIA chip (6520) which reads data from the joysticks or paddles connected to controller jacks 1 and 2. PORT A is at 54016 ($D300).

PORT B — In the 400 and 800 computers, PORT B is the same as PORT A, except for controller jacks 3 and 4. In the XL series, there are no controller jacks 3 and 4 and PORT B is used for completely different applications. During the vertical blank, the OS reads the PORT A values and writes them in the PORT B shadows (634 for joystick jack 3, 635 for jack 4, 640, 641, 642, and 643 for paddle triggers 4, 5, 6, and 7). This means that any game which used jacks 3 and 4 will work in jacks 1 and 2.

PORT B is located at 54017 ($D301). PORT B is used to:

- Disable/enable the OS ROM
- Disable/enable the BASIC ROM
- Turn on/off LEDs on the 1200XL
- Disable/enable the self test ROM

Bit 0 is used to disable the OS ROM. If bit 0 is one, the OS ROM is enabled. If it is zero, then the RAM is free and the OS is disabled and a new OS can be loaded in. No new OS's are available at this writing. The hardware registers for GTIA, POKEY, PIA, and ANTIC are always located in their usual positions, just because they are hard wired.

Bit 1 is used to disable BASIC. If bit 1 is zero, then BASIC is enabled. If bit 1 is one, then BASIC is enabled. Pressing the OPTION button as the power is turned on disables BASIC.

Bits 2 and 3 turn on LEDs on the 1200XL when the bits are zero. The LEDs turn off when the bits are one.

Bits 4, 5, and 6 are reserved for future use.

Bit 7 is used to disable the self-test ROM which takes up memory from $5000 to $57FF when it is enabled. A zero enables the self test and a one disables the self-test, freeing up this memory.

POSITION — The BASIC POSITION statement is used to put the cursor at a specified X and Y location on the screen. Text or plotting can then begin there. The cursor need not be visible to use POSITION. The format for POSITION is:

POSITION x,y

where x is the horizontal column number and y is the vertical row number.

POST RECORD GAP — On a cassette file, the POST RECORD GAP (PRG) is a section of up to one second of miscellaneous data written to separate one record from another. The POST RECORD GAP is actually between the data of one record and the Pre-Record Write Tone of the next record.

POWER-UP — The Operating System performs initial activities whenever the power to the computer is turned on or a coldstart is called. The activites done by the OS during POWER-UP are as follows.

1. Find the highest page of free RAM available. Put it in location 6 ($06) and then transfer it to 106 ($6A)
2. Write zeros to all free RAM.
3. Set up RAM interrupt vectors.
4. Set up the device handler table, HATABS at 794-831 ($31A-$33F)
5. Put screen in graphics 0 mode.
6. Boot cassette if START key is pressed.
7. Check cartridge slots to see if disk should be booted.
8. Boot DOS if drive is on line.
9. Transfer control to cartridge, program in RAM or MEMO PAD.

POWER SUPPLIES — A POWER SUPPLY is an electrical transformer which takes household voltage at 110 volts and reduces it to between five and nine volts for use by the personal computer. Although power supplies for the 810 disk drive, the 400 and 800 computers and the 850 interface look similar and will appear to be interchangeable, but they are not. The 810 has the biggest potential problem because it is the device which uses the most power. All of the power supplies output nine volts AC except for the XL series power supplies which output five volts DC. The computers and interface will work from any Atari power supply provided the plug fits in the socket. The 810 needs the larger power supply, rated at 31 watts (it may be labeled 31 VA). The smaller supplies are rated at 15.3 watts and will result in speed variations when the motor on the drive calls for more current. Check your system to make sure that the 810 drive is using the 31 watt power supply.

PRE-RECORD WRITE TONE — On a cassette file, the PRE-RECORD WRITE TONE is either a three second mark tone (5327 Hz), if the file is in the Normal IRG mode, or a .25 second mark tone if it is a Short IRG mode file. THE PRWT is used between all adjacent records (128 bytes) in a file.

PRWT (Pre-Record Write Tone) MARKER
RECORD DATA
PRG (Post Record Gap)
PRWT
MARKER
RECORD DATA
PRG
.
.
.

PRINT (PR. or ?) — PRINT is the BASIC command to send a character or value to a device. The question mark is a handy abbreviation for PRINT. Characters can be sent to a disk drive, cassette, printer, or screen though an OPEN channel.

PRINT USING — In Microsoft BASIC and BASIC XL, PRINT USING is available on the extension disk. It is a formatting statement used primarily for printing dollar amounts. A pound sign (#) is used to reserve a place for each digit. For example, PRINT USING "$# # # #.# #" would be used to print dollar and cents values up to $9999.99 with the cents place always maintained. Substituting an ampersand (&) for the pound sign (#) would fill all leading blanks with zeros.

PRINTER BUFFER — A PRINTER BUFFER or spooler is a peripheral device which will accept data bound for your printer at a much higher rate than the printer could print it. The PRINTER BUFFER then signals your Atari that it is finished and you can have your computer back for other tasks while the buffer continues to dump its data to the printer at its normal rate of 20 to 160 characters per second. This is a time saving tool for users who write a lot.

PRINT . . . SPC — In Microsoft BASIC, PRINT followed by SPC(X) will insert X spaces between the last character printed and the next character to be printed.

PRINT . . . AT — In Microsoft BASIC, PRINT #IOCB, AT(X,Y) can be used in two ways. The first way is if the IOCB is OPENed as the screen, the value of a variable can be printed at the X and Y location specified. The second way is if the IOCB is OPENed as a disk, the PRINT..AT statement can be used to write the value of a variable to sector X and byte starting at Y, providing the sector is included in a file designated for that sector.

PRINTER TABLES — The huge variety of printers available for personal computers makes translation of software difficult. Although there are some conventions in printer control codes, there are many differences. The following tables are designed to allow translation of some codes for the more popular printers. This is signified by an esc in the table. You can send the decimal codes by using the CHR$(X) statement, where X is the decimal code listed in the table. Some codes require an ESC to be sent before the control byte. To do this, you can send a CHR$(27) and then the code or else hit ESC ESC in order to produce an ESCape character (if you are in BASIC). In LETTER PERFECT, you can send the decimal control codes by using the CTRL-V before the code. In ATARI WRITER, you use a CTRL-O before the code. Almost all codes can be sent as either CHR$ values which are the decimal equivalents of the ATASCII character called for or else by sending the ATASCII string. The string is generated by placing the character inside the quotes which hold the string. The following charts are not exhaustive; they are meant to give a common point of reference among the printers on the market.

ATARI PRINTERS

	ATARI 825		ATARI 1025		ATARI 1027	
FUNCTION	**Dec**	**Keystrokes**	**Dec**	**Keystrokes**	**Dec**	**Keystrokes**
Sound Buzzer		-na-		-na-		-na-
BACKSPACE	8	CTRL-H		-na-		-na-
HORIZONTAL TAB		-na-		-na-		-na-
LINE FEED	10	CTRL-J	10	CTRL-J	10	CTRL-J
VERTICAL TAB		-na-		-na-		-na-
TOP OF FORM		-na-		-na-		-na-
CARRIAGE RETURN	13	CTRL-M	13	CTRL-M	13	CTRL-M
DOUBLE WIDTH ON	27 14	esc CTRL-N	27 14	esc CTRL-N		-na-
CONDENSED ON	27 20	esc CTRL-T	27 20	esc CTRL-T		-na-
CONDENSED OFF	27 19	esc CTRL-S	27 15	esc CTRL-O		-na-
DOUBLE WIDTH OFF	27 15	esc CTRL-O	27 15	esc CTRL-O		-na-
DP MODE (200cps)		-na-		-na-		-na-
CORRESPONDENCE		-na-		-na-		-na-
ESCAPE	27 27	esc esc	27 27	esc esc	27 27	esc esc
UNDERLINE ON	15	CTRL-O	27 25	esc CTRL-Y	27 25	esc CTRL-Y
UNDERLINE OFF	14	CTRL-N	27 26	esc CTRL-Z	27 26	esc CTRL-Z
LINE FEED = 1/8"		-na-	27 56	esc 8		-na-
LINE FEED = 7/72"		-na-		-na-		-na-
LINE FEED = 1/6"		-na-	27 54	esc 6		-na-
ITALICS ON		-na-		-na-		-na-
ITALICS OFF		-na-		-na-		-na-
IGNORE PAPER OUT		-na-		-na-		-na-
ENABLE PAPER OUT		-na-		-na-		-na-
RESET ALL FUNCTIONS		-na-		-na-		-na-
EMPHASIZE ON		-na-		-na-		-na-
EMPHASIZE OFF		-na-		-na-		-na-
DOUBLE STRIKE ON		-na-		-na-		-na-
DOUBLE STRIKE OFF		-na-		-na-		-na-
SUPERSCRIPT		-na-		-na-		-na-
SUBSCRIPT		-na-		-na-		-na-
RESET SUB/SUPER		-na-		-na-		-na-
UNIDIRECTIONAL ON		-na-		-na-		-na-
BI-DIRECTIONAL		-na-		-na-		-na-
PROPORTIONAL SPACE	27 17	esc CTRL-Q		-na-		-na-
INTERNAT'L CHARS ON		-na-	27 23	esc CTRL-W	27 23	esc CTRL-W
INTERNAT'L CHARS OFF		-na-	27 24	esc CTRL-X	27 24	esc CTRL-X
64 COLUMN MODE		-na-	27 83	esc S	27 83	esc S
80 COLUMN MODE		-na-	27 76	esc L	27 75	esc L

NOTE: Documentation for the 1025 and 1027 printers is scarce.

EPSON PRINTERS

	EARLY mx-80		w/GRAFTRAX		FX-100	
FUNCTION	**Dec**	**Keystrokes**	**Dec**	**Keystrokes**	**Dec**	**Keystrokes**
Sound Buzzer	7	CTRL-G	7	CTRL-G	7	CTRL-G
BACKSPACE		-na-	8	CTRL-H	8	CTRL-H
HORIZONTAL TAB	9	CTRL-I	9	CTRL-I	9	CTRL-I
LINE FEED	10	CTRL-J	10	CTRL-J	10	CTRL-J
VERTICAL TAB	11	CTRL-K	11	CRTL-K	11	CTRL-K
TOP OF FORM	12	CTRL-L	12	CTRL-L	12	CTRL-L
CARRIAGE RETURN	13	CTRL-M	13	CTRL-M	13	CTRL-M
DOUBLE WIDTH ON	14	CTRL-N	14	CTRL-N	14	CTRL-N
CONDENSED ON	15	CTRL-O	15	CTRL-O	15	CTRL-O
CONDENSED OFF	18	CTRL-R	18	CTRL-R	18	CTRL-R
DOUBLE WIDTH OFF	20	CTRL-T	20	CTRL-T	20	CTRL-T
DP MODE (200cps)		-na-		-na-		-na-
CORRESPONDENCE		-na-		-na-		-na-
ESCAPE	27 27	esc esc	27 27	esc esc	27 27	esc esc
UNDERLINE ON		-na-	27 45 1	esc CTRL-A	27 45 1	esc CTRL-A
UNDERLINE OFF		-na-	27 45 0	esc CTRL-,	27 45 0	esc CTRL-,
LINE FEED = 1/8"	27 48	esc 0	27 48	esc 0	27 48	esc 0
LINE FEED = 7/72"	27 49	esc 1	27 49	esc 1	27 49	esc 1
LINE FEED = 1/6"	27 50	esc 2	27 50	esc 2	27 50	esc 2
ITALICS ON		-na-	27 52	esc 4	27 52	esc 4
ITALICS OFF		-na-	27 52	esc 5	27 53	esc 5
IGNORE PAPER OUT		-na-	27 56	esc 8	27 56	esc 8
ENABLE PAPER OUT		-na-	27 57	esc 9	27 57	esc 9
RESET ALL FUNCTIONS		-na-	27 64	esc @	27 64	esc @
EMPHASIZE ON	27 69	esc E	27 69	esc E	27 69	esc E
EMPHASIZE OFF	27 70	esc F	27 70	esc F	27 70	esc F
DOUBLE STRIKE ON	27 71	esc G	27 71	esc G	27 71	esc G
DOUBLE STRIKE OFF	27 72	esc H	27 72	esc H	27 72	esc H
SUPERSCRIPT		-na-	27 83 0	esc S CTRL-,	27 83 0	esc S CTRL-,
SUBSCRIPT		-na-	27 83 1	esc S CTRL-A	27 83 1	esc S CTRL-A
RESET SUB/SUPER		-na-	27 84	esc T	27 84	esc T
UNIDIRECTIONAL ON		-na-	27 85 1	esc U CTRL-A	27 85 1	esc U CTRL-A
BI-DIRECTIONAL		-na-	27 85 0	esc U CTRL-,	27 85 0	esc U CTRL-,
PROPORTIONAL SPACE		-na-		-na-		-na-
DOUBLE WIDTH ALL		-na-	27 87 1	esc W CTRL-A	27 87 1	esc W CTRL-A

OTHER PRINTERS

	GEMINI-10		NEC 8023A		OKI MICROLINE	
FUNCTION	**Dec**	**Keystrokes**	**Dec**	**Keystrokes**	**Dec**	**Keystrokes**
Sound Buzzer	7	CTRL-G		-na-		-na-
BACKSPACE	8	CTRL-H	8	CTRL-H		-na-
HORIZONTAL TAB	9	CTRL-I	9	CTRL-I	9	CTRL-I
LINE FEED	10	CTRL-J	10	CTRL-J	10	CTRL-J
VERTICAL TAB	11	CTRL-K	11	CTRL-K	11	CTRL-K
FORM FEED	12	CTRL-L	12	CTRL-L	12	CTRL-L
CARRIAGE RETURN	13	CTRL-M	13	CTRL-M	13	CTRL-M
DOUBLE WIDTH ON	14	CTRL-N	14	CTRL-N	31	esc CTRL-*
CONDENSED ON	15	CTRL-O	27 81	esc Q	28	esc CTRL—
CONDENSED OFF	18	CTRL-R	18	CTRL-R	30	esc CTRL-+
10 CPI MODE	27 66 1	esc B CTRL-A	27 78	esc N	30	esc CTRL-+
12 CPI MODE	27 66 2	esc B CTRL-B	27 69	esc E	29	esc CTRL-=
17 CPI MODE	27 66 3	esc B CTRL-C	27 81	esc Q	28	esc CTRL—
DOUBLE WIDTH OFF	20	CTRL-T	20	CTRL-T	30	esc CTRL-+
DP MODE (200cps)		-na-		-na-	27 48	esc 0
CORRESPONDENCE		-na-		-na-	27 49	esc 1
ESCAPE	27 27	esc esc	27 27	esc esc	27 27	esc esc
UNDERLINE ON	27 45 1	esc CTRL-A	27 88	esc X	27 45 1	esc CTRL-A
UNDERLINE OFF	27 45 0	esc CTRL-,	27 89	esc Y	27 45 0	esc CTRL-,
LINE FEED = 1/8"	27 48	esc 0	27 65	esc A	27 56	esc 8
LINE FEED = 7/72"	27 49	esc 1		-na-	27 78 7	esc N CTRL-G
LINE FEED = 1/6"	27 50	esc 2	27 66	esc B	27 54	esc 6
ITALICS ON	27 52	esc 4		-na-		-na-
ITALICS OFF	27 53	esc 5		-na-		-na-
IGNORE PAPER OUT	27 56	esc 8		-na-		-na-
ENABLE PAPER OUT	27 57	esc 9		-na-		-na-
RESET ALL FUNCTIONS	27 64	esc @		-na-	27 24	CTRL-X
EMPHASIZE ON	27 69	esc E	27 33	esc !	27 84	esc T
EMPHASIZE OFF	27 70	esc F	27 34	esc "	27 73	esc I
DOUBLE STRIKE ON	27 71	esc G		-na-	27 72	esc H
DOUBLE STRIKE OFF	27 72	esc H		-na-	27 73	esc H
SUPERSCRIPT	27 83 0	esc S CTRL-,		-na-	27 74	esc J
SUBSCRIPT	27 83 1	esc S CTRL-A		-na-	27 76	esc L
RESET SUB/SUPER	27 84	esc T		-na-	27 77	esc M
UNIDIRECTIONAL ON	27 85 1	esc U CTRL-A	27 91	esc [	27 79	esc O
BI-DIRECTIONAL	27 85 0	esc U CTRL-,	27 93	esc]	27 80	esc P
PROP. SPACE ON	27 90	n esc Z n	27 80	esc P	27 78 n	esc N n

	MANNESMAN			
	SMC TP-1		TALLY	
FUNCTION	Dec	Keystrokes	Dec	Keystrokes
Sound Buzzer	7	CTRL-G	7	CTRL-G
BACKSPACE	8	CTRL-H	8	CTRL-H
HORIZONTAL TAB	9	CTRL-I	9	CTRL-I
LINE FEED	10	CTRL-J	10	CTRL-J
VERTICAL TAB		-na-	11	CTRL-K
FORM FEED	12	CTRL-L	12	CTRL-L
CARRIAGE RETURN	13	CTRL-M	13	CTRL-M
DOUBLE WIDTH ON		-na-	14	CTRL-N
CONDENSED ON		-na-	15	CTRL-O
CONDENSED OFF		-na-	18	CTRL-R
10 CPI MODE		-na-	18	CTRL-R
12 CPI MODE		-na-		-na-
17 CPI MODE		-na-	15	CTRL-O
DOUBLE WIDTH OFF		-na-	20	CTRL-T
DP MODE (200cps)		-na-		-na-
CORRESPONDENCE		-na-		-na-
ESCAPE	27 27	esc esc	27 27	esc esc
UNDERLINE ON	25	CTRL-Y	27 88	esc X
UNDERLINE OFF	25	CTRL-Y	27 89	esc Y
LINE FEED = 1/8"		-na-	27 48	esc 0
LINE FEED = 7/72"		-na-	27 49	esc 1
LINE FEED = 1/6"		-na-	27 50	esc 2
ITALICS ON		-na-	27 52	esc 4
ITALICS OFF		-na-	27 53	esc 5
IGNORE PAPER OUT		-na-	27 56	esc 8
ENABLE PAPER OUT		-na-	27 57	esc 9
RESET ALL FUNCTIONS		-na-	27 64	esc @
EMPHASIZE ON		-na-	27 69	esc E
EMPHASIZE OFF		-na-	27 70	esc F
DOUBLE STRIKE ON		-na-	27 71	esc G
DOUBLE STRIKE OFF		-na-	27 72	esc H
SUPERSCRIPT		-na-	27 83 0	esc S CTRL-,
SUBSCRIPT		-na-	27 83 1	esc S CTRL-A
RESET SUB/SUPER		-na-	27 72	esc H
UNIDIRECTIONAL ON		-na-	27 85 1	esc U CTRL-A
BI-DIRECTIONAL		-na-	27 85 0	esc U CTRL-,
PROP. SPACE ON		-na-	27 80	esc P

PRINTER ECHO — The following machine language routine will make your PRINTER ECHO every line which is sent to the screen. The routine fits in page 6. There is no way to use the printer as a typewriter (to print each key as it comes from the keyboard). This is mainly because the E: device gets statements when the RETURN key is pressed.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** FILE: ECHO.BAS
30 REM ** PRINTER ECHO PROGRAM
40 REM ** TURN ON PRINTER & EVERY LINE
50 REM ** WRITTEN TO SCREEN WILL GO TO
60 REM ** PRINTER ALSO.
70 REM **
80 FOR X=1536 TO 1667:READ Y:POKE X,Y:
NEXT X
90 DATA 160,15,185,0,228,153,131,6,136
100 DATA 16,247,169,131,141,33,3,169,6
,141,34
110 DATA 3,169,74,141,137,6,141,70,3,1
69,6
120 DATA 141,138,6,141,71,3,169,51,141
,135
130 DATA 6,169,6,141,136,6,133,8,108,2
50,191
140 DATA 141,146,6,32,63,6,32,96,6,208
,23
150 DATA 173,5,228,72,173,4,228,72,173
,146
160 DATA 6,96,32,96,6,173,7,228,72,173
,6,228
170 DATA 72,173,146,6,174,147,6,172,14
8,6
180 DATA 96,141,146,6,142,147,6,140,14
8,6
190 DATA 201,32,144,21,173,55,228,72,1
73,54,228
200 DATA 72,173,146,6,201,125,240,4,20
1
210 DATA 156,208,2,169,155,96,32
220 Z=USR(1536)
```

Another, simpler technique is to change two memory locations so that data is sent to the printer instead of the screen. This can be done by POKE 838,166 and POKE 839,238. To return to the screen-only mode, POKE 838,175 and POKE 839,242.

PRINTER UNIT — In the XL Operating System, it is now possible to address one of eight printers by specifying the device (P1: through P8:). This is due to a change in the device handler which allows a unit number in the IOCB.

PRIORITY CONTROL REGISTER — The PRIORITY of a player, missile, or playfield refers to which image will be visible or which will be hidden when a collision occurs. Register $026F (623 decimal) is the shadow for $D01B (53275) which controls these priorities. Six bits (D0 through D5) are used for priority control. Two of the bits in this register (D7 and D6) are used to set GTIA modes 9, 10, and 11.

SETUP	PRIORITY LIST
POKE 623,1	PLAYERS 0,1,2,3 PLAYFIELDS 0,1,2,3 BACKGROUND
POKE 623,2	PLAYERS 0,1 PLAYFIELD 0,1,2,3 PLAYERS 2,3 BACKGROUND
POKE 623,4	PLAYFIELD 0,1,2,3 PLAYER 0,1,2,3 BACKGROUND
POKE 623,8	PLAYFIELD 0,1 PLAYER 0,1,2,3 PLAYFIELD 2,3 BACKGROUND
POKE 623,16	Add four missiles to make another player
POKE 623,32	Overlapping players take another color
POKE 623,64	Set GR.9
POKE 623,128	Set GR.10
POKE 623,192	Set GR.11

PROGRAM LIBRARY — As a computer hobbyist, you will probably begin to collect hundreds of short programs in your personal library. As soon as you have more than a few dozen programs, it becomes difficult keeping track of their whereabouts. PROGRAM LIBRARY distributed by APX and written by Ron and Lynn Marcuse is an indispensable tool for maintaining a catalog. The program was published in *Compute!* in October 1981 and later distributed by APX on disk. Each disk is assigned a code and an entry is written on the disk in a file called DISK.CAT. Disks containing commercially prepared software should not be written on and can be cataloged manually. PROGRAM LIBRARY reads the directory of an Atari DOS disk and notes the filespec and size in sectors. The program then asks for a short description of the file, the date acquired, and the type of file (game, utility, etc.) All of this data is put into a file and you can then sort, print reports, search for a program or disk number, and more. The program as it comes can handle around 300 records. It can be modified to handle more.

PROGRAM COUNTER — The PROGRAM COUNTER (PC) is a register in the 6502 processor. Actually, it is comprised of two 8 bit registers which act as a 16 bit register. The PC has two parts, the PCL or program counter low (for the low byte) and the PCH or program counter high (for the high byte). The PC is actually a program address pointer which is used to address or choose the next memory location from which to fetch an instruction or data. The number placed in the PC addresses one of the 65,536 eight bit data words (bytes) in the computer's address space. The selected word is transmitted through the 16 address lines in the processor bus system.

PROM — Programmable Read Only Memory. A PROM is an integrated circuit, or chip, into which programs or code can be written one time and read out many times. A PROM is programmed in a device called a PROM burner. The PROM burner actually melts tiny circuits in the chip so that an irreversible change is made which captures the program. PROMs are used to make small quantities of read-only programs as the time to produce one copy is quite long.

PROMPT — A PROMPT is a character or figure which appears on the display screen, asking you or waiting for you to input some data. The familiar "READY" in Atari BASIC is a PROMPT. In FORTH, "ok" is often used as a prompt.

PROPORTIONAL SPACING — PROPORTIONAL SPACING is a technique of printing text with variable widths between characters and words. The Atari 825 printer is capable of PROPORTIONAL SPACING if used with the correct software. Most printers and programs displaying text on the screen do not use PROPORTIONAL SPACING. Although it is easier to read proportional print, typing programs which are printed this way is often more difficult.

PROTECT — In BASIC XL, the PROTECT statement is used to lock a file. A PROTECTed file cannot be deleted or rewritten. The file will be erased if the disk is formatted.

PSEUDO ARRAY — Since Atari BASIC does not handle string arrays, PSEUDO ARRAYS must be used. (Microsoft BASIC does allow string arrays in multiple dimensions.) A string array is a set of subscripted string variables; e.g., STRING$(1), STRING$(2), STRING$(3), etc., where each subscripted string variable is different. A PSEUDO ARRAY divides one big string into multiple, equally sized, substrings which can be accessed by counting from the first element of the string. All elements must be the same size.

PTRIG — In Atari BASIC, PTRIG is the command to look at the Paddle TRIGger to see if it is being pressed. There are eight PTRIG's to look at, one for each paddle (0 through 7). A zero indicates the paddle trigger is engaged, A one means it is open. PTRIG(X) is the statement to look at paddle X. The PTRIG registers are located at $027C through $0283 (636 through 644 decimal).

PUBLIC DOMAIN SOFTWARE — Many programmers have written useful utilities and games which are not usually suitable for commercial distribution but have some utility. Often these programs are put in the PUBLIC DOMAIN. This means that

the programs can be freely exchanged unlike copyrighted software. The best place to find PUBLIC DOMAIN SOFTWARE is in the established user groups. The larger groups may have 20 to 40 disks or cassettes full of PUBLIC DOMAIN programs. Other sources are bulletin board systems and magazines. Notable PUBLIC DOMAIN programs are: Jonesterm, Amodem, the AMIS BBS, and a bug chasing game called Myriapede.

PUBLIC DOMAIN SOFTWARE — Many programmers have written useful utilities and games which are not usually suitable for commercial distribution but have some utility. Often these programs are put in the PUBLIC DOMAIN. This means that the programs can be freely exchanged unlike copyrighted software. The best place to find PUBLIC DOMAIN SOFTWARE is in the established user groups. The larger groups may have 20 to 40 disks or cassettes full of PUBLIC DOMAIN programs. Other sources are bulletin board systems and magazines. Notable PUBLIC DOMAIN programs are: Jonesterm, Amodem, the AMIS BBS, and a bug chasing game called Myriapede.

PUT and GET — GET is the BASIC command to retrieve a byte from a device through an OPEN channel. The format for GET is just like PUT except that with GET, input from the K: device (keyboard) is allowed. Using GET with the program recorder or disk drive will call in a 128 byte block into an appropriate buffer and each GET will retrieve a character from the buffer until it is empty. More GETs will call in the next block of 128 bytes.

R

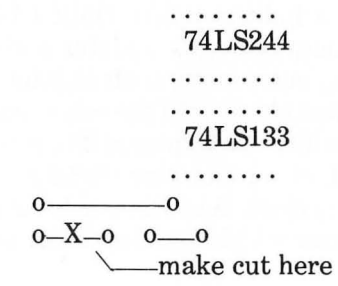
RAD — In Atari BASIC, the RAD statement sets up trigonometric functions to evaluate arguments in radians as opposed to degrees. The command DEG switches the expectation to degrees. Radians is the default mode on powerup.

RAM — Random Access Memory. RAM is a term which commonly refers to a type of integrated circuit used to store data while a computer is in operation. RAM refers to the technique of addressing any location in the RAM without searching through all of the data. A floppy disk is also a Random Access Memory, but it is almost never referred to as such. A more descriptive term for RAM is Read/Write Memory since RAM can be cleared by removing the power and new data can be entered and used. RAM comes packaged in a DIP (Dual In-line Package) which is made of plastic or ceramic. The RAM circuit may have 8K, 16K, 64K, or 256K bits of memory per chip. In order to get 16K bytes of memory in the Atari, eight 16K bit RAM chips are required.

RAMDISK — Axlon, Inc. makes a peripheral card for the Atari 800 which allows you to add enough memory to simulate a disk drive. During your sessions, you would store all files in the 128K RAMDISK. When you finish, you save the contents of the RAMDISK to a floppy disk. The advantage is the high speed at which files are stored and retrieved in RAM. Filemanager 800+ from Synapse is configured to work directly with the RAMDISK. The RAMDISK is actually organized as eight 16K blocks. Writing a

value to the range $0FC0 to $0FFF or to the range $CFC0 to $CFFF switches from one bank to the next. The address bus cannot address more than 64K. An Integrator board from ADS allows most software to operate with the RAMDISK.

One disadvantage of the RAMDISK is that if software loads into the $0FC0 to $0FFF range, it will crash. The following modification will cause the other range mentioned above to be the bank switch. Find pin 18 on the edge of the 16K RAM card. This should go to pin 1 of IC Z501. Connect a 6" piece of insulated wire to this pin by soldering or sticking the wire in the socket. Connect the other end of the wire to pin 15 of 74LS133 on the RAMDISK board. Now you have to cut one trace. Use the following diagram to find and cut the trace. With this modification, you will not need to switch out the 128K when a program crashes with RAMDISK. This modification was developed by David Young and is included in the Omnimon instruction manual from CDY Consulting.



RAMTOP — RAMTOP is the name given to the register at location 106 ($6A). The value stored in this location is the number of pages (256 byte blocks) which are available for use. A 48K system has RAMTOP set at 160 on power-up for a RAMTOP value of 40960. The Operating System, hardware registers, and ROM cartridges are located above RAMTOP. You can forcibly lower RAMTOP by POKEing in a lower number and saving a few extra pages of RAM for character set data, or player data, or programs. First read the value in 106 by PEEK(106) and then POKE in the same number minus the number of pages you want saved.

TRY: POKE[106],PEEK[106]–4 to save four (4) pages. BASIC will not write into this four page reserved area.

RANDOM ACCESS — RANDOM ACCESS to a storage device means that you can go more or less directly to some area and read data which is stored there. A disk file can be accessed randomly and RAM can be accessed randomly. Cassette files, which must be read serially in order to get to a specific location, are not RANDOM ACCESS. NOTE and POINT commands are used to randomly access a disk file. RAM can be read by PEEK in BASIC, LDA in assembly, or @ in FORTH.

RANDOMIZE — In Microsoft BASIC, RANDOMIZE supplies a new seed to the RND function to ensure that a truly random number results.

RASTER — RASTER refers to the type of cathode ray tube in which the beam is scanned horizontally across the face of the tube from top to bottom. Other types of CRT's use vector drawing in which the image is drawn directly with the beam.

RCA PLUG — The plug at the end of the cable on the Atari 400 and 800 computer which connects to the TV antenna box. This is the same type of plug used to connect stereo receivers, amplifiers, turntables, and tape decks together. The cable is usually coaxially shielded to prevent interference from garbling the signal to the television. These cables are available at very low prices at your local electronics shop.

READ (REA.) — READ is the BASIC command used to start inputting data from a series of DATA statements. A READ with no DATA will cause an error.

READY — This is the prompt that tells you that the initialization sequence has been successful and your BASIC language is ready for use. If the disk drive is on and no DOS or proper boot sequence is available, then no READY prompt will be given.

REALTIME CLOCK — The Atari computer has a built-in REALTIME CLOCK which can be used to time events or to delay activities. The clock updates registers located at 18, 19 and 20 ($12, $13 and $14) once during every vertical blank interrupt. Actually, location 20 ($14) is incremented every vertical blank until the value reaches 255. Then location 19 ($13) is incremented once. When location 19 passes 255, it resets to zero and increments location 18. See *ANTIC* Vol. I, No. 4, Oct/Nov 1982 for a program by Pete Goodeve which produces a very accurate clock. A continuously running clock could be used to make a realtime schedule program which would alert the owner to meetings, appointments, or events.

The following program will put a clock in the upper right corner of your screen. Type it in, save it as CLOCK.BAS, and then type RUN. Nothing will happen, but you should type the time as HHMMSS. The colons should appear after HH and MM. When you finish the SS, the clock should appear and the program will be erased from memory.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** REAL TIME CLOCK IN CORNER
30 REM ** FILE D:CLOCK.BAS
40 REM ** AFTER TYPING RUN, TYPE IN TI
ME
50 REM ** HHMMSS
60 REM
70 TRAP 120
80 FOR A=1 TO 300
90 READ MRTC
100 POKE A+1535,MRTC
110 NEXT A
120 X=USR(1536)
130 REM
140 DATA 162,0,32,199,6,24,42,42,42,42
,141,255
150 DATA 6,32,199,6,24,109,255,6,157,2
40,6,232
160 DATA 224,3,240,11,169,58,141,251,2
,32,208,6
```

```
170 DATA 76,2,6,173,49,2,133,205,173,4
8,2,24
180 DATA 105,63,133,204,144,2,230,205,
169,0,141,14
190 DATA 212,169,79,141,36,2,169,6,141
,37,2,169
200 DATA 64,141,14,212,76,0,160,162,0,
202,208,253
210 DATA 248,14,162,0,173,243,6,105,1,
141,243,6
220 DATA 201,96,144,96,142,243,6,173,2
42,6,105,0
230 DATA 141,242,6,201,96,144,35,142,2
42,6,173,241
240 DATA 6,105,0,141,241,6,201,96,144,
20,142,241
250 DATA 6,173,240,6,105,0,141,240,6,2
01,18,144
260 DATA 5,169,1,141,240,6,173,221,6,2
08,41,24
270 DATA 162,2,160,8,189,240,6,72,41,1
5,9,16
280 DATA 145,204,104,136,106,106,106,1
06,41,15,9,16
290 DATA 145,204,136,169,26,145,204,13
6,202,16,225,200
300 DATA 169,0,145,204,76,98,228,142,2
54,6,32,226
310 DATA 246,174,254,6,142,254,6,32,17
0,246,41,15
320 DATA 234,174,254,6,96,0,0,0,0,0,0,
0
```

RECORD— On a cassette file, a RECORD is a group of 128 bytes. This is similar to a sector on a disk file. In a database file, a RECORD is part of the file and it is comprised of fields.

REDEFINED CHARACTER SET — The normal characters you see when you type on the keyboard are contained in ROM within the Atari computer. A pointer is used to tell the Operating System where to go to find the character set data. The pointer is called CHBAS and is located at 756 ($2F4). By using your own character set data and changing the pointer to point to your data, you can use a REDEFINED CHARACTER SET. See CHARACTER SET.

REGISTER— A REGISTER is a location in memory which may be written to or read from by the computer. The computer is made of REGISTERs with numbers assigned from 0 to 65536 in decimal or $0000 to $FFFF in hexadecimal. Each REGISTER is comprised of eight bits. A bit is a digit which can be either a 1 or a 0. Bits are used to compose binary numbers. In order to address 65536 locations, 16 bits must be

used. This is derived from taking the value of 2 to the 16th power (65,536). Some of the 65,536 locations are hardware REGISTERs. These REGISTERs are wired directly to some devices, such as a joystick trigger. Other REGISTERs act as storage locations for data flowing in and out. Since some operations need only one bit as a signal and each register has eight bits, multiple uses can be gained from registers. REGISTER 54016 ($D300) uses four of the eight bits to monitor the paddle triggers. In other combinations, the same REGISTER looks for joystick movements and the keypad controller. A memory map, such as the very detailed *Mapping the Atari* by Compute! Books will describe the functions of all known REGISTERs.

RELATIONAL OPERATOR — RELATIONAL OPERATORs are used for the comparison of variables. The RELATIONAL OPERATORS are <, >, =, < >, >=, and <=. The RELATIONAL OPERATOR sets up a condition which may be true or false. Depending upon the condition, a branch or some other activity may occur.

```
For Example: IF X > Y THEN 100
```

In this statement, if the value of X is greater than Y, then the program will jump to line 100. If X is not greater than Y, then the next line will be executed.

REM (R. or .) — REM is used only for the purpose of inserting REMarks or comments in a BASIC program. REMarks are intended to give credit to an author or to clarify the actions of a particular line number in a program. REMs are not needed to run any BASIC program, but if you delete a REM statement to which a GOTO or other branch is sent, an error will occur. Be careful when deleting REM statements.

RENUMBERER — BASIC programs often need RENUMBERing for publication in a newsletter or magazine. Several commercial programs will do this job without the author having to manually go through the program and change every line number. These programs will even change arguments hidden in GOTOs and GOSUBs. Some of the fastest RENUMBERers are included in the BASIC Commander package by MMG Software, BASIC XL, and the Monkey Wrench. RENUMBERing of even very large BASIC programs is essentially complete before you get your finger off of the RETURN key. Other programs by APX, and those in the public domain, are not nearly as fast but will produce the same end result.

RENAME — In BASIC XL, RENAME is a statement which performs the same function as the E option in Atari DOS 2.0. A DOS file can be RENAMEd with this command. The format for using RENAME is:

```
RENAME "D:NEWNAME.BAS,OLDNAME"
```

In Atari DOS 2.0, the E function for RENAME file asks for the old name first followed by the new name to be given to the file.

RENUM — In Microsoft BASIC and BASIC XL, RENUM is a command available on the extension diskette which will RENUMber your BASIC program. The format for using RENUM is:

```
RENUM start,inc
```

where start is the first line number and inc is the increment between lines.

RESET — To initiate a coldstart, that is, to reboot the disk drive and start the power-up sequence, POKE 580,1 from BASIC. When SYSTEM RESET is pressed, the coldstart will occur. Immediately after booting, memory location 8 contains a 0 until SYSTEM RESET is pressed. After RESET is pressed, location 8 contains a 255. This location is vectored (pointed) to 58484 ($E474) which handles the warmstart routine. If SYSTEM RESET is changed to do a coldstart, then the initialization address can be put in locations 12 and 13 ($0C and $0D) for the next code to execute. See the SYSTEM RESET CHANGER to RUN a BASIC program from a RESET.

RESIDENT DISK HANDLER — The Resident Diskette Handler is not the same as the File Management System (which incidentally is resident in RAM). This handler does not use the CIO utility so it does not have an entry in the device handler table. It is used by filling in the proper parameters in the device control block (DCB) and jumping through the DISKINV vector at 58451 ($E453).

RESTORE — The RESTORE command in BASIC resets a pointer which moves down data in a DATA statement. Even if all of the DATA is not read, RESTORE will reset the pointer to the top of the DATA so that the first piece of data will be read next.

RESUME — In Microsoft BASIC, RESUME transfers control back to a line which caused an error. RESUME NEXT transfers control to the line after the line which caused the last error.

RETURN — After a subroutine is executed, a RETURN command is needed to send control back after the GOSUB. The line number to which the RETURN is supposed to go is left on the stack unless a POP command is executed.

RETURN KEY — The RETURN KEY generates a CHR$(155) code which sends the cursor to the first column and moves down one line (or moves the text up one line if on the bottom screen line). The logical line is then sent to an input buffer for processing.

REV. B ROM — The Operating System for Atari computers has been revised several times. Revision A had several irritating bugs, such as the drive and printer going to sleep for long, unpredictable periods. revision B of the OS fixed this and several other bugs. To test the version in your 400 or 800, type the following line while in the immediate mode.

```
PRINT PEEK(58383) < Return >
```

If you get a 0 result, you have REVISION B. If you get a 56, you have REVISION A. If you get a 249, you have the European PAL version of the OS. See OPERATING SYSTEM.

REVERSE LINE FEED — The Atari 825 printer is one of the few printers available which can perform a REVERSE LINE FEED. This means moving the paper back into the printer instead of out as it normally does. With this feature it is possible to print double columns and to overstrike characters for foreign language printing.

REVERSE POLISH NOTATION — RPN is a syntax used for mathematical operation where the arguments are put on a stack and the operators are entered after the arguments. This differs significantly from regular algebraic notation (a + b). RPN would use (a b +) for this operation. FORTH uses RPN for its math handling.

RF SWITCHBOX — In order to connect the Atari computer to your television receiver, you should install your RF SWITCHBOX on the antenna terminals. The switchbox will allow you to switch to regular television or to the computer without using a screwdriver to reconnect the terminals.

RFI — RFI refers to Radio Frequency Interference. RFI is caused by switching signals inside the computer at radio frequencies. The metallic shielding in all Atari computers and peripherals effectively prevents most interference with radios and television broadcasts. Hardware modifications often leave gaps through which interference signals can leak out. Printers are especially bad for interference because of inadequate shielding. The interference will not affect the human body other than the irritation it can cause by obscuring your favorite TV show. The Atari computer is (RFI-wise) the quietest computer. This is probably because it was designed before the Federal Communications Commission released the Part 15 rules to create separate home and industrial standards (Class A and Class B).

RGB — RED, GREEN, BLUE — This terminology refers to a color monitor which is capable of a very high graphics resolution. RGB monitors can handle graphics over 1,000 by 1,000 pixels of 256 colors. RGB signals are sent such that each color has its own line and controls an individual gun in the CRT. The Atari computer cannot support such high resolution graphics because of the limited memory available. Video interfaces for the Atari are limited to an NTSC (National Television Standards Committee) signal for a monitor or an RF modulated signal for a home TV receiver.

**RIGHT$** — In Microsoft BASIC II, RIGHT$ returns the rightmost characters of a string according to the number specified.

RND — The RND command is used in BASIC to generate random numbers. In BASIC XL, this statement is replaced by RANDOM. The RND statement must be accompanied by an argument, as in RND(0), for use in a program. RND(0) is treated just like a variable in a program. To print a list of random numbers, try this short program:

```
10 FOR X=1 TO 10
20 PRINT INT(100*RND(0))
30 NEXT X
```

You can also use PEEK(53770) to get random numbers between 0 and 255 much more quickly than you can through BASIC and RND(0).

ROM VECTOR — A ROM VECTOR is a memory location which contains the information on the address of a particular routine which is contained in the Operating System Read Only Memory. The ROM VECTOR is used so that Atari programmers can make upgrades to current Operating Systems and so that existing software will still work. If programmers stick to using standard ROM VECTORS then the contents of the vector can be changed but compatibility will remain. This problem is quite serious with the Operating System used in the Atari 1200XL and the XL series. When programmers jump right to the routine without using the ROM VECTOR, and Atari has changed the location of the routine, existing software will not work. (See XL OPERATING SYSTEM.)

ROUND OFF — A simple technique to round off a number to the nearest 1s, 10s 100s, etc. is to add a small increment and then take the integer of the new number. For example, to round off 10.14 to the nearest .1, add .05, multiply by 10, take the integer of the sum and divide by 10. In BASIC, INT((10 * (10.14 + 0.05)))/10 will round off to the nearest .1.

ROW — A ROW is a horizontal line of characters or pixels on the screen.

RS232C HANDLER — The R: device is not resident in the OS ROM. This means that when you start up your system, you do not have the ability to use the RS232 port unless you do several things. First, make sure the 850 interface is connected to the serial bus via the black cables used for the disk drives and computer. You must have the 850 ON when you boot the system. You will hear a two or three second tone after DOS is booted in, assuming you have your TV volume turned up. This tone is the signal that the RS232 handler program has loaded in. Voila, you now have a new device added to the handler table. A modem, voice synthesizer, or other device can be attached to the serial port on the 850. Make sure the parameters are set up correctly. See HANDLERS.

The following program will build an RS232 booting file in an AUTORUN.SYS format. Type in this program and put it on your modem program disk. Run it and hit OPTION when you have inserted your formatted disk and are ready to write the file onthe disk. To use it, make sure your 850 interface is on when you boot the RS232 handler loader. Listen for the tone to tell you it is loaded. You can append another file to this AUTORUN.SYS file to run another BASIC program such as a menu of Jonesterm. Use the C option in DOS with the /A for append. See APPEND.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** PROGRAM NAME: RS232.BAS
30 REM ** PUBLIC DOMAIN PROGRAM
35 REM
40 PRINT CHR$(125)
50 PRINT " HIT OPTION TO WRITE AN "
60 PRINT " RS 232 BOOTING AUTORUN.SYS"
70 PRINT " FILE.  "
80 IF PEEK(53279)=3 THEN 100
90 GOTO 80
100 OPEN #1,8,0,"D:AUTORUN.SYS"
110 FOR I=1 TO 88
```

```
120 READ D
130 PUT #1,D
140 ? D;" ";
150 NEXT I
160 CLOSE #1
170 END
180 REM DATA IS LOADED AT 14336 (DEC)
190 DATA 255,255,0,56,75,56,169,80
200 DATA 141,0,3,169,1,141,1,3,169
210 DATA 63,141,2,3,169,64,141,3,3
220 DATA 169,5,141,6,3,141,5,3,169
230 DATA 0,141,4,3,141,9,3,141,10
240 DATA 3,141,11,3,169,12,141,8,3
250 DATA 32,89,228,16,1,96,162,11
260 DATA 189,0,5,157,0,3,202,16,247
270 DATA 32,89,228,48,6,32,6,5,108
280 DATA 12,0,96
290 DATA 226,2,227,2,0,56
```

RTCLOK — This is the label for the realtime clock in the Atari computer. It is available through registers $12, $13, and $14 (18, 19 and 2Ø in decimal). See REAL TIME CLOCK.

RTI — RTI is the assembly language mnemonic code for ReTurn from Interrupt. An RTI must be the last instruction in a routine which is run during an interrupt. The RTI returns the 6502 processor back to the main program.

RUN — The BASIC command RUN will begin execution of any BASIC program which is in memory. RUN can be used to LOAD and RUN a tokenized (SAVEd) BASIC program from a disk or cassette tape. RUN"D:TESTPROG" will LOAD a program called TESTPROG from disk and RUN it when it is in memory. This is essentially the same as typing LOAD"D:TESTPROG", waiting for the READY prompt, and then typing RUN. RUN can be used as a command in a BASIC program.

RUN TIME PACKAGE — A RUN TIME PACKAGE is a utility which is loaded and required for some compiled language programs. The RUN TIME PACKAGE is essentially a language in and of itself which allows the compiled code to run as a more compact and faster program.

RUN TIME STACK — The STACK is a one page long block of memory, located at page 1 between locations 256 and 511 ($0100 and $01FF), which is used by the Operating System, BASIC, and DOS for storage, notes, processing, and so on. A pointer points to the top of the STACK at $01FF and the STACK moves downward toward $0100 everytime the program encounters a GOSUB, ON, FOR, JSR or PHA. When the program meets the corresponding GOTO or GOSUB from the ON statement, the NEXT from the FOR/NEXT, RTS, RTI or PLA, it reads from the top of the STACK. There are

only 256 bytes available in this STACK, and if more than 256 bytes are written, it will wraparound starting at $01FF (511 decimal) again.

STACK	CONTENTS at TIME 1	CONTENTS at TIME 2
511	AA	FF
510	01	AA
509		01
508		
507		

RUN ONLY PROGRAM — A RUN ONLY PROGRAM is one that can only be RUN, not LOADED, LISTed, or examined in any way. This protection is added by altering the BASIC Statement Cursor Pointer in locations $008A and $008B (138 and 139 decimal). To get a RUN ONLY BASIC PROGRAM, add the following line to the end of your program which you want to protect:

```
32767 POKE PEEK(138)+PEEK(139)*256+2,0:S
AVE "D:RUNONLY":NEW
```

Do not RUN this program until you have SAVEd it under another name (Not D:RUNONLY). Type GOTO 32767 <Return> and you will have your protected version saved as D:RUNONLY. You will only be able to RUN this program by the command RUN"D:RUNONLY". The same technique will work with C: on cassette recorders.

Another simple protection technique for BASIC programs is to change all of the variables and the statements of a tokenized program. This can be done by adding two lines to the end of your BASIC program. Be sure to SAVE the program before you RUN it with these two lines or you will never see it again. The program can only be RUN, not LISTed or LOADed; however, the program can be copied.

```
32766 FOR A=PEEK(130)+PEEK(131)*256 TO P
EEK(132)+PEEK(133)*256:POKE A,125:NEXT A
32767 POKE PEEK(138)+PEEK(139)*256+2,0:S
AVE "D:FILESPEC":NEW
```

S

SAVE — SAVE is an Atari BASIC command. SAVE will store a BASIC program as a tape or disk file when executed as SAVE"D:FILENAME.EXT" or SAVE"C:". Files are stored in a tokenized form. Tokenization is a method of translating BASIC lines into symbolic format for easier processing. For instance, the BASIC command GOSUB is tokenized as $0C instead of as the letters G. O. S. U. and B. Numbers are stored as six byte binary coded decimal numbers, so that programs with many numbers will take up

much more disk space if SAVEd than if LISTed. A SAVEd file is comprised of two parts: a series of pointers which are stored in page 0 and point into the file, and the tokenized file data. The pointers point to the Variable Name Table, the Statement Table, String/Array area, RUN stack, and MEMTOP. These pointers are visible when the SAVEd BASIC program is dumped to a printer or to the screen. None of the normal BASIC commands are visible if you examine the contents of the disk file because of the tokenization. See VARIABLE NAME TABLE and TOKENIZATION.

SAVE "S:" — You can use the SAVE"S:" command to examine the tokenized BASIC program which you have in memory. Simply LOAD in a BASIC program, and while in the immediate mode, type SAVE"S:" <Return>. The screen will clear and the tokenized program will be listed on the screen.

One further extension of the SAVE"S:" command is to examine the contents of your Atari's memory by using the screen. You must change the value of the registers which store the end of the BASIC file. You can then list out all memory to the top of memory ($FFFF). To do this, POKE 140, 255 and POKE 141, 255, then type SAVE"S:". When this has been done, your program will list, then all free memory, followed by the BASIC cartridge and the Operating System.

SAVE...LOCK — In Microsoft BASIC II, the SAVE"D:filename" LOCK command is used not only to SAVE a BASIC program to a disk file, but it prevents any user from listing, merging, or examining the file.

SCALAR — A SCALAR is an array of one element. A SCALAR cannot be subscripted as an array can. A variable such as "X" in LET X=1 is a SCALAR. See ARRAY.

SCAN LINE — See HORIZONTAL SCAN LINE.

SCIENTIFIC NOTATION — Atari BASIC will convert any number longer than nine digits or any number whose absolute value is less than 0.01 to SCIENTIFIC NOTATION. This format consists of a + or–sign, the mantissa or number which may be a floating point number with up to eight decimal places, the letter E, another + or– sign for the exponent, and the exponent (a two digit integer). The number 0.00001234 would be converted to 1.234E-05 in SCIENTIFIC NOTATION. The E-05 indicates that the decimal point should be moved five places to the left to convert to normal algebraic notation.

SCOPY 810 — This utility from Alliance Software is a very efficient program for making back-up copies of non-protected data or program disks. An entire disk can be copied in at most two passes on a full 48K system. Options allow copying to or from multiple drives, copying of selected sectors, multiple copies from the same reading, verifying writes or not verifying writes, and the option to format the disk before writing. When you boot up SCOPY you will see the following menu:

SCOPY
Insert Source Disk

SOURCE DRIVE	01
DESTINATION DRIVE	01
STARTING READ SECTOR	001

SCOPY
Insert Source Disk

ENDING READ SECTOR	2D0
STARTING WRITE SECTOR	001
NUMBER OF DESTINATIONS	01
VERIFY WRITES?	YES
FORMAT DESTINATIONS?	NO
WRITE BLANK SECTORS?	NO

You can just hit RETURN to set all of the above parameters with the default settings. When SCOPY finds and tries to copy a bad sector, it will generate an ERROR message. To continue copying around the bad sector, just hit the START key. SCOPY 810 is very compact which allows the maximum use of RAM for copying. All numbers are displayed in hexadecimal. SCOPY 810 is licensed to users' groups and is available only through these groups. The price is around $10.

SCREEN EDITOR - E: — One of the resident handlers (see HANDLERS) is the screen editor. This device is a program that accepts characters from the keyboard (K:) and sends them to the screen (S:). The device vectors for the screen editor start at 58368 ($E400). The screen editor of the Atari computer is among the best of any available on personal computers. The most outstanding feature is the ability to place the cursor on a line and have the entire logical line be sent to the editor when RETURN is pressed. Many systems require you to run your cursor over the line to the end and then press RETURN. The E: device accepts 16 special control codes which perform various editing functions. These control codes are:

FUNCTION	HEX	CHR$	KEYSTROKES
Escape	$1B	27	ESC-ESC
Cursor Up	$1C	28	ESC-Ctrl –
Cursor Down	$1D	29	ESC-Ctrl =
Cursor Left	$1E	30	ESC-Ctrl +
Cursor Right	$1F	31	ESC-Ctrl *
Clear (Screen)	$7D	125	ESC-Ctrl <
Back Space	$7E	126	ESC-Back Space
Tab	$7F	127	ESC-Tab
End Of Line	$9B	155	Inverse-RETURN
Delete Line	$9C	156	ESC-Shift Back Space
Insert Line	$9D	157	ESC-Shift Insert
Clear Tab	$9E	158	ESC-Ctrl Tab
Set Tab	$9F	159	ESC-Shift Tab
Bell	$FD	253	ESC-Ctrl 2
Delete Character	$FE	254	ESC-Ctrl Back Space
Insert Character	$FF	255	ESC-Ctrl Insert

If any of these characters are sent to the screen from a disk file or through the RS232C port, the appropriate editing function will occur. This will happen typically when you use the C. function of DUP.SYS to send a binary load file to the E: device. To use these editing functions in programs you want to list to a printer, use the CHR$ (character str-

ing) value. For example, use CHR$(125) in a BASIC program listing to clear the screen.

SCREEN MEMORY — The screen memory area is located on the memory map between MEMTOP and RAMTOP. RAMTOP is the top of BASIC RAM which is $9FFF (40959) in a 48K system on power-up and the value is found in location $006A (106). MEMTOP is the top of free BASIC RAM (where your program can go). MEMTOP is found by looking in locations 741 and 742. In between these markers you will find the memory which is used to display information on the screen. Locations $0058 and $0059 (88 and 89 decimal) contain the address of the lowest screen memory data which is the byte that will fill the upper left-hand corner of the screen. By POKEing another address into 88 and 89, you can change the memory which is accessed by the display. The Display List uses 88 and 89 for its normal pointer to the display memory.

In GR.0, the value in 88 and 89 on power-up is 40000. If you POKE 40000 with a number (try 99) you will see the corresponding internal character appear in the upper left corner of the screen. If you POKE 40000,99 you will see a lowercase letter "c" appear.

SCRIPTOR — *Compute!* magazine has published a fairly sophisticated word processor written in BASIC in the April 1983 issue. The code is over four full pages of magazine text, so it is very tedious to type in. Also, the program is quite slow because it is written in BASIC. The editing and formatting commands are quite useful for creating text files or printing reports. Format lines are implemented roughly as in Letter Perfect. A mini-DOS is available in the program by hitting the ESC key.

**SCRN$** — In Microsoft BASIC II, the SCRN$ statement looks at the position of a pixel or character on the screen (at an X,Y coordinate) and returns the color number or character when used with the ASC command. The format for use is: CHAR$= ASC (SCRN$(X,Y)) where X and Y are the horizontal and vertical positions of the pixel or character position in question.

SCROLLING — Almost every personal computer can scroll data across the screen. Scrolling is what you see when you LIST a program and it begins printing from the top of the screen to the bottom and then begins rolling off the top. This is known as coarse scrolling. Fine scrolling involves moving the data one pixel line at a time across the screen resulting in a very smooth motion. The XL series has a built-in fine scrolling function in BASIC. To fine scroll, you need only do a POKE 622,255:GR.0 <Return>. All text listed after this statement will be smooth scrolled vertically. Scrolling can be horizontal or vertical. Vertical scrolling is like the LIST example. Horizontal scrolling is a sideways motion across the screen. Eastern Front from Atari uses both horizontal and vertical scrolling. Diagonal scrolling results from a combination of horizontal and vertical scrolling.

There are two ways to produce scrolling. One is to move the data through the area of memory (RAM) that is used for screen memory. Since a 40 by 24 character screen has 960 bytes, a lot of memory must be shifted in order to scroll. A graphics 8 screen has over 8000 bytes of data which would have to be shifted in order to scroll. The short amount of time needed to shift one byte adds up considerably when thousands of bytes

are shifted and the program slows down noticeably. That is why another type of scrolling is used. Instead of moving the data through the window, the data stays put and the window is moved. Two bytes can be used to specify where the ANTIC is to find the memory to put on the screen. If these bytes are changed according to a joystick input or some other technique, we will get scrolling.

The Display List (Instructions for the ANTIC) contains an instruction called Load Memory Scan (LMS). This instruction tells the ANTIC where in memory to get the data to fill the lines which make up the display on the screen. The instruction is really comprised of three bytes:

INSTRUCTION * LOW BYTE * HIGH BYTE (of address)

The instruction for an LMS is a simple 64 ($40 hex). The address of memory for the display lines is in the familiar LO,HI format. After the LMS instruction, the display list needs a series of numbers to tell it how many lines of which graphics modes are to be put on the screen. (See DISPLAY LIST). We are interested in the address for applications to scrolling. If the address in the LMS instruction is incremented by 1, the entire portion of the screen covered by that instruction will scroll to the right. If the memory address is decreased by 1, the display will scroll to the left. A coarse vertical scroll is done by adding the total number of bytes contained in one mode line (20 or 40) to the address in the LMS.

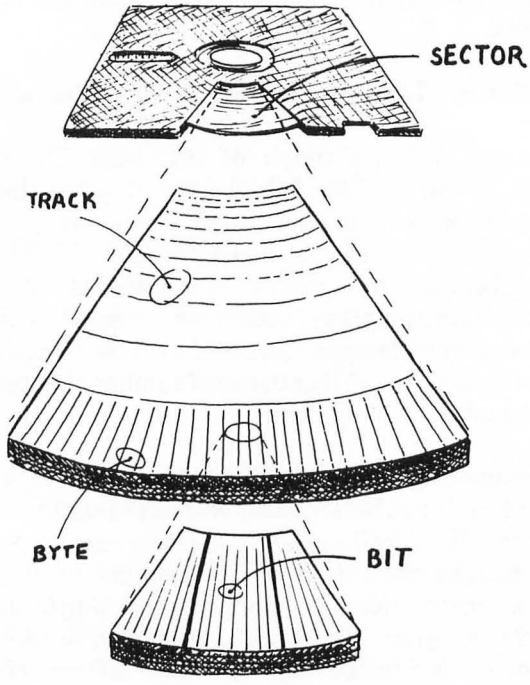
Fine scrolling is the movement of the screen data by one scan line as opposed to one character line. Fine scrolls can be done only within the confines of a character block. In other words, we cannot fine scroll across the entire screen without special preparation. Two preparations must be made to do smooth scrolling. First, you must set the appropriate bits in the LMS instruction. For horizontal scrolling this requires you to set bit 4 (add 16) to the LMS instruction. We used 64 previously, so adding 16 produces 80. The bits change as follows: LMS instruction 64= 0100 0000 → 80= 0100 1000 (bit 4 set) For vertical scrolling, bit 5 is set (add 32) on the LMS instruction. Then you must also put the number of clock cycles or scan lines to scroll in the respective hardware registers. These are 54276 ($D404) for horizontal and 54277 ($D405) for vertical. See the *Hardware Manual*, section II and *De Re Atari*, section 6 for a detailed explanation of scrolling.

SEARCH — A SEARCH routine is highly recommended for any word processor or data base program you are considering. A SEARCH by the computer will save your eyes from the task of reading an entire file and looking for a specific entry. The most common type of SEARCH is one that looks for a string of characters and stops at the beginning of the string when it is found. A replace function in combination with a SEARCH is also very useful for editing. Some word processors, such as Letter Perfect, will SEARCH for CTRL chararacters and some will not.

SECTOR — On a floppy disk, the data are layed out in concentric rings of formatted sections. The rings are called tracks and there are 40 tracks on an Atari DOS disk. The sections are called SECTORs and there are 18 sectors per track. Within each track are 128 bytes of eight bits each. There are timing marks and ID marks between SECTORs

which are not visible to the user of an 810 or similar disk drive. The disk is layed out as follows:

40	TRACKS/DISK	
18	SECTORS/TRACK	= 720 SECTORS/DISK
128	BYTES/SECTOR	= 92,160 BYTES/DISK
AND		
8	BITS/BYTE	= 737,280 BITS/DISK



SECTOR

SECTOR EXAMINER The following program will allow you to read the contents of any sector on an Atari disk. If the sector is a directory sector, the directory information will be formatted and interpreted as to the length and starting sector of the files.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** EXAMINE SECTOR UTILITY
30 REM ** BY RIC MEARS
40 REM ** FILE EXAMINE.BAS
50 REM
60 REM Equates
70 UNIT=769:ACTION=770:BUFLO=772:BUFHI
=773
80 SECLO=778:SECHI=779:REEDSEC=82:PUUT
SEC=87
```

```
90 REM Initialize
100 POKE 83,38:POKE 82,2
110 DIM BUFFER$(128),Z$(1)
120 BUFFER$=" ":BUFFER$(128)=" ":BUFFE
R$(2)=BUFFER$
130 FOR I=1536+128 TO I+4:READ A:POKE
I,A:NEXT I
140 DATA 104,32,83,228,96
150 REM Print Titles
160 ? CHR$(125):? "          Disk Examin
e Utility":?
170 ? "   WRITTEN BY RIC MEARS  4/10/1
981 ":? :?
180 ? ," NOT FOR SALE"
190 ? "      PUBLIC DOMAIN SOFTWARE":?
200 ? "   Remove line 350 if examining
 a          Non-Atari-DOS diskette. "
210 ? :?
220 REM Input Sector to Display
230 ? :? :? "  SECTOR or Return for ne
xt   "
240 ? "DIRECTORY IN 361...368  ";:TRAP
 260:INPUT SEC
250 REM Set CIO Parameters & Call
260 POKE UNIT,1
270 POKE BUFLO,ADR(BUFFER$)-INT(ADR(BU
FFER$)/256)*256
280 POKE BUFHI,INT(ADR(BUFFER$)/256)
290 POKE SECLO,SEC-INT(SEC/256)*256
300 POKE SECHI,INT(SEC/256)
310 POKE ACTION,REEDSEC
320 D=USR(1536+128)
330 REM Check for Exceptions
340 IF PEEK(771)<>1 THEN ? CHR$(125);"
 Non-existent Sector":GOTO 230
350 IF SEC>360 AND SEC<369 THEN 490
360 REM Display Sector Data
370 ? CHR$(125);" SECTOR ";SEC:LSEC=SE
C
380 ? "_____________________________
      ";
390 FOR I=1 TO 128:? CHR$(27);BUFFER$(
I,I);:NEXT I:?
400 ? "_____________________________
      ";
410 ? "File #";INT(ASC(BUFFER$(126))/4
);"    ";
```

```
420 IF ASC(BUFFER$(128))<127 THEN ? AS
C(BUFFER$(128));:GOTO 440
430 ? 125;
440 ? " BYTES     LINK --> ";
450 SEC=ASC(BUFFER$(127))+(ASC(BUFFER$
(126))-INT(ASC(BUFFER$(126))/4)*4)*256
460 ? SEC
470 GOTO 230
480 REM Display Directory Data
490 ? CHR$(125)
500 ? " DIRECTORY  Sector ";SEC:?
510 ? "STATUS     #    FILENAME    #SEC S
TART"
520 ? "------     --  ------------ --- -
-----"
530 FOR I=1 TO 8
540 L=(I-1)*16
550 IF BUFFER$(L+1,L+1)>CHR$(158) THEN
 ? "Deleted ";:GOTO 580
560 IF BUFFER$(L+1,L+1)>CHR$(0) THEN ?
 "Active  ";:GOTO 580
570 ? "          ";
580 K=ASC(BUFFER$(L+1))
590 K=K-INT(K/64)*64:IF K>=32 THEN ? "
* ";:GOTO 610
600 ? "  ";
610 K=(SEC-361)*8+I-1:? K;" ";:IF K<10
 THEN ? " ";
620 ? BUFFER$(L+6,L+13);".";BUFFER$(L+
14,L+16);
630 POSITION 28,I+4:? ASC(BUFFER$(L+2)
)+256*ASC(BUFFER$(L+3));
640 POSITION 34,I+4:? ASC(BUFFER$(L+4)
)+256*ASC(BUFFER$(L+5));
650 ? :NEXT I:? :? :SEC=SEC+1
660 GOTO 230
```

SECTOR FORMAT — The Atari disk drive contains its own microprocessor. The drive is actually a small computer with 4K of addressable memory. It contains the 6810, 6532, 2316, and FD1771 floppy disk formatter/controller chips. The 6810 addresses drive memory from $0080 to $00FF. The 6532 addresses $001B to $01FF. The 2316 addresses are $0800 to $0FFF. The microprocessor takes care of writing all of the data necessary to make and mark the sectors on the floppy disk during formatting. These data, which are between the standard 128 bytes which you can read, are listed below. You need special hardware to read or modify these bytes.

NAME OF FIELD	# BYTES	# USAGE	VALUES
Pre-sector gap	6	Marker	$00s
ID Field	1	ID Address Mark	
Track Number	1	Contains Track #	$00 to $27
Zero Marker	1	Marker	$00
Sector Number	1	Contains Sector #	$01 to $12
Sector Size	1	$00 means 128 bytes/sect $01 = 256 $02 = 512 $03 = 1024	$00
CRC	2	Cyclical Redundancy Check (Written on formatting)	$00 to $FF
Pre-data field	17	Data Field Coming Up	$00s
Data Address	1	Data Address Mark	
Data Field	128	Sector Data ($FF=blank)	$00 to $FF
CRC	2	Cyclical Redundancy Check (Written after write)	$00 to $FF
Post Sector Gap	9	Marker	$00s (0s)
Post Sector Gap	3	Marker	$FFs (1s)

The Cyclical Redundancy Check (CRC) is a type of checksum which adds the data preceding it to provide a number for comparison. If a discrepancy is found, the sector or write operation is flagged as defective.

SECTOR NUMBER — Atari disk drives format disks with 720 individually numbered sectors. Sector 0 is not readable. Sector 720 is readable, but cannot normally be written to. NOTE and POINT make use of the absolute SECTOR NUMBER for reading and writing data to a disk. Sectors are not laid out sequentially on around the tracks on a disk. They are interleaved to allow for faster reading. An early format, called the slow format, laid out the sectors on a track as follows:

1,8,15,4,11,18,7,14,3,10,17,6,13,2,9,16,5,12
SLOW FORMAT

The faster format, which is contained in a newer ROM "C" (Part # CO11299C in all units made after May 1982), formats disks so that the sectors are laid out as follows:

1,3,5,7,9,11,13,15,17,2,4,6,8,10,12,14,16,18
FAST FORMAT

The only difference is the speed at which data loads into the computer. For very high data loading rates, see the Warp Speed by Happy Computing.

SECTOR READ — The following routine will read any sector from an Atari disk and put the data in that sector into a string called A$. You can then use the string data any way that you wish. See *Antic Magazine*, May 1983, Vol. 2, No.2 to use this routine to hide ID information in sector 720. The original program and article were written by

Chuck McMath. This utility is used in the program called LJKDIR.BAS to print a directory for LJK DOS disks in the program found in the LJK DOS entry. The program finds the directory entry and puts it into a string. The appropriate filenames are extracted and chained to make a printable string.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** PROGRAM: SECTREAD.BAS
30 REM
40 DIM SECT$(44),A$(128)
50 FOR B=1 TO 44
60 READ C
70 SECT$(B,B)=CHR$(C)
80 NEXT B
90 ? "TYPE SECTOR # TO READ ";
100 INPUT SECT
110 REM ++ MAKE A BUFFER ++
120 FOR D=1 TO 128:A$(D,D)=" ":NEXT D
130 REM ++ READ EXAMPLE ++
140 X=USR(ADR(SECT$),ADR(A$),SECT,0)
150 PRINT A$
160 GOTO 90
170 END
180 DATA 104,104,141,5,3,104,141,4,3,1
04
190 DATA 141,11,3,104,141,10,3,104,104
,201
200 DATA 1,208,7,169,87,141,2,3,208,5
210 DATA 169,82,141,2,3,169,1,141,1,3
220 DATA 32,83,228,96
```

SECTOR TYPES — Atari DOS sets up five SECTOR TYPES on a disk. They are:

1. Boot Sectors
2. File Sectors
3. VTOC
4. Directory
5. Lost Sector

The Boot sectors are sectors 1, 2, and 3. They are read by the Operating System (in ROM) when the computer is cold started. If th' 'sk drive (D: device) responds, the boot sectors are read and processed. The boot sectors . .y load the DOS.SYS, AUTORUN.SYS, or some other program on the disk (if it is set up as an autoboot file).

File sectors are the major data carrying sectors on the disk. The are usually sectors 4 through 359 and 369 through 719. The last three bytes of a linked file sector contain information on the next sector, file number, and sector size. Data files usually have no

such information but are a full 128 bytes of data. The following information is contained in the last three bytes of a file sector:

Byte 125 - Next sector (MSB two bits) and File number (six bits).
Byte 126 - Next sector (LSB eight bits).
Byte 127 - No. of bytes in sector (seven bits) and.
Flag for short sector (one bit).

The VTOC or Volume Table Of Contents sector is sector number 360. This sector contains most of the organizational data for the disk. It contains a map (in binary code) of the disk which represents the active and inactive sectors. It also does the accounting to keep track of how many sectors are free, unused, and available. If you have a crash, power failure, or bug while writing to a file, you may not have a current VTOC. This can result in files being overwritten; the DOS may think some sectors are free because the VTOC is not current. The best thing to do in case of a crash is to stop using the disk immediately. Put a write protect tab on it and make a backup copy with a sector copier such as SCOPY 810. The symptom of a bad VTOC is an ERROR 164. The best way to correct the damage is to use a VTOC rebuilder such as the routine contained in DISKEY by Adventure International or Disk Fixer by APX. These utilities trace each file and rebuild the map in sector 360. Depending on the amount of damage, you may or may not have success. The VTOC sector is laid out as follows: (Of the 128 bytes in sector 360, only 100 are used.)

SECTOR 360 - VTOC

Byte 0	Flag for DOS 1 or DOS 2. 01=DOS 2, 00=DOS.1
Bytes 1 & 2	LSB and MSB for total number of free sectors with no files written on disk (707 nominally).
Bytes 3 & 4	LSB and MSB for the number of currently free sectors on the disk.
Bytes 5 - 9	Unused by VTOC.
Bytes 10-99	Bit map of occupied sectors. one bit per sector. 0=Unused, 1=Used.

Directory sectors are found in sectors 361 through 368. As files are added, the directory entries are added to this group. When you use the "A." option in DOS, you are reading the information in the directory sectors. Each directory entry uses 16 bytes and there can be a maximum of 64 files and directory entries in DOS 1 and DOS 2 disks. The directory is constructed as follows:

SECTORS 361 THROUGH 367 - DIRECTORY

Byte 0 FLAG byte (described below).
Bit 0 - Open file flag (0=closed, 1=open).
Bit 1 - Flag for DOS 2 (0=DOS 1, 1=DOS 2).
Bit 2 - ?
Bit 3 - Not used.
Bit 4 - Not used.

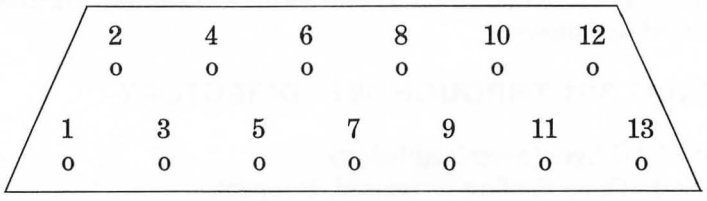
Bit 5 - Flag for locked file (1=Locked, 0=Unlocked).
Bit 6 - Valid file flag.
Bit 7 - Deleted file flag.
Bytes 1 & 2 - LSB and MSB for total number of sectors in file.
Bytes 3 & 4 - LSB and MSB for sector number of start of file.
Bytes 5 - 12- Filename.
Bytes 12- 15- Filename extender.

The lost sector is sector number 720. This sector is not used by DOS because the computer counts from 0 to 719. The DOS has numbered the sectors from 1 to 720. This means that you can only use sectors 1 to 719. Sector 720 is a good place to hide things; for instance, identification, error checks, secret messages, etc.

SELECT — The console key labeled SELECT can be monitored by the system by looking at bit 1 in location $D01F (53279 decimal). If bit 1 is zero, then the SELECT key is being pressed. The following values may be found by PEEKing location 53279 when the given combination of SELECT and other keys is pressed:

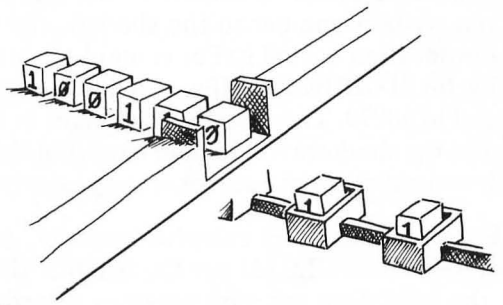
VALUE IN 53279		KEYS PRESSED
Decimal	Binary	
0	00000000	SELECT, START, and OPTION
1	00000001	SELECT and OPTION
4	00000100	SELECT and START
5	00000101	SELECT

SERIAL PORT — The Atari 400 and 800 computers use a SERIAL PORT to input from and output to peripheral devices such as disk drives, cassette recorders, and printers. The 850 interface is a device which attaches to the serial bus and provides a parallel output to drive most printers and RS 232 serial ports for modems and similar peripherals. The serial port operates asynchronously at 19,200 baud. Data is sent out in eight bit bytes with the lowest significant byte sent first. A start bit, which is a logical zero (0), is sent before each byte, and a stop bit, which is a logical one (1), is sent after each byte. A clock line goes high and the data line starts a new byte. Several other lines are used to control data and signals in the SERIAL PORT. Looking at the trapezoidal port of the computer, the pins appear as follows:



PIN	Function
1	CLOCK IN TO COMPUTER
2	CLOCK OUT FROM COMPUTER
3	DATA IN TO COMPUTER

4	GROUND
5	DATA OUT FROM COMPUTER
6	GROUND
7	COMMAND LINE goes low when a command frame is sent
8	MOTOR CONTROL for cassette
9	PROCEED
10	+5V READY also 5V, 50mA power supply
11	AUDIO IN from cassette
12	+12 Volt supply— (unknown current rating)
13	Interrupt (not used by present Operating System or peripherals)



SERIAL

SERIAL DATA TRANS — SERIAL DATA refers to the TRANSmission of data over a single communication line. Data is sent one bit at a time as opposed to multiple bits at a time as with parallel data transmission. See PARALLEL PORT.

SETCOLOR (SE.) — In BASIC, The SETCOLOR command is used to set the color registers in locations $02C4 through $02C8 (708 through 712 decimal). After the SETCOLOR command, you must specify the color register number, the hue, and the luminance. The color register can be 0,1,2,3, or 4. The hue must be a number between 0 and 15. The luminance must be an even number between 0 and 14.

COMMAND	PLAYFIELDS AFFECTED
SETCOLOR 0	Points or lines in graphics modes except 8 (COLOR 0) Uppercase letters in GR.1 and GR.2
SETCOLOR 1	Points and lines in GR.3, 5, 7 (COLOR 1) Lowercase letters in GR.1 and GR. 2 Luminance only in GR.0 and GR.8
SETCOLOR 2	Points and lines in GR. 3,5,7,8 (COLOR 2) Inverse uppercase letters in GR.1 and GR.2 Character hue and background color in GR.0 Background color in GR.8

SETCOLOR 3	Inverse lowercase characters in GR.1 and GR.2
SETCOLOR 4	Border color for all modes Background color for GR.1,2,3,4,5,6,7 Points and lines for GR.3,4,5,6,7 (COLOR 4)

SGN — The BASIC SGN command checks a number or variable to see if it is positive, negative, or zero. If it is negative, a –1 is returned. If it is positive, a 1 is returned. If the number is zero, a 0 is returned. The format for using SGN is: SGN(X) where X is a number or numeric variable.

SHADOWING — SHADOWING is the technique of using one register or memory location as an image of another location, usually in the top part of memory in the Operating System. If you write a number to the shadow, the same number will be transferred to the shadow location behind it. For example, color register 0 at location $02C4 (708) is the shadow for $D016 (53270). If you POKE 708 with the number 32, the 32 will also be transferred to 53270. This allows for changes in future revisions of the Operating System because the shadows will remain fixed, but the locations which they reflect may be changed, and software will still execute properly in a new OS revision.

SHORT IRG MODE — On the Atari cassette recorder, programs stored with CSAVE and programs loaded with CLOAD use the SHORT IRG (Inter-Record Gap) MODE. In this mode, the tape does not stop between records. Tokenized files are loaded directly into computer memory and are not "processed." The SHORT IRG MODE is characterized by a 0.25 second Pre-Record Write Tone and a variable length Post Record Gap of unknown frequency. The following chart specifies the cassette timings. (See FILE-CASSETTE, IRG, PRE-RECORD WRITE TONE).

Read IRG delay (Short mode)	.16 Seconds
Write IRG (Short mode)	.25 Seconds
Write IRC (Long mode)	3.00 Seconds
Read IRG DElay (Long mode)	2.00 Seconds
Write file leader	19.20 Seconds
Read Leader Delay	9.60 Seconds
Beep Cue Duration	.50 Seconds
Beep Cue Separation	.16 Seconds

SIG*ATARI — CompuServe has a Special Interest Group which is dedicated to Atari users. To get to the SIG*ATARI, type GO PCS-132 after you are logged on. This is a very good place to catch up on the latest rumors and news about Atari computers.

SIGNIFICANT DIGITS — This term refers to the number of places used by a number. The number 123 has three SIGNIFICANT DIGITS. The number 123.0000000 appears to have 10 SIGNIFICANT DIGITS, but it is wise to question the source of such a figure as it is very difficult to measure anything to 10 SIGNIFICANT DIGITS.

SIN — The BASIC trigonometric operator SIN returns the SINe of the number placed in parentheses to the right of the operator. The SINe will be calculated based on radians (unless the command DEG is first executed).

SINGLE LINE RESOLUTION — The players memory map is extended to 256 bytes long when SINGLE LINE RESOLUTION is specified. To specify SINGLE LINE RESOLUTION, set bit 4 (add 16) of register $022F (559). Each byte will map out as one eight-bit-wide scan line and the player may use up to 256 of them in SINGLE LINE RESOLUTION. Double Line Resolution will also use up to 256 lines, but the lines will be mapped from only 128 bytes, thus giving the player only half of the resolution. Double line resolution is the normal default situation.

SIO — Serial Input/Output. The SIO is the communication line between the computer, program recorder, disk drive, and 850 interface (printer and RS232). Operations are carried over the serial bus in the form of → Command frame, Acknowledgement from the peripheral, Data frame, and Complete signal. The Command frame is comprised of the device ID, command code, auxiliary bytes for device information, and checksum. After a Command frame is sent, the device should respond with an ACK for ACKnowledge. If the device is not operating, an ERROR 138 will result. After the checksum of the frame data, the device should respond with a Complete byte which is sent down the DATA IN line. The command codes can be for READ, WRITE, STATUS, PUT, FORMAT, READ ADDRESS, READ SPIN, MOTOR ON, and VERIFY SECTOR. See OS manual page 145.

SIXTEEN BIT MACHINE — While the Atari computers, the Apple, Commodore, Radio Shack, and most older personal home comuters are called eight bit machines, the new generation of high end personal or business computers are 16 bit machines. Instead of processing data in bytes which are eight bits wide, they are capable of handling data which is 16 bits wide. The advantages are easy to identify. Instead of having to process an address in two parts (high and low bytes), the whole address can be used in one operation. An eight bit machine is limited in most cases to a maximum addressable memory defined by two bytes or 64K. A 16 bit machine can address millions of bytes of memory by using 16 bits plus 4 more for an offset. IBM PC s are sometimes referred to as 16 bit machines.

SIXTY-FIVE - OH - TWO (6502) — The 6502 is the central microprocessor chip for the Atari computer line. This chip does all of the data processing. Data comes in from the keyboad, disk, joystick, tape, or modem and is used to produce some output to the screen, printer, disk, or cassette. The 6502 is an eight bit processor. This means that the data path which the chip processes is eight bits wide. The address bus or path is 16 bits wide. This means that the 6502 can address 2 to the 16th power or 65,536 bytes. This is the basis for the 64K of RAM. There is an Arithmetic Logic Unit which does all of the calculations in the 6502. These operations are done basically by adding, subtracting, and shifting 1s and 0s via the assembly language instructions. A clock running at 1.79 MegaHertz (cycles per second) sets the pace for these instructions and operations. Register A (for Accumulator) is the most often used input and output register of the 6502. Other registers called X, Y, and S (for Stack) are used to execute programs. Bytes or numbers are loaded in from memory (or through memory) and all programs are processed by the central processor.

SLOW LISTER — If you do not have a printer on line and are trying to debug a lengthy BASIC program, you may have had troubles stopping or reading from the screen. This program will help by allowing you to scroll slowly through the listing.

Type in this listing and LIST it to disk under the name "D:SLOLIST.LST". Then when you have your program in memory enter the SLOLIST program by typing ENTER "D:SLOLIST". Type GOTO 30000 to start SLOLISTing. S (for Slow) will slow the listing speed and F (for Fast) will speed it up. (From Portland Atari Club newsletter, Nov., 1982.)

```
1 ? CHR$(125):? "Hit S to slow listing."
:? "Hit F to speed up listing.":STOP
30000 X=PEEK(136)+256*PEEK(137):POKE 764
,255:WAIT=50
30001 LINE=PEEK(X)+256*PEEK(X+1):X=X+PEE
K(X+2):IF LINE>29999 THEN STOP
30002 LIST LINE:FOR TIME=1 TO WAIT:NEXT
TIME:IF PEEK(764)=62 THEN WAIT=WAIT+20
30003 IF PEEK(764)=56 THEN WAIT=WAIT-20:
IF WAIT<0 THEN WAIT=0
30004 POKE 764,255:GOTO 30001
```

SOFTSIDE — Softside is a magazine oriented toward home computer users who enjoy games and light applications. A disk version is produced each month. Softside Publications, 6 South Street, Milford, NH 03055 603-673-0585.

SOFT SECTORED DISK — Atari Disk drives use SOFT SECTORED DISKs. This means that the sectors are divided by timing marks written on the medium during the formatting process as opposed to using holes punched in the medium. Atari drives can use hard sectored disks, but they will be used as soft sectored.

SOUND (SO.) — In BASIC, the SOUND command is used to turn on the four voices in the Atari computers. Since the tones generated are fairly square wave sounds, no high quality music is really possible using SOUND commands in BASIC. The structure of the SOUND command is:

SOUND V, P, D, L where:
V is the voice from 0 to 3.
P is the pitch arguement from 0 to 255.
D is the distortion from 0 to 14 (even numbers only).
L is the loudness from 0 to 15.

SOURCE CODE — Most programmers (but not all) who produce machine language programs use assembly language to write the instructions for the microprocessor. The listing which is produced in assembly language is referred to as the SOURCE CODE. A machine language program or routine can be "reverse engineered" to the assembly language instructions through a disassembler program. The reconstituted SOURCE CODE is not usually true SOURCE CODE because it often lacks labels and comments and is very difficult to follow. Ultra Disassembler from Adventure International will label the source from object files or memory.

SOURCE DISK — In copying or duplicating disks, the SOURCE DISK is the original which you wish to duplicate. The destination disk is the blank or formatted disk which will become the copy.

SPACE — In serial telecommunications a SPACE is a high voltage or a low frequency signal and it translates to a logical zero (0). The mark tone is a high frequency signal which is interpreted as a logical one (1). In modem communications, the mark and SPACE are generated by shifting the frequencies which are transmitted over the phone line. In the originate mode, the SPACE is a 1070 Hz tone. In the answer mode, the SPACE is a 2025 Hz tone.

SPEAKER — The console SPEAKER is controlled by register $D01F (53279 decimal). This is the same location as for the console keys. To start the SPEAKER clicking, POKE in a number between 0 and 7. The continuous loop

```
1 POKE 53279,Ø: GOTO 1
```

will generate a continuous humming noise. In the XL series, the SPEAKER noise is routed to the television SPEAKER.

SPELL WIZARD — Spell Wizard, a utility for identifying and correcting misspelled words in Atari DOS text files, was created by Datasoft, Inc. for Atari Writer and Text Wizard word processors. Spell Wizard is menu driven and is very easy to use. When you run the program, the document is read and the total number of words as well as the number of unique words is displayed. Only the unique words are checked. This usually ends up as a number between 300 and 700. The use of a unique word list reduces the amount of time required to process a document. The unique words are compared to a 33,000 word dictionary. This contains over 90 percent of the words in most non-technical reports. The proofing operation allows you to leave a word alone if it is tagged or you may search through the user dictionary for a match. You can make your own dictionary, for example, of orthodontic terms, ceramic engineering terms, celestial mechanics terms, etc. You can do a wild card search for the correct spelling of a word if you know that part of it is correct. This is done using an asterisk for the wild part. Spell Wizard recognizes all capitalized letters as lowercase for the purposes of spelling checking. Datasoft, Inc.

SPOOLER — A SPOOLER is a buffer for temporarily storing data before it is sent to a final input or output device. It is the same as a printer buffer. A printer SPOOLER is a handy tool which can accept a file that may be headed to a printer at a much faster rate than the printer, freeing your computer for other activities. A buffer may be 16K up to 48K in size. The ATR8000 and MicroMainframe systems have built in 16K printer SPOOLERs which may be expanded to 64K.

SQR — This BASIC operator returns the SQuare Root of a number supplied in parentheses after the command.

STACK — The hardware STACK is a block of memory 256 bytes long, located in page 1 (between $0100 and $01FF). The hardware STACK is used for storing parameters from interrupt and subroutine branches. Parameters are put on the STACK from the

top ($01FF) and are pushed downward every time a new one is added. If more than 256 bytes are added, the entries will wraparound, causing all kinds of trouble. When a program sees a JSR (Jump to SubRoutine) or a PHA (PusH Accumulator on STACK), it leaves an address contained in the program counter on the STACK. When the return command (RTS from a Subroutine or RTI from an Interrupt) is found, the address is pulled off the STACK and put back in the program counter. The PLA code also reads (Pulls the Accumulator) off of the STACK. The processor status register contents can be pushed onto the STACK with a PHP, or the processor status register can be filled by a PLP operation (PulL Processor status from STACK).

Another type of STACK is used by BASIC. This is the RUN TIME STACK which keeps track of the BASIC line number where the program must return from a GOSUB command. The RUN TIME STACK is pointed to by $008E and $008F (142 and 143 decimal).

START — The START key is a console button whose status is monitored in the CONSOL register at $D01F (53279). The following values will appear in 53279 when the START key and/or other console keys are pressed.

VALUE IN 53279		CONSOLE KEYS PRESSED
DEC	Binary	
0	00000000	START, OPTION, AND SELECT
2	00000010	START AND OPTION
4	00000100	START AND SELECT
6	00000110	START

STAR RAIDERS — Atari's space game on cartridge which combines fast space ship fighting action with a board game. The TV screen is used as a window of a well armed space fighter. This game was resposible for the sale of many thousands of Atari 400s and 800s. This game was written by Doug Neubauer who also designed the POKEY chip.

STATEMENT — A STATEMENT in BASIC is a complete logical line or sentence which is part of a BASIC program. STATEMENTs which are tokenized are stored in a STATEMENT table. In a LISTed program, STATEMENTs are separated by a colon.

STATEMENT TABLE — A BASIC program which has been transferred or inputted into Atari memory exists in a block of tokenized code called the STATEMENT TABLE. Accompanying this table is the Variable Name Table and the Variable Value Table. The tokenized BASIC is a series of codes which can be executed by the computer through the BASIC cartridge program.The STATEMENT TABLE can be found by looking up its address in locations $0088 and $0089 (136 and 137 decimal). Any immediate mode statements which have been tokenized are also stored in the STATEMENT TABLE.

STATUS — STATUS is a BASIC command which will return a code generated by the last input or output through a specified channel. The value returned is put into a variable which is defined by the user. An example is:

```
STATUS #1, ERROR
```

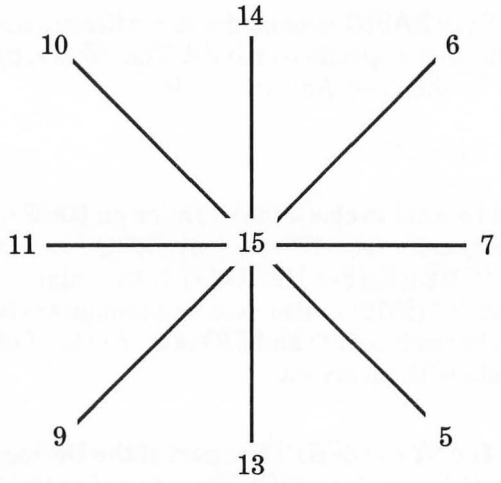
This statement would be used to check the situation on IOCB (channel) 1. If the value assigned to ERROR is greater than 128, then an ERROR has occurred. The value represents the ERROR NUMBER. (See ERRORS). If the value is 1, 2, or 3 then no error has occurred. Location 793 ($319) is also used as a temporary buffer for STATUS and this location may be checked. A CIO and SIO status byte of 001 means that the last operation was complete with no errors.

STATUS BYTE — The STATUS BYTE is part of the Device Control Block used for transferring data over the serial bus (SIO). The Device Control Block is a group of four bytes located at $0300, $0301, $0302, and $0303. The last one, $0303, is the STATUS BYTE. After a call is made to transfer data over the SIO, a status code is put into the STATUS BYTE address. This is a one-byte code to indicate what has happened after the transfer. There are seven possible values for a STATUS BYTE:

HEX	DEC	MEANING
$01	001	Operation completed and no errors
$8A	138	Device Timeout- No response from device
$8B	139	Device NAK- Negative AcKnowledge
$8C	140	Serial Bus Input Framing Error
$8E	141	Serial Bus Data frame overrun error
$8F	142	Serial Bus Data frame checksum error
$90	144	Device Done Error

STEP — STEP is used in BASIC to increment a FOR/NEXT loop by a number other than 1. A FOR/NEXT loop will normally increment by 1 beginning with the number assigned after the FOR and continue looping up to the number after the TO. By using a STEP, you can increment (or decrement) by larger or smaller units. For example, FOR X=0 to 100 STEP 5 will count 0, 5, 10, 15 . . . up to 100. STEP can also be used with a negative argument.

STICK — In Atari BASIC, STICK(X) reads the register assigned to the joySTICK ports (where X can be from 0 to 3 for ports 1 to 4). There are only two ports on the XL series computers. PORTB, which handled 3 and 4, is used for the Operating System controls. (See PORTB). The joystick registers are located from $0278 to $027B (632 to 635). These are shadowed from the PIA registers $D300 and $D301 (54016 and 54017). For each of the four STICK registers, there are nine values which can be read. These are generated as the joystick is moved to open or close switches inside the mechanism. The values generated are as follows:



VALUES GENERATED BY MOVING JOYSTICKS
READ BY USING STICK(X) WHERE X= 0,1,2, or 3

STOP— When a BASIC program encounters a STOP command, the program will halt execution at the line where the STOP appears. CONT will CONTinue the program from that location.

STOP LISTING — You can temporarily STOP LISTING a BASIC program or directory listing from DOS by simultaneously pressing the CTRL and 1 keys. The following routine will allow you to use the START key to both list and STOP LISTING your BASIC program. The program fits in Page 6 and will remain there until you shut down or overwrite the program.

```
10 REM ** ABCS OF ATARI COMPUTERS
20 REM ** FILE D:STOPLIST.BAS
30 REM ** STOP LISTING WITH START KEY
40 REM
50 FOR X=1536 TO 1791:READ A:POKE X,A:
NEXT X
60 DATA 0,0,169,1,44,31,208,240,27,173
,1,6,201,1,208,17,169,0,141,1,6,173,0,
6,201,1,240,5,169,0,141,255,2,76,95
70 DATA 228,173,1,6,201,1,208,15,169,2
,44,31,208,208,5,169,1,141,0,6,76,95,2
28,169,1,141,1,6,173,0,6,201,1,240
80 DATA 8,169,1,141,255,2,76,95,228,16
9,0,141,0,6,76,95,228,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
90 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,72,138
100 DATA 72,152,72,169,0,141,0,6,141,1
,6,162,6,160,2,169,6,32,92,228,104,168
,104,170,104,104,96,0,0,0,0,0,0,0,0
110 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
120 DATA 0,0,0,0,0,0,0,0
130 T=USR(1696)
```

**STR$** — In BASIC, the STR$ command converts a number or numeric variable to a STRing. You will also need a string variable name to place the string into if you want to use it somewhere else. ONE23$=STR$(123) will assign the string "123" to the string variable ONE23$.

STRIG — In Atari BASIC, STRIG(X) looks at the joystick triggers for the sticks connected to ports X=0,1,2, or 3. If the trigger is pressed when the STRIG command is executed, the value 0 will be returned. If the trigger is not pressed, a 1 will be returned. The registers for STICK are found from locations $0284 to $0287 (644 to 647). These are shadows for locations $D010 to $D013 (53264 to 53267).

STRING — A STRING is a collection of bytes. A STRING has no numerical value even if it looks like a number. On your screen, a STRING will appear as a line of ATASCII CHARACTERS. In memory a STRING is stored as a series of hexadecimal bytes. SEE STRING VARIABLE.

STRING VARIABLE — A STRING VARIABLE is one of the three types of variables in Atari BASIC. A STRING VARIABLE must be declared or DIMmed in Atari BASIC before it can be used in a program. This is not necessary in Microsoft BASIC or in BASIC XL if the string is less than 40 characters long. All STRING VARIABLE names must end with a dollar sign ($). When a STRING VARIABLE is DIMmed, BASIC reserves one byte for each character in the DIM command. For Example, DIM X$(199) will reserve 199 bytes of memory in the string table just for the string X$. A quick way to clear a long string is to use the routine: A$(1)="":A$(2)=A$(1). A STRING VARIABLE can be assigned the ATASCII bytes corresponding to the machine language instructions for a short program. The string will be located somewhere in memory and its address can be found by the command ADR(STRING$). By jumping to the address found (use a USR), the machine language program can be run. Unless there is a proper exit back to BASIC, you will lose control after the jump.

Strings can be joined together or be broken into substrings. See CONCATENATION and SUBSTRING.

STRING/ARRAY AREA — The STRING/ARRAY AREA or table is the part of Atari memory which contains the actual data for the strings and arrays which have been assigned. When a program comes across a string or array variable, it checks the VARIABLE NAME TABLE (Start of VNT=PEEK(132)+PEEK(133)*256) to see if the name is registered and to get the number which was assigned when the name was entered. Then BASIC goes to the VARIABLE VALUE TABLE (Start of VVT= PEEK(134)+PEEK(135)*256) with the number it found in the VNT and it looks up where it must go to find the STRING/ARRAY AREA. The start of the STRING/ ARRAY AREA is found by looking in locations $008C and $008D (140 and 141). The variable STARP = PEEK(140)+PEEK(141)*256 will take on the start of the STRING/ARRAY AREA. Individual strings or arrays are found by looking at memory at an offset from the start. The offset is also stored in the VVT. See VARIABLE VALUE TABLE.

SUBROUTINE — A SUBROUTINE is a program, usually shorter than the main program in which it is contained, which performs some repetitive utility. If a utility is not needed more than one time, you are better off not placing it in a SUBROUTINE. A SUBROUTINE is entered by a GOSUB NN where NN is the line number of the start of the subroutine. The end of every subroutine must have a RETURN to get back to the main program.

SUBSCRIPT — A SUBSCRIPT is a character, usually in smaller print, which is printed lower than the main line on which it is printed. The formula for water, H_20, should have a SUBSCRIPTed 2. Many newer dot matrix printers can handle subscripts when they are given the proper commands.

SUBSCRIPT — also refers to the element number of an Atari BASIC array. An array can have one or two dimensions and thus will require one or two subscripts. String arrays cannot be subscripted in Atari BASIC but they can be subscripted in Microsoft BASIC.

SUBSTRING — A string can be broken into a shorter SUBSTRING by supplying the information on which characters are supposed to be the first and last of the new string. If a string already has an assignment, a SUBSTRING can be created by assigning the characters to a new string name. If A$="ONEBIGSTRING", the SUBSTRING B$ can be assigned "BIG" by the command B$=A$(4,6) where the numbers in parentheses are the first and last characters of the SUBSTRING to be copied. In Atari BASIC, SUBSTRINGS can be used to simulate string arrays in Atari BASIC.

SUPERSCRIPT — A SUPERSCRIPT is a character, usually printed in smaller print, which appears above the center of the main line of text in which it appears. The expression $E=I^2R$ includes a SUPERSCRIPTed 2.

SYNC MARK — The SYNC MARK is a "space tone" marker for audio tracks on cassettes. It is a 3,995 Hz tone.

SYNTAX — SYNTAX is to the BASIC language what grammar is to the English language. If you make a mistake in syntax, you may not be understood properly. Atari BASIC checks the SYNTAX of every statement entered when the RETURN key is pressed. If there are any syntactical errors, you are notified of the location on the next line. Microsoft BASIC does not check SYNTAX in this way.

SYSTEM DATABASE — Pages 0 through 4 are known as the SYSTEM DATABASE. This part of memory is used exclusively by the Operating System for its vectors, stacks, buffers and registers. There is very little usable free RAM in this area.

SYSTEM RESET — Pressing the SYSTEM RESET button enables an Operating System routine which is entered through location $F11B (61723). The SYSTEM RESET initialization is similar to a warmstart. OS RAM in Pages 2 and 3 is zeroed. RAM between $0010 and $007F (16 and 127) is also zeroed. User RAM is not erased. DOS is not booted unless an initialization address is provided in DOSINI at locations $000C and $000D (12 and 13). See SYSTEM RESET CHANGER to change the action of the SYSTEM RESET key.

SYSTEM RESET CHANGER — This BASIC program will POKE in a machine language routine which will reset the disk boot pointer to a new program that essentially types "RUN" when you push SYSTEM RESET. This is easy to do for machine language programs, but is not so clear for BASIC programs. To make machine language programs restart, put the initialization address in locations 12 and 13 ($0C and $0D). SYSTEM RESET will just start the program over. To reset and RUN a BASIC program, type or load in this routine (it goes in page 6). Then LOAD your BASIC program. Type POKE 12,0 and POKE 13,6 to run the program when SYSTEM RESET is pressed. You can put the POKEs in the program if you are not going to have to access the disk drive in the program. From Novatari, February, 1983.

```
1 REM ** ABCS OF ATARI COMPUTERS
2 REM ** FILE: CHNGSYS.BAS
3 REM **
10 FOR B=1536 TO 1590:READ A:POKE B,A:
NEXT B
20 DATA 162,0,142,68,2,232,134,9,173,4
8,2,133,203,173
30 DATA 49,2,133,204,160,4,177,203,133
,205,200,177,203
40 DATA 133,206,162,0,160,82,189,52,6,
145,205,232,200,224
50 DATA 3,208,245,169,12,141,252,2,108
,250,191,50,53,46
55 LIST 60,70
60 REM ** BE SURE TO POKE 12,0 AND
70 REM ** POKE 13,6 AFTER TYPING RUN
```

T

TAB — In BASIC XL, the TAB statement is used to insert a certain number of blank spaces. TAB can be used on the screen to tabulate a number of spaces, or it may be used to insert blanks into a cassette or disk buffer. The format is:

```
TAB #iocb,space
```

where iocb is the channel in which you want to insert blanks, and space is the number of blanks you want to insert.

TAN — In BASIC, the function TAN returns the trigonometric function tangent of the operand. The format is TAN(X), where X is the number of degrees or radians.

TANDON — The fast growing maker of disk drives, Tandon, now makes some of the mechanisms for the the Atari 810 and 1050 disk drives. These units are different from the early model made by Micro Peripherals, Inc. The manufacturer can be determined by looking at the door mechanism. If it swings up, it is a Tandon. If it slides straight up, it is probably a MPI.

TAPE PROBLEMS — The Atari 410 program recorder is the cause of many complaints by users. Some problems arise because of head misalignment. Programs recorded on one tape deck may not line up for the maximum signal when reading on another recorder. Adjustments may make the problem worse if you lose the ability to read all of the cassettes in your library. Prudence is suggested in realigning the heads.

Other problems may arise from leaving the recorder in the PLAY or RECORD position for lengthy periods. The capstan presses against the rubber roller and may leave a long-term depression in the roller. This will give erratic results during loading or saving of programs.

A software problem with an easy fix is the emptying of the cassette buffer. By typing LPRINT after you CSAVE a program, the cassette buffer is emptied. An ERROR message will be generated, but it is not relevant. By emptying the buffer, you will reduce the chances of a scrambled file during input or output.

TERMINAL PROGRAM — A TERMINAL PROGRAM is a software package which lets your computer act as a computer terminal. That is, with a modem and the proper interface, you can communicate over a standard telephone line with other computers (and people). There are several fine public domain terminal programs and some excellent commercial packages. Jonesterm and AMODEM are in the public domain. (See AMODEM). Syn-Comm from Synapse, Teletalk by Datasoft, Teletalk by Tronix, and T.H.E. Smart Terminal and Chameleon by APX are some commercial packages.

TERMINATOR—A TERMINATOR is a symbol or marker which is used to turn off or denote the end of something. For example, a TERMINATOR would be used to end the underlining of text. In Letter Perfect, a CTRL-D is used as a delimiter or TERMINATOR for text moves or deletes.

TEXT WIZARD — This text processor program is one of the most popular used on Atari Home Computers. Files are produced under Atari DOS, and any LISTed BASIC or other file can be edited with TEXT WIZARD.Text Wizard leaves more free RAM than the Atari Writer. Only the Atari 825 (Centronics 737) and the Epson MX-80 printers are supported directly by TEXT WIZARD. Printer setup commands are limited to a few built-in codes such as CTRL+E for elongated print. There is no way to send custom commands (such as for subscripting) on the new Epson printers. This is a severe limitation. The program is not menu driven for the save, print, edit, and load routines as are the Letter Perfect and Atari Word Processors. Files to be printed must be printed in entirety, as there is no facility for printing specific pages or multiple copies. You can print from a certain page from the middle to the end of the document, but the printer control codes will not be sent. Search, move, and replace features make the program very handy for editing long documents or programs. TEXT WIZARD may be the best way to debug very long BASIC programs with difficult to read code. Datasoft Inc.

THEN — This BASIC word is the second part of the IF/THEN branching command. THEN specifies what action is to be taken if the IF condition is true.

**TIME$** — In MICROSOFT BASIC II, the TIME$ statement returns a string with the time expressed as HH:MM:SS. You can set the time by using a BASIC statement TIME$="HH:MM:SS". This clock routine is accurate to within 90 seconds per 24 hours.

TIMERS — There are numerous timers built into the Atari computer. They are used to synchronize operations in hardware and software. The Real Time Clock and Attract Mode TIMERS are two which are readily accessible. The Real Time Clock is in three

bytes located at 18, 19, and 20 ($12, $13, and $14). Location 20 is incremented by 1 every 60th of a second. When the value reaches 255, location 19 is incremented and location 20 starts at 0 again. When location 19 reaches 255, location 18 is incremented.

TIMER LOCATION		
20 ($14)	19 ($13)	18 ($12)
0-255	0-255	0-255

INCREMENTED		
Every 65536 VBLANKS	Every 255 VBLANKS	Every VBLANK

The incrementation occurs every VBLANK which is approximately every 60th of a second. Actually it is every 59.92 seconds. You can build a clock using this information. The clock will lose time, however, during disk access. See Antic, Vol.1, No.4 for a real time clock placed safely in the cassette buffer.

The attract mode TIMER is in location 77 ($4D). This register is incremented every 255 VBLANKs, or approximately every 4.255 seconds. When the value in location 77 reaches 127 ($7F), the color and luminance rotation of the screen begins. A 254 is placed in location 77 until a key is pressed or the value is reset away from 254. Pressing the trigger will not reset this register to zero. Hitting any key on the keyboard will. You can experiment with the Attract mode TIMER with the following program. First set the value of 77 to some number, say 120, so you do not have to wait so long for the rotation of colors to start.

```
1 POKE 77,120
2 PRINT PEEK(77):GOTO 2:REM ** NOW HIT A
 KEY
```

TOKEN — A TOKEN is a byte which is used by the BASIC interpreter (on cartridge) to represent part of the BASIC program. BASIC statements are TOKENized during RUNning, LOADing, or in immediate mode. See TOKENIZING.

TOKENIZING — The TOKENIZING process takes a fairly simple BASIC program and converts it to code which is more readable to the computer. The process is roughly:

a. Get a line from program or immediate mode
b. Check syntax (FOR must have TO, etc.)
c. Create tokens during syntax check
d. Move tokens to statement table
e. Execute if in immediate mode (no line number)

The BASIC cartridge actually contains a program which is BASIC. This program gets the statements to TOKENIZE from the screen, E:, disk, or cassette. The Operating System takes care of getting the statements. As the data is received, it goes into a

BASIC Input Line Buffer located at $580 to $5FF. The first check is for a line number. If one is found, it is converted to a two byte integer. These two bytes are the first two in the TOKENIZED statement. If no line number is found, the statement is assumed to be in the immediate mode and an $8000 is assigned to these positions. The next two bytes cannot be completed until the entire line is TOKENIZED. The length of the line is counted (in bytes) and the number (in hex) is put in the third position. Next, the length from the start of the line to the start of the next statement is calculated. There may be many statements per line and they will be separated by a colon. At this time there are four bytes in the TOKENIZED statement.

BYTE #	1 2	3	4
	04 00	09	05
	Line 4	9 bytes for line	5 bytes for stmt

After the housekeeping bytes are produced, BASIC begins looking at the statements and checking syntax. Words are checked against a Command table in ROM which has all valid BASIC commands. If a match is made then a code is assigned for that operation. For example, REM is $00, DATA is $01, INPUT is $02, COLOR is $03, etc. If no match is found, then the TOKENIZING is aborted and an ERROR at a certain LINE number is printed. After a successful command is found, the next item must be one of seven possibilities: A variable, constant, operator, function, quotation mark (") , another statement, or an EOL character. The next character is checked to see if it is a number (numeric character). If the test is negative, then BASIC goes to the Variable Name Table to see if the characters that follow comprise a variable. If this test is negative, then BASIC goes to the Operators and Functions tables to see if the characters form a function or operator (like +, –, > or USR, ADR, EXP, etc.). If this test is negative, it is assumed that the entry is a new variable. In this case, an entry is made in the Variable Name Table. Along with this entry, eight bytes in the Variable Value Table are set aside. The statement FOR X=23 to 88 would be TOKENIZED as follows:

08 80 2D 0E 40 23 00 00 00 00 19 0E 40 88 00 00 00 00

08 = FOR
80 = X
2D = EQUAL SIGN OPERATOR
0E = Numeric constant
40
23
00
00
00
00
} SIX BYTES FOR BINARY CODED DECIMAL 23
19 = TO
0E = Numeric Constant
40
88
00
00
00
00
} SIX BYTES FOR BINARY CODED DECIMAL 88

A $0F and a dummy byte for the string length are written if quotation marks are found (in BASIC, the characters enclosed within the quotes are recognized as a string). When the length is determined, the length is written to the dummy byte. Since the maximum value that can be stored in a byte is 255, the limit to string length for use in BASIC programs is 255 characters.

An End of Statement character, $14, is placed in the TOKENIZED program every time a colon is encountered. The offset from the beginning of the line to the next statement is placed in the dummy byte following the $14. At the end of the line of statements, a $16 or End Of Line character is generated.

The TOKENIZED BASIC program can be unTOKENIZED by looking at the sector of a disk containing the SAVEd BASIC program and decoding the bytes by using the COMMANDS, OPERATORS, and FUNCTIONS table of tokens. This should never be necessary except in case of disk damage to a very valuable and irreplaceable BASIC disk based program. See SAVE and VARIABLE NAME TABLE.

TOP OF FORM — A printer control code for TOP OF FORM will cause line feeds to feed paper until the point where the page begins is met. The TOP OF FORM is set when the printer is turned on. This point is normally set at the perforations on form feed paper. An ASCII 12 or CTRL-L usually generates a TOP OF FORM command.

TRACE — In Microsoft BASIC, the TRON command turns on the TRACE mode. TRON is available only on the extension disk. As the program executes, the TRACE mode prints each line number on the screen as it is encountered. TRACE is turned off by TROFF. The command is called TRACE in BASIC XL, and TRACEOFF turns off the TRACE function.

TRACK — The concentric rings of data on a disk which contain formatted sections of polarized magnetic domains are called TRACKs. An Atari DOS disk is set up with 40 TRACKs. The spacing between TRACKs on Atari disks is very large by industry standards. This means that very low quality disks can usually be used without failure due to dropouts or head misplacement. You can also usually format and use the backside of almost any floppy disk without problems. Many disks use 96 TRACKs per inch, while the Atari 810 disk drive uses less than half this density.

TRACKBALL — A TRACKBALL is a game controller which acts like a joystick except that it is controlled by a rolling billiard sized ball. A trackball usually works by sending digital pulses into the joystick ports. This occurs as a beam of light inside the mechanism is interrupted by a spinning slotted disk.

TRAILING ZEROS — TRAILING ZEROS on the right-hand side of the decimal point are dropped by Atari BASIC. In order to force TRAILING ZEROS to appear, you can try two things. First, use Microsoft BASIC with the PRINT USING format. This is on the extension disk. Second, you can convert your numbers to a string for presentation in Atari BASIC. This is a complex technique.

TRANSLATION — When using a terminal program such as Jonesterm, you are often asked whether you want TRANSLATION. TRANSLATION does not affect the way data are received and stored by your computer. It does affect the way you see the data if you sent the data to the screen for viewing. The reason that TRANSLATION is necessary is that many of the control characters perform editing functions on the screen and these may also be part of some machine language code. If you are receiving data from a computer sending ASCII code, your Atari will not be able to respond to a carriage return, BELL, or backspace without TRANSLATION. The translator in Jonesterm does this:

ASCII 13 (CR) is translated to ATASCII 155 (EOL)<Return-inverse>
ASCII 07 (BEL) is translated to ATASCII 253 Bell <Esc-CTRL-2>
ASCII 08 (BS) is translated to ATASCII 126 Delete/Backspace<ESC-BackSpace>

Other characters can be translated if necessary.

The 850 interface module is also capable of doing TRANSLATION. Atari computers use bytes which are capable of representing a 256 element character set. This includes inverse (high bit set), uppercase, and lowercase characters. Not all computers handle this number of characters. Some systems use only seven bit words which usually means there is no inverse set. Other systems may use six bit words, eliminating the lowercase characters. The 850 will make eight bit words (bytes) out of the incoming data if you set it up correctly. It will also translate the End Of Line, Carriage Return, and Line Feed characters. The 850 interface can be set up in a No TRANSLATION, Light TRANSLATION, or Heavy TRANSLATION mode. No TRANSLATION is used for talking between two Atari computers and all characters are sent and received as is. In Heavy or Light TRANSLATION, some characters are transformed. The ATASCII EOL character (155 decimal, $9B hex or inverse-RETURN) is converted to ASCII CR (13 decimal, $0D hex, or CTRL-M) in both Heavy and Light modes. There is an option available to append a LF (line feed) to the CR translated character. This option is activated by adding a 64 to the first aux argument in the XIO 38 command.

Light TRANSLATION strips away the high bit. That is, it removes the inverse characters for input and changes the CR to an EOL. Output characters are left in inverse and the EOL is changed to a CR. All other characters are left alone.

Heavy TRANSLATION effects more changes. The high bit is again stripped away as in Light TRANSLATION and the CR is changed to an EOL on input. Only characters which are the same in both ASCII and ATASCII are transmitted. These characters are numbered 32 to 124 ($20 to $7C). Control characters (those below 32) and graphics characters (those above 124) are set to a "won't translate" character on input. This is nominally a 0, but can be set to any character. For output, no character will be sent if it is not translatable.

In BASIC, TRANSLATION is set up by the XIO 38 command according to the following format:

```
XIO 38, #chan, aux1, aux2, "R:"
```

where # chan must be an OPEN channel between 1 and 5, and aux1 is a number between 0 and 255. Add the following increments to get appropriate features.

aux1	Additions
0	Light TRANSLATION
16	Heavy TRANSLATION
32	No TRANSLATION
64	Append LF after CR

Aux2 is also a number between 0 and 255. This is where the "won't translate" character is found. The ATASCII equivalent of the "won't translate" character is put in this argument.

ATASCII	No TRANS.	Light TRANS.	Heavy TRANS.
EOL 155 ($9B)	EOL 155 ($9B)	CR 13 ($0D)	CR 13 ($0D)
EOL 155 ($9B)	EOL 155 ($9B)	CR + LF *	CR + LF *
0 to 255	SAME	0 to 127 only	32 to 124 same

* With Line feed Append Option in the XIO 38 command.

TRANSLATOR — There are several changes in the XL series operating system which makes certain programs crash if the programmer violates the documented rules provided by Atari. Atari has provided a TRANSLATOR disk which essentially replaces the XL Operating System (600XL, 800XL and 1200XL) with the Revision B OS from the 400 and 800. This is possible because of the special register which allows you to disable all of the OS ROM and use it as RAM. See PORT B.

There are two versions, one being a more thorough translation than the other. To use the TRANSLATOR, you must boot in the disk provided and wait for the prompt. If you do not need BASIC to run the program, you must hold down the OPTION key when you boot the TRANSLATOR and the cartridge will be de-selected. When you get the prompt, put the disk you want to load in the drive and press SELECT. The computer should do a coldstart and run the program.

TRAP — TRAP is an ERROR handling device in Atari BASIC. TRAP sets up a specified line number as the place to which program control goes when an ERROR (any ERROR) occurs. For example, if you place the statement TRAP 100 in a BASIC program and the program tries to print while your printer is turned off, your program will jump to line number 100. TRAP must be undone by the command TRAP 40000 (Actually, any number between 32768 and 65535 will do.) Merely hitting BREAK will not reset a TRAP. Debugging with a TRAP set can be confusing, because the TRAPs stay set until you release them. You must set the TRAP before the line where you expect the ERROR.

TRIGGER, JOYSTICK — The red button on the Atari joystick is called the TRIGGER. The TRIGGER is actually a switch which is connected through pin 6 of the ports on the front of the 400 and 800 models (or on the side of the XL models). When the TRIGGER button is pressed, the switch closes and it can be used to activate some activity in a

software routine. Pin #6 (the orange wire) of each of the joysticks is connected directly to hardware registers called TRIG0 through TRIG3 ($D010 through $D013 or 53264 through 53267). The ports are numbered 0 through 3. In Atari BASIC, the TRIGGER register can be scanned by using a STRIG statement. If the circuit is open the hardware register will contain a 1. If the TRIGGER is depressed (the switch is closed) the register will contain a 0. The format of a test in BASIC on joystick 0 would be:

```
IF STRIG(Ø) = Ø THEN 1ØØ
```

If the button is pushed, then line 100 will be executed next. If the button is not pushed, the next statement will be executed.

Another way to read the TRIGGER register is to PEEK locations 53264 through 53267 ($D010 through $D013). In BASIC, the button can be checked like this:

```
IF PEEK(53264) = Ø THEN 1ØØ
```

To examine this register in real time, try the folowing short program. Plug the joystick in port 1 and press the button.

```
1 PRINT PEEK(53264):GOTO 1
```

This is equivalent to the STRIG statement above. To check other joystick buttons, use the registers 53265 through 53267. In assembly language, the contents of the registers can be checked for a 1 or 0. These registers should be scanned often to make sure the activity (shooting) reacts quickly to the TRIGGER.

TABLE

Register	Address (HEX)	Address (DEC)	OS Shadow (HEX)	OS Shadow (DEC)
STRIG 0	D010	53264	284	644
STRIG 1	D011	53265	285	645
STRIG 2	D012	53266	286	646
STRIG 3	D013	53267	287	647

By setting bit 2 (adding 4) to register #D01D (53277), you can latch the joystick trigger. This means that instead of the TRIG registers resetting to 1 when the button is released, the 0 remains after the trigger button is pressed until bit 2 of 53277 is set to zero.

TRIGGER, PADDLE — The PADDLE TRIGGERs are similar in operation to the joystick triggers. The paddles come in pairs and each pair shares a controller port. The BASIC command for monitoring the paddle trigger is PTRIG(X) where X is a number from 0 through 7. When a PADDLE TRIGGER is pressed the PTRIG statement returns a 0. When a trigger is not pressed the statement returns a 1. The hardware registers are a bit more complicated than the joystick trigger registers. Location 54016 ($D300) contains the registers for PADDLE TRIGGERs 0,1,2 and 3. Location 54017 handles the PADDLE TRIGGERs for 4,5,6 and 7. When no PADDLE TRIGGERs are

pressed, location 54016 contains a 255. Each trigger controls one bit in this byte so the values change with each trigger press. The bits are controlled as follows:

		PORT A		BYTE 54016 ($D300)				
D7	D6	D5	D4	D3	D2	D1	D0	BIT
3	2			1	0			PADDLE TRIGGER

The following values are found when various triggers are pressed.

No triggers	255
Trigger 0	251
Trigger 1	247
Trigger 2	191
Trigger 3	127

The same values are found in register 54017 for PADDLE TRIGGERs 4 through 7. RUN the following program to examine these registers:

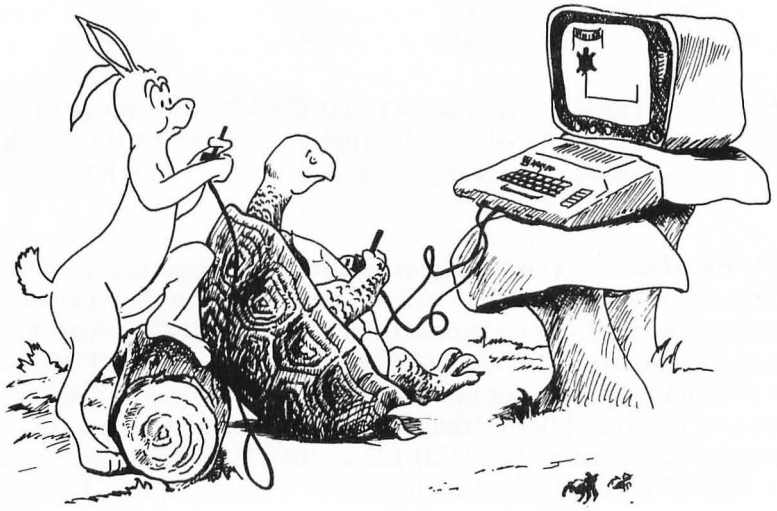
```
1 PRINT PEEK(54016):GOTO 1
```

TROFF — See TRACE.

TRON — See TRACE.

TRUNCATION — The BASIC command INT(X) will TRUNCATE all digits to the right of the decimal from X. Any number printed by Atari BASIC will be TRUNCATED to nine digits. TRUNCATED numbers are not rounded up or down to the nearest digit, the excess digits are just dropped.

TURTLE GRAPHICS — The familiar Cartesian coordinate system of drawing uses notation such as PLOT X,Y and DRAWTO X+5,Y+10. TURTLE GRAPHICS puts you in the drivers' seat with statements such as MOVE 10, TURN 90, MOVE 5. The dif-



ference is that the turtle method is much easier for people, especially small ones, to conceptualize. PILOT, LOGO, WSFN, and some FORTHs use TURTLE GRAPHICS. The cursor is usually not turtle shaped, although there is no reason not to have one implemented if one desires. Atari's LOGO does use a turtle shaped cursor.

TYPO — When you type in programs from Atari magazines you are apt to make a few mistakes in keying in the lines. If you make a syntax error, BASIC will tell you immediately. If you type a wrong line number or DATA statement you will generally have some trouble tracing the mistake. Antic magazine and ANALOG both have commissioned programmers to provide a typing error detector program. Antic calls their program TYPO and ANALOGs is CHECK. The TYPO program is printed in Antic Volume 2, #1. CHECK is in ANALOG #10. These programs basically do a sum of the tokenized programs after you have typed them in and they compare the sum to known correct values. Any differences will help you find the line number where the error lies. One other tip — always SAVE your program before typing RUN, even if you are sure there are TYPOs. If the computer locks up and you do not have a backup, you will have to retype all of the code again.

U

UNARY OPERATORS — A UNARY OPERATOR is one that changes or works on one operand. The minus sign (–) is a UNARY OPERATOR. It takes a number and changes it to its negative without affecting or comparing it to anything else. The BASIC operator NOT is a UNARY OPERATOR. It produces the complement or logical opposite of the operand it works on. AND, OR, =, <>, etc. are binary operators because they work with two operands.

UNCONDITIONAL BRANCH — A statement such as GOTO or GOSUB sends program control to another line number UNCONDITIONALLY. That means the program must go to that line number no matter what else is true. Conditional branches which use IF/THEN, ON/GOSUB, and ON/GOTO will branch if some conditions are met. Conditional branches are necessary to put customization or sophistication in your programs. Without conditional branches, a program must do the same thing every time it is run. See CONDITIONAL BRANCH.

UPLOAD — UPLOADing is the process of sending a program or file from one computer to another. The most common application of UPLOADing is sending a program to a bulletin board system for others to copy and use. A BASIC program should be in LIST format when you send it. If it is SAVEd, the tokenized file will contain many control characters which will be translated by the receiving computer in most cases. A binary load or object file must be sent in ATASCII mode where all characters will be sent and received as they are stored in the file. A terminal program with XMODEM protocol, such as AMODEM, should be used to UPLOAD files with control characters. The bulletin board system will usually prompt you as to whether you need to be in ASCII or ATASCII mode when UPLOADing.

UPWARD COMPATIBLE — Compatibility between hardware and software became an issue when Atari introduced the 1200XL in January of 1983. Despite claims that existing software would run on the new 600XL, 800XL and 1200XL, changes in the operating system made much of the existing software base incompatible. This was not the fault of Atari, but rather, it was the programmers who violated documented Operating System rules. The TRANSLATOR program makes all of the existing software UPWARDly COMPATIBLE with the new machines.

USERS' GROUP— A USERS' GROUP is an informal association of individuals who share the same interest in computers, usually of a particular brand. Atari has not supported USERS' GROUPs until recently and now they are a major source of information about the end user. USERS' GROUPs typically meet monthly and discuss hardware, software, rumors, facts, and Atari, Inc. Often the experts in a group will give classes in programming or applications software. Many groups publish a newsletter in which one finds the latest tricks and news. One of the best attractions of a USERS' GROUP is the library of Public Domain software most USERS' GROUPs maintain. Members can copy the disks or buy for minimal cost programs which are in the public domain. This means that the programs are not copyrighted and can be given away or traded freely. There are hundreds of disks full of known public domain programs for Atari computers.

USR — The USR function is the method you can use to access machine language routines from BASIC. At least one parameter must be provided with a USR function (sometimes called a USR jump because it "jumps" into a machine language program.) The parameter which must be provided is the decimal address in memory of the routine to be run. Other parameters provided in the function are put on the hardware stack (Page 1) and used by the machine language routine if necessary. The parameters are converted to hexadecimal numbers in two bytes and pushed onto the stack, low byte first, then high byte. The address of the machine language program is not pushed onto the stack. After all parameters are pushed, the number of parameters pushed is then put on the stack. If none is entered then a 0 is pushed on.

In order to use a USR function, you must load in your routine (page 6 is a likely place). Then you simply issue the command ?USR(ADDRESS) where ADDRESS is the address in memory of the start of the program.

UTILITY— A UTILITY is a program written to help you program better or to perform some help routine. UTILITIES are generally available on public domain disks from users' groups. There is a UTILITY available to do virtually anything you want, from designing players, to setting up printers, to calculating HEX from DEC codes.

V

VAL — In BASIC, VAL performs the opposite function as STR$. VAL converts a string which is made of numeric characters to a numerical variable or value. X=VAL(Y$) will assign the value of 123 to the variable X if Y$ were a string called "123". If Y$ is an alphabetic character, an error will result.

VARIABLES — There are several kinds of VARIABLES, scalar VARIABLES, string VARIABLES, and array VARIABLES. Scalars are undimensioned. String and Array VARIABLES may be dimensioned or undimensioned. Undimensioned strings and arrays rarely appear in programs. A maximum of 128 VARIABLES of all kinds can be used at any time. The VARIABLES are not cleared out after they are used. Typing NEW will wipe out all existing VARIABLES and allow you to start over. It is wise to LIST the program to disk or tape before you do this or you will lose VARIABLES and program together. A scalar variable takes up six bytes of memory regardless of the size of the number. Each scalar is stored as a six byte binary coded decimal (BCD). This means it uses 12 four bit nibbles to store each value. Two nibbles are used for the exponent. Arrays, which also store numeric data use six bytes per dimension unit. An array dimensioned for 1000 elements will take up 6,000 bytes. String variables use only one byte per DIMensioned element. The following examples will clarify the difference among the variable types.

SCALAR	X	6 Bytes– Binary Coded Decimal (BCD)
ARRAY	X(5)	30 Bytes– BCD– Must DIM X(5) first
STRING	X$="ABCD"	4 Bytes– Must DIM X$(4) first

All VARIABLES have entries in the Variable Name Table (VNT). The pointer to the beginning of the VNT is in locations $0082 and $0083 (130 and 131 decimal). The VNT sits at the bottom of free BASIC memory. VARIABLES are entered in the table in the order in which they are used. They are numbered from $80 to $FF (this determines the maximum number of 128 variables). The VNT is recognizable when viewed directly via a file or memory dump by the inverse characters at the end of each variable name. Scalar VARIABLES end in an inverse character. (Same character as in normal name with bit 7 set). String VARIABLES end in an inverse dollar sign ($). Array VARIABLES end in an inverse right parenthesis ")".

VARIABLE LISTER — You can check the variable names which have been used in your BASIC program in BASIC XL simply by typing the LVAR command. Likewise, if you have the Monkey Wrench installed in the right slot of your 800, you can type >V and get a list of the variables used. The following utility program can be placed at the end of your BASIC program and used to give you a list of current variable names used. This is handy to check for unused names as you are only allowed a maximum of 128 total. Unused variable names are not deleted until you LIST and ENTER a BASIC program (otherwise your old VNT is SAVEd). Type in the program and make sure it works first. You will see a list of the 18 variables in this program (they all start with a V). Then LIST it to a disk or tape file. Load in your BASIC program and then ENTER the VARIABLE LISTER program below. By typing GOSUB 32000, you will get a list of variable names used. Remember to put a :RETURN at the end of the subroutine to return you to your BASIC program:

```
32000 REM ABCS OF ATARI COMPUTERS
32010 REM PROGRAM: VLIST.BAS
32020 REM LIST THIS PROGRAM TO DISK.
32030 REM USE IT AS A SUBROUTINE (GOSU
B
```

```
32040 REM 32000) TO LIST OUT ALL VARIA
BLES
32050 REM YOU HAVE USED.  NOTE THAT EV
EN
32060 REM WHEN YOU CHANGE THE NAME OF
A
32070 REM VARIABLE, IT REMAINS IN THE
32080 REM VNT UNTIL YOU 'LIST' AND
32090 REM 'ENTER' THE PROGRAM BACK TO
32100 REM DISK.
32110 VNTP=PEEK(130)+PEEK(131)*256:VNT
D=PEEK(132)+PEEK(133)*256
32120 TRAP 32130:DIM VNAM$(1023):TRAP
32767
32130 VNAM$(1)=" ":VNAM$(1023)=" ":VNA
M$(2)=VNAM$
32140 V82=PEEK(82):POKE 82,0:PRINT CHR
$(125);"   SEARCHING....    "
32150 VK=1:VS=1:VJ=0
32160 FOR VI=VNTP TO VNTD-1
32170 VX=PEEK(VI)
32180 IF VX<128 THEN VNAM$(VS+VJ,VS+VJ
)=CHR$(VX):VJ=VJ+1:NEXT VI:VS=VS+8:GOT
O 32220
32190 VX=VX-128:VNAM$(VS+VJ,VS+VJ)=CHR
$(VX)
32200 VK=VK+1:VS=8*VK-7:VJ=0
32210 NEXT VI
32220 VNAM$(VS)=""
32230 VENT=LEN(VNAM$)/8:? "   SORTING.
.. "
32240 TRAP 32250:DIM VTX$(8):TRAP 3276
7
32250 VLEN=8:VLEN1=VLEN-1:VMAX=VLEN*(V
ENT-1)+1
32260 FOR VI=1 TO VMAX STEP VLEN
32270 VDONE=1
32280 FOR VK=1 TO VMAX-VI-VLEN1 STEP V
LEN
32290 VSLEN1=VK+VLEN1:VSLEN=VK+VLEN:VS
LSL1=VSLEN+VLEN1
32300 IF VNAM$(VK,VSLEN1)<=VNAM$(VSLEN
,VSLSL1) THEN 32330
32310 VDONE=0
32320 VTX$=VNAM$(VK,VSLEN1):VNAM$(VK,V
SLEN1)=VNAM$(VSLEN,VSLSL1):VNAM$(VSLEN
,VSLSL1)=VTX$
32330 NEXT VK
```

```
32340 IF VDONE THEN POP :GOTO 32360
32350 NEXT VI
32360 ? :? :? "  VARIABLE LISTING  ":?
32370 FOR VI=1 TO INT(LEN(VNAM$)/8)
32380 VS=8*VI-7:? VNAM$(VS,VS+7),
32390 NEXT VI
32400 ? :? :? "  NUMBER OF VARIABLES =
 ";VENT
32410 POKE 82,V82
32420 REM ************************
32430 REM THIS PROGRAM WILL ALWAYS
32440 REM LIST ITS OWN VARIABLES.
32450 REM ************************
32460 REM VDONE VENT   VI      VJ
32470 REM VK    KSLEN  KSLEN1  KSLSL1
32480 REM VMAX  V82    VS      VLEN
32490 REM VLEN1 VNAM$  VNTD    VNTP
32500 REM VX    VTX$
32510 REM # VARIABLES = 18
32520 REM RETURN
32530 REM REMOVE REM IN 32520 TO USE T
HIS AS A SUBROUTINE
```

Based on program concept in A-MAGIC Report, 2/83, by Mike Hall.

VARIABLE NAME TABLE — The VARIABLE NAME TABLE (VNT) is a list of all variables which have been created in a BASIC program. The VNT sits right at the bottom of free RAM. Because this location will change depending upon conditions, a pointer is used to find the VNT. The pointer is in locations $0082 and $0083 (130 and 131 decimal). Use ? PEEK(130)+PEEK(131)*256 to find the start of the VNT. The following short program will list the variables in the VNT. Note that the last character of each variable name is inverse.

Scalar variables end in an inverse character
Array variables end in an inverse left parenthesis "("
String variables end in an inverse dollar sign ($)

```
10 LET ONE=1:TWO=2:THREE=3:DIM ARRAY(5),
TWODEE(3,4),STRING1$(3),BIGSTRING$(789)
20 VNTAB=PEEK(130)+PEEK(131)*256
30 FOR CHAR=0 TO 50:? CHR$(PEEK(VNTAB+CH
AR)):NEXT CHAR
```

VARIABLE VALUE TABLE — The VARIABLE VALUE TABLE (VVT) sits immediately above the VNT and it contains the data on where, what size, and what types of variables have been created. The pointer to the VVT is in locations $0086 and $0087 (134 and 135). Eight bytes are assigned to each variable name. The following information is contained in the VVT:

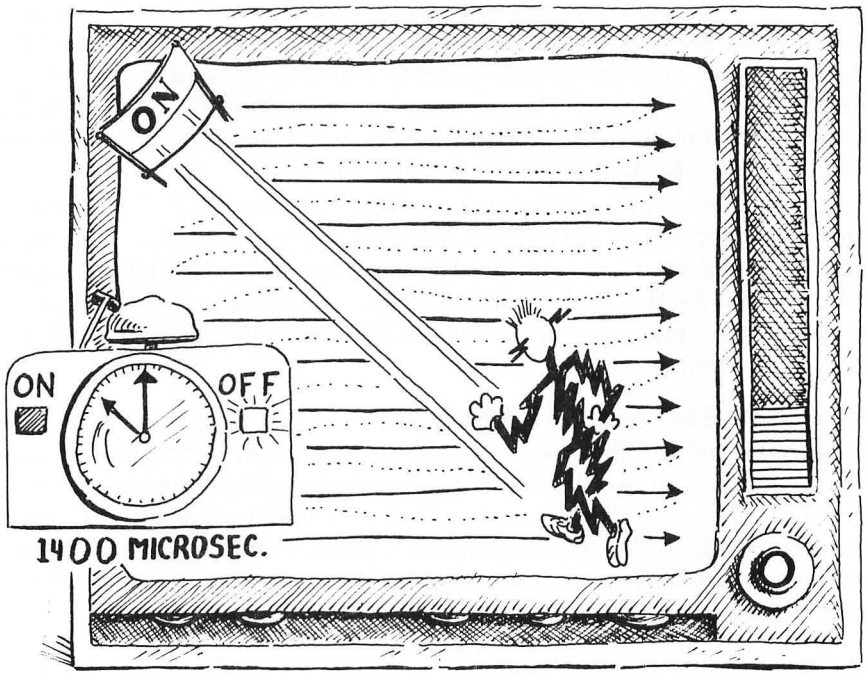
Variable Type		BYTE 0	1	2	3	4	5	6	7
SCALAR	0	V#	SIX BYTE BCD VALUE FOR THE VARIABLE						
DIMmed Array	65	V#	STARP OFFSET			X DIM+1 LO	HI	Y DIM+1 LO	HI
unDIMmed Array	64	V#							
DIMmed String	129	V#	STARP OFFSET			STRING LENGTH LO	HI	DIM LO	HI
unDIMmed String	128	V#							

The first byte in the VVT entry is a code for the type of variable. Scalars use a 0, arrays use 64 or 65, and strings use 128 or 129. The second byte is the number of the byte's entry in the table. The V# can be from 0 to 127. The next six bytes are used as a binary coded decimal representation of the number for scalar variables. Strings and arrays use these six bytes for the offset from the beginning of the String/Array area and to note the DIMensions of the variables. The string/array area is pointed to by locations $008C and $008D (140 and 141 decimal). The offset is the number of bytes from the beginning of this area where the variable data for this variable is stored. The following program can be used to dump the contents of the VVT so you can better visualize how it works.

```
10 LET ONE=1:TWO=2:THREE=3:DIM ARRAY(5),
TWODEE(3,4),STRING1$(3),BIGSTRING$(789)
20 VVTAB=PEEK(134)+PEEK(135)*256
30 FOR VNUM=0 TO 10:? "VARIABLE #";VNUM;
" ";
40 FOR X=0 TO 7:? PEEK(VVTAB+VNUM*8+X);"
,";:NEXT X:?
50 NEXT VNUM
```

VBI — Vertical Blank Interrupt. During the vertical blanking time, short machine language routines can be executed. There are some routines that must be executed by the Operating System: Update hardware registers, read keyboard, change timers, and set attract mode register. Short routines of about 2,000 machine cycles can be done in the immediate VBI mode. Longer routines of up to 20,000 machine cycles can be done in the deferred VBI mode. The interrupts are implemented through vectors which are checked during every vertical blank. The immediate mode vector is at $0222 and the deferred mode vector is at $0224. If the routines cannot be completed in the cycles they are allotted, they will interfere with the screen writing and disrupt the display.

VBLANK — VERTICAL BLANK — A television produces images by scanning an electron beam across the surface of a phosphor coated faceplate. The beam starts in the upper left corner and travels left to right, turns off, moves down one line, and continues scanning until all 525 lines are covered. When the beam gets to the bottom (after 1/60



second), it is turned off and restarted at the top left. During this time, the OS takes care of its business. The shadow registers are transferred to hardware registers and other short programs can be executed. This interval is called the VERTICAL BLANK.

VECTOR — A VECTOR is a memory location which contains the location of the first byte of a table or subroutine. VECTORs are comprised of two bytes. The low byte is usually presented first, followed by the high byte (LO-HI). A VECTOR is similar to a pointer.

VERIFY — Atari DOS does a read after every write to VERIFY that the data has been written correctly. This READ routine makes writing time about twice as long as it would be without the VERIFY. You can eliminate the VERIFY by modifying DOS. Type POKE 1913,87 in BASIC and then go to the DUP.SYS menu. Write the DOS files back to disk using the H option. This version of DOS is now modified to NOT do the VERIFY. To return to standard DOS with VERIFY, you can repeat the procedure by doing a POKE 1913,80. Generally, very few errors are made during writing, so the verification is usually not needed.

VERTICAL SCROLLING — VERTICAL SCROLLING is the movement of information vertically through the screen window. The LIST command in BASIC does VERTICAL COARSE SCROLLING. The ANTIC handles VERTICAL FINE SCROLLING. VERTICAL SCROLLING on the Atari computer is very easy to implement because of

the Display List. The first requirement for scrolling is to have something in memory. The object of scrolling is to move the data in memory past the screen. This can be done by pushing the data past the area of memory which is mapped to the screen, OR by changing the pointer which tells the screen where to go to find screen data. The latter approach is much simpler to do. In the XL series, fine scrolling is implemented very simply by typing POKE 622,255:GR.Ø <Return>.

There is an instruction in the Display List which tells the ANTIC chip where to find screen memory. This is the LMS (Load Memory Scan) instruction. By incrementally changing the two address bytes of the LMS, you can implement VERTICAL COARSE SCROLLING. See Section 6 of *De Re Atari* for a detailed treatment of scrolling. In order to do fine scrolling, you must set the VERTICAL FINE SCROLLING ENABLE BIT. See the next entry.

VERTICAL FINE SCROLL ENABLE BIT — In order to properly implement vertical fine scrolling, bit 5 of the Display List instruction must be set. This is done by adding 32 to the decimal value of the instruction or adding $20 to the hex value. To set up GR.0 for vertical fine scrolling, you would use the instruction $62 ($42 + $20) or 98 (66 + 32 decimal). In addition to setting bit 5, you must also put the number of pixel lines you wish to scroll in register $D405 (54277 decimal). Both of these actions must be done for VERTICAL FINE SCROLLING.

VIDEO 80 — Eighty Column Emulator — *COMPUTE!* Magazine, April 1983 published a program which defines a new device called "V:" which allows 80 column formatting on monochrome televisions or monitors. No special hardware is needed, however the keyboard editor is not implemented. This program could be merged with a word processor to create an 80 column utility program.

VIRTUAL MEMORY — VIRTUAL MEMORY or VIRTUAL Storage refers to a technique of using auxiliary memory (floppy, hard disk, RAM disk) to swap data in and out of the main memory of a computer. This could give you unlimited or at least very extensive working memory. This technique is usually implemented on minicomputers or mainframes but it has not yet been demonstrated on Atari computers.

VISICALC — VISICALC was developed by Personal Software (later called VisiCorp) and was responsible for an estimated 30-50 percent of personal computer sales. Atari recently bought the rights to distribute VISICALC for Atari computers. VISICALC is a spreadsheet program. This means that it is essentially designed to handle a matrix of numbers and variables. The matrix is made up of rows and columns. Columns are labeled A through BK and rows are numbered 1 through 254. Each cell has a location, such as A1 or BK254. Cells can be added together by using commands such as +A1+A2. If the value stored in 1 were changed then every other cell that referenced A1 would change. Very complex models can be built. Financial statements and budgets are favorite applications for VISICALC. The limitation of 48K of memory prohibits very extensive models from being built on Atari computers. The program is so large that only 21K is left after the program is loaded in.

VOICE — VOICE is the channel through which the SOUND commands are heard. Atari BASIC supports four VOICES (0 through 3). The first argument of the SOUND statement specifies the VOICE number.

VOICE SYNTHESIZER — A VOICE or SPEECH SYNTHESIZER is a hardware or software addition to your computer which allows you to output sound similar (in some ways, at least) to a human voice. This synthesis is accomplished by using the standard phonemes to modulate waveforms to resemble speech. Most programs now will take standard English (or other words) and attempt a translation directly into speech. Often you will have more understandable sound if you use a phonetic spelling, like, KOM-PEW-TER instead of COMPUTER. S.A.M. (software) by Don't Ask Software and the Voice Box (hardware) by the Alien Group are two reliable VOICE SYNTHESIZERS.

VTOC — Volume Table Of Contents. Sector 360 on Atari DOS disks is the VTOC. The main feature is the bit map of the sectors in use. Every sector is represented by a bit. DOS marks the occupied sectors with a 0 and the free sectors with a 1. See SECTOR TYPES for a full treatment of VTOC.

W

WAIT . . . AND — In Microsoft BASIC, WAIT...AND uses an address, a mask byte, and a compare byte to halt a program until certain conditions are met. When this command is encountered, execution stops until the compare byte, when logically ANDed to the mask byte, equals the byte in the memory address specified.

WARMSTART — A WARMSTART will reset most registers to their default values. Pressing SYSTEM RESET does a WARMSTART. Jumping through location 8 ($8) will also do a WARMSTART. Programs in memory will not be cleared as they will be with a COLDSTART. Variables will also remain unchanged. All Operating System RAM in pages 2 and 3 is set to zero. RAM from $0010 to $007F is also set to zero. The screen margins and colors are reset to their default values.

WARP SPEED — WARP SPEED is a technique developed by Richard Adams of Happy Computers which speeds up the input and output to disk drives by about three times. WARP SPEED works with the Happy Enhancement. All reading and writing occurs at around 40,000 baud instead of the original 19,200 over the serial I/O port. WARP SPEED is implemented by a special modification to DOS available from Happy Computers.

WIDE PLAYFIELD — By POKEing 559,35 you can enable the WIDE PLAYFIELD. This mode of display broadens the screen to 48 characters wide (only 40 are visible, unfortunately). The Atari hardware fetches 192 color clocks for a scan line. However, the hardware also cuts off the four outside characters (two on each side), which leaves 44 characters in order to meet NTSC standards. If there were more clocks on a line, the TV would not synchronize. The editor is still set for 40 characters in this mode so columnar output printed to the screen will not line up properly. The WIDE PLAYFIELD will eliminate unsightly edges on the playfield by placing them outside of the screen area. The location of the register is $022F (559), and setting bits 0,1, and 5 enables the WIDE PLAYFIELD. See NARROW PLAYFIELD.

WILD CARD — A WILD CARD character is a dummy character, such as an asterisk (*) or a question mark (?), which can take on any value in a name. Wild cards are used in the DOS utility. For example, the asterisk represents any filename or filename extender. Pressing D and *.* will delete every file on a disk. The question mark is a WILDCARD for a single character in a filename. Pressing D and ?.BAS will delete all one-character file names ending in .BAS. Pressing D and *.BA? will delete all files with BAK, BAS, or BA(any character) as its extension. Also, be careful not to use a wildcard when renaming files or you may end up with multiple identical filenames on a directory.

WRITE ONLY REGISTERS — Many of the hardware registers in the Atari computer cannot be read. They are WRITE ONLY REGISTERS. Read/Write registers are also known as RAM. The values written in these registers are transferred to other locations for other work. Some registers are used for writing, but when read, contain bytes grabbed from other locations. *The Hardware Manual* gives a fairly complete list of read and write functions of all of the registers in Atari computers.

WRITE PROTECT — The square notch near the top right corner of a floppy disk is the write enable notch. When the hole is covered with an opaque tab, there is no way to write to, erase, or format a disk in the drive unless your drive suffers from a hardware failure. This does happen, so be careful when you have difficulty with reading or writing. Black electrical tape makes an excellent write protect tab if you are short of the real thing. White adhesive labels are not safe because the photocell which acts as the interlock can sometimes read through the paper. Almost all disks can be flipped over and used as double sided floppies. A write enable notch must be punched or cut on the left side (opposite the existing one) and the disk can be formatted for use. Very few disks are not useable on the flip side despite their rating as single sided disks due to the low density of the tracks. One interesting point is that the writing surface is on the opposite side of the floppy disk as the label. The Read/Write head on the Atari 810 is on the bottom side of the drive.

WRITE PROTECT OVERRIDE — You can install a switch on the front of your Atari 810 disk drive to enable yourself to write on the back side of a floppy disk without cutting a notch or a write enable hole. This operation is not recommended for those not handy with a soldering iron or components, nor for those with a disk drive in warranty. Also note that you must attempt this modification at your own risk. It has been known to affect drive speed and will give unpredictable results with a Happy Enhancement due to its use of the write enable line.

This technique puts two LEDs on the front face of the drive to indicate whether or not you can format any disk in the drive (notched or not).

You will need five parts to make this modification.

1. A green LED.
2. A yellow LED.
3. A DPDT switch (on-off-on).
4. A 2.2K ohm resistor, 1/4 watt.
5. A 330 ohm resistor, 1/4 watt.

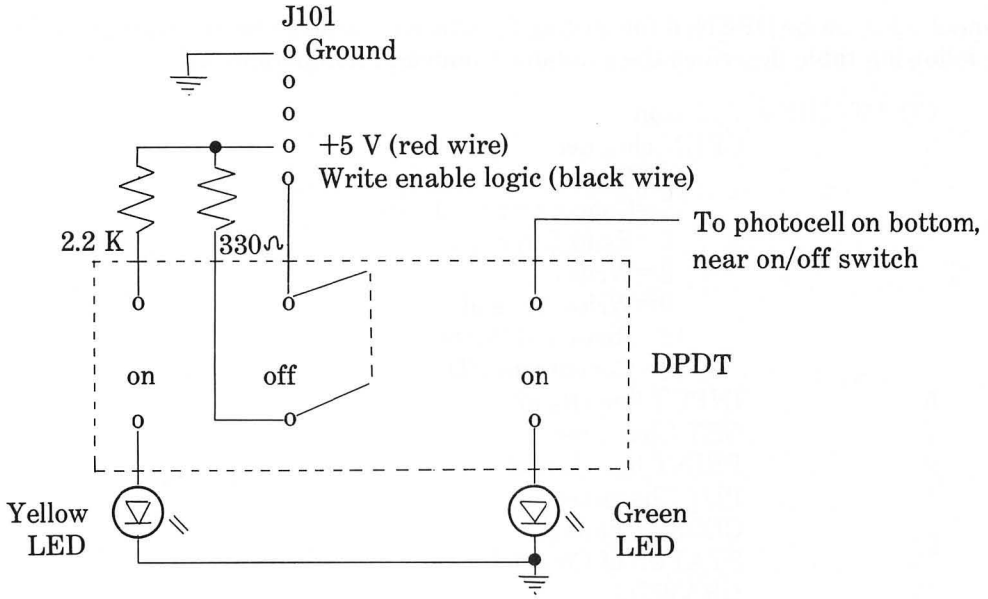
Mount the switch and two LEDs on the front bezel of your 810. This involves some judgement in drilling the holes. Remove the four screws under the circular tabs on the top of the drive case and open the top case. Find the side board (with the metal can shield) and locate the five-conductor connector labeled J101. If you have five wires in this connector, remove #2, the red one. If you have only three wires in the connector, pull the plug out.

Make the connections between the switch, LEDs, resistors, and connector according to the schematic below.

When everything is connected properly, flip the switch down and the yellow LED should turn on. You should be able to format a disk with no notch or with a write protect tab on it. Try it.

Flip the switch up and the green LED should come on. This indicates that you are in the normal mode. You will need a notch to write (you will not be able to write without one).

With the switch in the center (no lights on), you will not be able to write to any disks.



X

X REGISTER — The 6502 central processor of the Atari computer has several registers for use by the programmer. The X REGISTER is one of these. It is an eight bit register used for temporary storage, indexing or incrementing, and as a scratch pad for machine language operations. The codes LDX, INX, CPX, DEX, STX, TXS, and TAX use the X REGISTER.

XIO — The BASIC XIO command is a very powerful, yet poorly documented command. XIO commands provide a structured way of configuring registers or setting bits without POKEing or using machine language routines. Many DOS (DUP.SYS) functions can be performed through the XIO. All SIO commands can be engaged. A special graphics fill command is available only through the XIO. The RS232 port can be configured in BASIC only through the XIO series. XIO commands must contain the following components:

XIO	These three letters
Command No.	No. Between 3 and 254
Channel	IOCB no. between 1 and 7
Expression 1	Used to set up parameters
Expression 2	Used to set up parameters
Device	Specifies destination or file (D:, E:, C: etc.)

The format is shown in the following example which will delete a file called TEST on Drive 1.

```
XIO 33,#1,Ø,Ø,"D:TEST"
```

Channel #1 must be OPENed for writing for this command to be executed properly. The following table describes the available Command numbers in XIO.

COMMAND #	Function
3	OPEN channel 4=Read 5=Concurrent read (R:) 6=Read Directory 8=Write 9=Write Append 12=Read and Write 13=Concurrent I/O
5	INPUT line (Read)
7	GET Character
9	PRINT line (Write)
11	PUT Character
12	CLOSE Channel
13	STATUS of Channel
17	DRAWTO
18	Fill
32	Rename file on disk
33	Delete file on disk
34	Set DTR, RTS and XMT in RS232 serial port
35	Lock file on disk
36	Unlock file on disk; set baud, word size, and stop bits
37	Move pointer in disk buffer
38	NOTE pointer in disk file; set translation mode in R:
40	Set up concurent mode I/O in R:
254	Format disk

XL SERIES MEMORY USAGE CHANGES — Several variables in the database area (page 0 through 4) have been changed in the XL series Operating System. These changes do NOT conflict with any documented usage of these locations, but are merely additions. The following list describes the changes.

LOCATION		NAME IN REV. B OS	NAME IN XL OS USE
Dec.	Hex.		
0	0000	Reserved	LNFLG - for in-house use
1	0001	Reserved	NGFLAG - for power-up self test
28	001C	PTIMOT-Moved to 0314	ABUFPT - Reserved
29	001D	PBUFNT-Moved to 02DE	ABUFPT - Reserved
30	001E	PBUFSZ-Moved to 02DF	ABUFPT - Reserved
31	001F	PTEMP- Deleted	ABUFPT - Reserved
54	0036	CRETRY-Moved to 029C	LTEMP - Loader temporary
55	0037	DRETRY-Moved to 02BD	LTEMP
74	004A	CKEY -Moved to 03E9	ZCHAIN - Handler loader temp.
75	004B	CASSBT-Moved to 03EA	ZCHAIN
96	0060	NEWROW-Moved to 02F5	FKDEF-Fn. key table pointer
97	0061	NEWCOL-Moved to 02F6	FKDEF
98	0062	NEWCOL-Moved to 02F7	PALNTS - PAL/NTSC flag
121	0079	ROWINC-Moved to 02F8	KEYDEF - Key table pointer
122	007A	COLINC-Moved to 02F9	KEYDEF
563	0233	Reserved	LCOUNT - Loader temporary
568/9	0238/9	Reserved	RELADR - Loader
581	0245	Reserved	RECLEN - Loader
583/618	0247/6A	LINBUF-Deleted	Reserved
619	026B	LINBUF-Deleted	CHSALT - Alt. char. set point.
620	026C	LINBUF-Deleted	VSFLAG - Fine scroll temporary
621	026D	LINBUF-Deleted	KEYDIS - Keyboard disable
622	026E	LINBUF-Deleted	FINE - Fine scrolling enable
648	0288	CSTAT-Deleted	HIBYTE - Loader
654	028E	Reserved	NEWADR - Loader
668	029C	TMPX1-Deleted	CRETRY - Moved from 54
701	02BD	HOLD5-Deleted	DRETRY - Moved from 55
713/4	02C9/A	Reserved	RUNADR - Loader
715/6	02CB/C	Reserved	HIUSED - Loader
717/8	02CD/E	Reserved	ZHIUSE - Loader
719/20	02CF/D0	Reserved	GBYTEA - Loader
721/2	02D1/2	Reserved	LOADAD - Loader
723/4	02D3/4	Reserved	ZLOADA - Loader
725/6	02D5/6	Reserved	DSCTLN - Disk sector size
727/8	02D7/8	Reserved	ACMISR - Reserved
729	02D9	Reserved	KRPDEL - Key repeat delay
730	02DA	Reserved	KEYREP - Key repeat rate
731	02DB	Reserved	NOCLIK - Key click disable
732	02DC	Reserved	HELPFG - HELP key flag
733	02DD	Reserved	DMASAV - DMA state save

LOCATION		NAME IN REV. B OS	NAME IN XL OS USE
734	02DE	Reserved	PBPNT - Moved from 29
735	02DF	Reserved	PBUFSZ - Moved from 30
745	02E9	Reserved	HNDLOD - Handler loader flag
757	02F5	Reserved	NEWROW - Moved from 96
758/9	02F6/7	Reserved	NEWCOL - Moved from 97/8
760	02F8	Reserved	ROWINC - Moved from 121
761	02F9	Reserved	COLINC - Moved from 122
782	030E	ADDCOR - Deleted	JMPERS - Option jumpers
788	0314	TEMP2 - Moved to 0313	PTIMOT - Moved from 28
829	033D	Reserved	PUPBT1 - Power-up reset
830	033E	Reserved	PUPBT2 - Power-up reset
831	033F	Reserved	PUPBT3 - Power-up reset
1000	03E8	Reserved	SUPERF - Screen editor
1001	03E9	Reserved	CKEY - Moved from 74
1002	03EA	Reserved	CASSBT - Moved from 75
1003	03EB	Reserved	CARTCK - Cartridge checksum
1005/16	03ED/F8	Reserved	ACMVAR - Reserved
1017	03F9	Reserved	MINTLK - Reserved
1018	03FA	Reserved	GINTLK - Cartridge interlock
1019/20	03FB/C	Reserved	CHLINK - Handler chain

XMODEM — XMODEM is a terminal program used primarily for CP/M based computers. The important feature of XMODEM is the error checking technique which sends a checksum after each group of 128 bytes is transferred. This is known as the XMODEM protocol and it allows highly reliable communications between computers. The AMODEM program listed under AMODEM contains the XMODEM protocol. Ward Christensen, a prolific public domain CP/M author, is the originator of the Christensen XMODEM protocol.

Y

Y REGISTER — The Y REGISTER is used just like the X register in the 6502 processor. See X REGISTER.

YOUR ATARI COMPUTER — A Guide to Atari 400/800 Personal Computers by Lon Poole, Osborne/McGraw-Hill. This book is a very extensive reference guide to the Atari BASIC language and, to a lesser degree, to the DOS and Operating System. A section on advanced graphics provides a good background for machine language and player missile programming. Osborne/McGraw-Hill (address).

Z

ZERO PAGE — Memory locations 0 through 255 ($00 to $FF) are used by the Operating System, BASIC, and by machine language programmers who want to produce very fast code. Many OS pointers are in the lower half of page 0. BASIC uses the top half of page 0, and some free user memory is also available. Since many of the bytes in page 0 are used by BASIC in different routines, it is very easy to crash a program by interference with BASIC. Seven bytes between 203 and 209 are always untouched by BASIC.

ZERO SLASH — In order to differentiate the letter O from the number zero (0), printers and character set ROMs often use a slashed zero. A zero which has been interchanged with an O is a good place to start looking when debugging a BASIC program.

Z80 — The eight bit Z80 microprocessor, developed by Zilog, affiliate of Exxon Enterprises, was the first commercial computer on a chip. Radio Shack was the first company to design a personal computer around the Z80 and to mass market computers to the public. As a result, there is a huge base of software written for the Z80 based machines and the CP/M Operating System which is used to control these computers.

Atari Source List

FIRM	ADDRESS	PRODUCTS
ACORN SOFTWARE	634 N.CAROLINA AVE. S.E. WASH., DC 20003	
ACTIVISION	3255-2 SCOTT BLVD, SANTA CLARA, CA 95051	GAMES
ADS	9202 CEDAR CREST DR., AUSTIN, TX 78750	INTEGRATER
ADVANCED COMPUTING	5516 ROSECHILD, SHAWNEE, KS 66216	
ADVENTURE INTERNAT'L	P.O. BOX 343, LONGWOOD, FL 32750	GAMES
ALIEN GROUP	27 W. 23 ST.,NEW YORK, NY 10010	VOICE BOX
ALLEN MACROWARE	P.O. BOX 2205, REDONDO BEACH, CA 90278	SCREENDUMP S/W
ALOG COMPUTING	1040 VERONICA SPGS., SANTA BARBARA, CA 93105	WORD PROCESS.
ALPHA SYSTEMS	4435 MAPLEPARK RD., STOW, OH 44224	PROTECTION BOOK
ALPHACOM	2323 SOUTH BASCOM, CAMPBELL, CA 95008	PRINTERS
AMDEK	2201 LIVELY BLVD., ELK GROVE VILLAGE, IL 60007	3" FLOPPY DRIVE
AMERICAN SOFTWARE CB	MILLWOOD, NY 10546	CHEAP SOFTWARE
AMULET ENTERPRISES	P.O. BOX 25612, GARFIELD HTS., OH 44125	DISKED
A.N.A.L.O.G.	P.O. BOX 23, WORCESTER, MA 01603	MAG, SOFTWARE
ANCHOR AUTOMATION	6624 VALJEAN AVE., VAN NUYS, CA 91406	MODEMS
ANTIC	524 SECOND ST., SAN FRANCISCO, CA 94107	MAGAZINE
APOGEE SOFTWARE	9615 FARRALONE AVE., CHATSWORTH, CA 91311	
APOLLO INC.	1300 ARAPAHO, RICHARDSON, TX 75081	
APPLIED COMP. ALTERN	1600 WILSON BLVD., ARLINGTON, VA 22209	CRAM-90K-400
APX	P.O. BOX 3705, SANTA CLARA, CA 95055	SOFTWARE
ARCADE PLUS	5276 HOLLISTER #2, SANTA BARBARA, CA 93111	
ARTWORX	150 NORTH MAIN, FAIRPORT, NY 14450	SOFTWARE
ASTRA SYSTEMS	5230 CLARK AVENUE, LAKEWOOD, CA 90712	DISK DRIVE,DSDD
ATACOMP	RR #3, BOX 21, COGGON, IA 55218	COMPILER
ATARI, INC.	1312 CROSSMAN, SUNNYVALE, CA 94086	
ATARI	P.O. BOX 61657, SUNNYVALE, CA 94086	COMPUTERS
AUSTIN FRANKLIN ASSOC.	43 GROVE STREET, AYER, MA 0143248	K 80-COL BD.
AUTOMATED SIMULATION	1043 KIEL COURT, SUNNYVALE, CA 94086	SOFTWARE
AVALON HILL	4517 HARFORD, BALTIMORE, MD 21214	GAMES
AXIOM CORPORATION	1014 GRISWOLD, SAN FERNANDO, CA 91340	PRINTERS
AXLON	1287 N. LAWRENCE STA, SUNNYVALE, CA 94089	RAM BOARDS
BANK INC.	4 ELM ST, BRAINTREE, MA 02167	
BIG FIVE SOFTWARE	P.O. BOX 9078-185, VAN NUYS, CA 91409	MINR 20 49'ER
BIT 3 COMPUTER	8120 PENN AVE SO., MINNEAPOLIS, MN 55431	80 COL. BOARD
BIZCOMP	P.O. BOX 7498, MENLO PARK, CA 94025	1080 VERSAMODEM
BRAM INC	18779 KENLAKE PL. NE, SEATTLE, WA 98155	GAMES
BRODERBUND	17 PAUL DR., SAN RAFAEL, CA 94903	GAMES
BUDGECO	428 PALA AVE, PIEDMONT, CA 94610	GAMES
BUSINESS DATA CENTER	6890 KINNE ST, E. SYRACUSE, NY 13057	
C.A.P. SOFTWARE	69 NEW BOSTON ROAD, YORK, ME 03909	DISK WIZARD
CALIFORNIA MICROLINK	1287 LAWRENCE STA, SUNNYVALE, CA 94086	EASY I/O
CAVALIER COMPUTER	1223 CAMINO DEL MAR,DEL MAR, CA 92014	GAMES
CBS SOFTWARE	41 MADISON AVE., NEW YORK, NY 10010	GAMES
CDY CONSULTING	421 HANBEE, RICHARDSON, TX 75080	ONMIMON, OS BOA
CODE WORKS	P.O. BOX 550, GOLETA, CA 93116	SOFTWARE
COMPUTER AGE,INC	9433 GEORGIA AVE., SILVER SPRGS, MD 20910	
COMPUTER MAGIC	P.O. BOX 2634, HUNTINGTON STA, NY 11745	
COMPUTER ALLIANCE	21115 DEVONSHIRE #132, CHATSWORTH, CA 91311	BASM COMPILER
COMPUTER CONTROL CTR	5005 CASS ST., SAN DIEGO, CA 92109	80-COL S/W

FIRM	ADDRESS	PRODUCTS
COMPUSERVE	2180 WILSON ROAD, COLUMBUS, OH 43228	INFO UTILITY
COMPUTARI	9607 ATHLONE, DALLAS, TX 75218	FINANCIAL WIZ
COMPUTE!	BOX 5406, GREENSBORO, NC 27403	MAGAZINE
6COMPUTE SEEN	3272 E. ANAHEIM, LONG BEACH, CA 90804	
COMPUMAX	BOX 7239, MENLO PARK, CA 94025	SOFTWARE
CONTINENTAL SOFTWARE	11223 S. HINDRY, LOS ANGELES, CA 90045	
CORVUS SYSTEMS	2029 OTOOLE, SAN JOSE, CA 95131	WINCHESTERS
COSMI	7031 CRES RD., PALOS VERDES, CA 90274	GAMES
CREATIVE COMP.S/W	201 SAN ANTONIO CIR, MT. VIEW, CA 94040	
CREATIVE COMPUTING	39 E. HANOVER, MORRIS PLAIN, NJ 07950	MAGAZINE
D-TECH DATA CORP.	3251 TECH DRIVE N, ST.PETERSBURG,FL 33702	RAM-$100
DATABAR CORP.	10202 CROSSTOWN CIR, EDEN PRARIE, MN 55344	BAR CODE PROGRAMS
DATAMOST,INC.	20660 NORDHOFF ST., CHATSWORTH, CA 91311	BOOKS, SOFTWARE
DATASOFT, INC.	19519 BUSINESS CTR, NORTHRIDGE, CA 91324	SOFTWARE
DAVID BOHLKE	192 NORTH LINN, COGGON, IA 52218	ATACOMP-COMPILER
DILITHIUM PRESS	11000 SW 11TH ST. #E, BEAVERTON, OR 97005	BOOKS
DISCWASHER	1407 N. PROVIDENCE, COLUMBIA, MD 65205	JOYSTICKS
DON'T ASK	2265 WESTWOOD BL. #B-150, LOS ANGELES, CA 90064	SOFTWARE
DORSETT	BOX 1226, NORMAN, OK 73070	EDUC. SOFTWARE
DOW JONES SOFTWARE	STOCK ANALYSIS	
DRESSELHAUS COMPUTER	P.O. BOX 929, AZUSA, CA 91702	
DYNACOMP	1427 MONROE AVE, ROCHESTER, NY 14618	SOFTWARE
EATERN HOUSE S/W	3239 LINDA DR.,WINSTON-SALEM, NC 27106	UTILITY PRODS
ECRL, INC.	P.O. BOX 387, CANBY, OR 97013	52K RAM FOR 400
EDU-WARE	28035 DOROTHY, AGOURA, CA 91303	EDUCATION SOFTWARE
EDUFUN	1100 RESEARCH RD, ST. LOUIS, MO 63132	SOFTWARE
EDUSOFT	P.O. BOX 2650, BERKELEY, CA 94702	EDUCATIONAL SOFTWARE
ELCOMP PUBLISHING	53 REDROCK LANE, POMONA, CA 91766	LANGUAGES, BOOKS
ELECTRONICS ARTS	2755 CAMPUS DRIVE, SAN MATEO, CA 94403	PINBALL SET
ELITE DIGITAL	P.O. BOX 1414, MELVILLE, NY 11747	LONG CABLES
ENGLISH SOFTWARE CO.	P.O. BOX 3185, REDONDO BEACH, CA 90277	GAMES
ENVIRONMENTAL CTRL S	9319 WILLOWVIEW LANE, HOUSTON, TX 77080	BSR X-10 CRTLR
EPSON AMERICA	3415 KASHIWA ST., TORRANCE, CA 90505	PRINTERS
FIRST STAR SOFTWARE	22 E. 41ST ST., NEW YORK, NY 10017	GAMES
FORTH INTEREST GROUP	P.O. BOX 1105, SAN CARLOS, CA 94070	FORTH DIMENSION
FUNSOFT	28611 CANWOOD ST., AGOURA, CA 91301	GAMES
G.A.M.E.S.	6626 VALJEAN ST., VAN NUYS, CA 91406	
GAMESTAR	1302 STATE ST., SANTA BARBARA, CA 93101	GAMES
GAMMA SOFTWARE	P.O. BOX 23625, LOS ANGELES, CA 90025	
GEBELLI SOFTWARE	1787 TRIBUTE RD.,#6, SACRAMENTO, CA	GAMES
GREENBRIAR MARKETING	8225 E. ROVEY AVE., SCOTTSDALE, AZ 85253	ITALK II SPEECH
HAPPY COMPUTERS, INC.	P.O. BOX 1268, MORGAN HILL, CA 95037	DRIVE ENHANCER
HARDSEL	P.O. BOX 565, METUCHEN, NJ 08840	MONITOR OUTPUT
HAYES MICROCOMPUTER	5835 PEACHTREE CRNRS. E., NORCROSS, GA 30092	MODEMS
HI-RES	933 LEE ROAD, #325, ORLANDO, FL 32810	ATARI MAGAZINE
HIGH COUNTRY MICROSY	THE 4TH WORKS, BOX 21147, DENVER, CO 80221	RAMPAGE-SHERLOK
HIGH TECH SOFTWARE	9910 US 395 NORTH, RENO, NV 89506	CAR DIAGNOSTICS
HOFACKER (ELCOMP)	53 REDROCK LANE, POMONA, CA 91766	BOOKS/FORTH
HUMAN ENG. SOFTWARE	150 NORTH HILL DR., BRISBANE, CA 94005	GAMES
IJG INC.	1953 W. 11TH, UPLAND, CA 91786	UTILITIES
IMAGIC	981 UNIVERSITY AVE., LOS GATOS, CA 95030	GAMES
IN HOME SOFTWARE	2485 DUNWIN DR#8, MISSISAUGA, ON CAN	L5L 1T1 UTILITIES
INDUS SYSTEMS	9304 DEERING AVE., CHATSWORTH, CA 91311	DISK DRIVES
INFOCOM	P.O. BOX 855, GARDEN CITY, NY 11530	ADV. GAMES
INNOVATIVE SOFTWARE DESN.	920 1ST NATL., BANK, LAS CRUCES, NM 88001	GAMES
INTEC PERIPHERALS	906 E. HIGHLAND AVE, SAN BERNARDINO, CA. 92404	RAM 48K-32K-16K
ISLAND GRAPHICS,INC.	P.O. BOX V, BETHEL ISLAND, CA 94511	
JERSEY SYSTEMS	P.O. BOX 332, EDISON, NJ 08818	RAM
JV SOFTWARE	3090 MARK AVE., SANTA CLARA, CA 95051	GAMES
K-BYTE	1705 AUSTIN, BOX 456, TROY, MI 48099	SOFTWARE
KANGAROO	332 SOUTH MICHIGAN, #700, CHICAGO, IL 60604	KIDS GAMES
KRELL SOFTWARE	1320 STONY BROOK RD., STONY BROOK, NY 11790	EDUC. SOFTWARE
L&S COMPUTERWARE	1589 FRASER, SUNNYVALE, CA 94087	GAMES
LEADING EDGE	8624A SPICEWOOD SPGS, BOX 10998, AUSTIN, TX 78766	DISK SYSTEMS
LIGHTNING SOFTWARE	P.O. BOX 11725, PALO ALTO, CA 94306	MASTERTYPE
LJK ENTERPRISES	7852 BIG BEND RD., ST. LOUIS, MO 63119	APPLICATION SOFTWARE
LONDON SOFTWARE	374 WILDWOOD, PIEDMONT, CA 94611	
LOOKING GLASS SOFTWARE	544 FORT LARAMIE DR., SUNNYVALE, CA 94087	EDUC. SOFTWARE

FIRM	ADDRESS	PRODUCTS
LOOKING GLASS MICRO	P.O. BOX 508, LOVELAND, CO 80537	INTERFACE 1 $85
LUCK SOFTWARE	1160 NIBLICK ROAD, PASO ROBLES, CA 93446	MONEY PROCESSOR
MACROTRONICS	1125 N. GOLDEN STATE, TURLOCK, CA 95380	SCREEN DUMP
MANNESMAN TALLY	8301 S. 180TH ST., KENT, WA 98232	PRINTERS
MASTER CONTROL SOFTWARE	P.O. BOX 26714, SALT LAKE CITY, UT 84126	GAMES
MATTEL M-NETWORK	5150 W. ROSECRANS, HAWTHORNE, CA 90059	
MAXIMUS	6723 WHITTIER AVE., MCLEAN, VA 22101	STORIES
MED SYSTEMS	P.O. BOX 3558, CHAPEL HILL, NC 27514	GAMES
MICRO SYSTEMS EXCHNG.	P.O. BOX 403, CONCORD, CA 94524	48K FOR 400-100
MICROTRONICS	BOX 8894, FR COLLINS, CO 80525	KEYBOARD-400
MICROGRAPHICIMAGE	9550 FOREST LANE #627, DALLAS, TX 75243	GAMES
MICROBITS PERIPHERAL	434 W. FIRST ST., ALBANY, OR 97321	MODEM/I/F
MICROPROSE SOFTWARE	10616 BEAVER DAM RD., HUNT VALLEY, MD 21030	GAMES
MICROPHERAL CORP.	2565 152ND AVE. NW, REDMOND, WA 98052	MODEM/I/F
MICROTEK	9514 CHESAPEAKE DR., SAN DIEGO, CA 92123	16K-32K RAM
MICRO MAINFRAME	11325 SUNRISE CIR. # E, RANCHO CORDVA, CA 95670	DRIVES I/F
MILES COMPUTING	7136 HASKELL AVE., #204, VAN NUYS, CA 91406	PAYROLL/ACCNTG
MILLIKEN PUBLISHING	1100 RESEARCH BLVD., ST. LOUIS, MO 63132	
MIND MOVERS	4286 REDWOOD HWY. #245, SAN RAFAEL, CA 94903	SECRET FORMULA
MMG MICRO SOFTWARE	P. O.BOX 131, MARLBORO, NJ 07746	GAMES/UTILS
MONARCH DATA SYSTEMS	P.O. BOX 207, COCHITUATE, MA 01778	UTILITIES
MOS TECHNOLOGY	950 RITTENHOUSE RD., NORRISTOWN, PA 19401	6502 CHIPS
MOSAIC ELECTRONICS	P.O. BOX 748, OREGON CITY, OR 97045	RAM
MOUNTAIN VIEW PRESS	P.O. BOX 4656, MOUNTAIN VIEW, CA 94040	FORTH PRODUCTS
MYOTIS SYSTEMS	P.O. BOX 13568, TUCSON, AZ 85732	ROBOT ARM
NEC	1401 ESTES AVE., ELK GROVE VILLAGE, IL 60007	PRINTERS
NEWELL SYSTEMS	3340 NOTTINGHAM LANE, PLANO, TX	FAST CHIP
NEWPORT CONTROLS	15425 LOS GATOS BLVD., LOS GATOS, CA 95030	PROSTICK II
NOVIN	P.O. BOX 22889, SEATTLE, WA 98122	SOFTWARE
NUFEKOP	21255 HWY. 62, SHADY GROVE, OR 97539	GAMES
ODESTA	930 PINTER, EVANSTON, IL 60202	CHESS
ON LINE	10944 NORTH MAY, OKLAHOMA CITY, OK 73120	FIN. WIZ
ORIGIN SYSTEMS	18100 UPPER BAY RD., #200, HOUSTON, TX 77258	GAMES
OSBORNE/MCGRAW HILL	630 BANCROFT, BERKELEY, CA 94710	BOOKS
OSS	1173D SARATOGA/SV RD., SAN JOSE, CA 95129	SOFTWARE, OS
PARKER BROS.	BEVERLY, MA 01915	GAMES
PENGUIN SOFTWARE	830 FOURTH AVE., GENEVA, IL 60134	GAMES
PERCOM	11220 PAGEMILL RD., DALLAS, TX 75243	DRIVES
PINK NOISE STUDIOS	P.O. BOX 785, CROCKETT, CA 94525	PNS FORTH
PRECISION SOFTWARE	1173D S. SUNNYVALE-SARATOGA RD., SAN JOSE, CA 95129	LANGUAGES
PROGRAM DESIGN INC	95 E. PUTNAM AVE., GREENWICH, CT 06830	SOFTWARE
PROGRAMMERS INSTITUTE	P.O. BOX 3191, CHAPEL HILL, NC 27514	LT PEN, SOFTWARE
QUALITY SOFTWARE	6660 RESEDA BLVD. #105, RESEDA, CA 91355	SOFTWARE
QUINTECH	1271 DUNDEE RD. #44B, BUFFALO GROVE, IL 60090	UTILITIES
R.O.M.	BOX 252, MAPLE RIDGE, BC CAN V2X 7G1	ATARI MAGAZINE
RADICAL SYSTEMS	2002 COLICE RD., SE, HUNTSVILLE, AL 35801	EPROM BURNER
RANA SYSTEMS	21300 SUPERIOR ST., CHATSWORTH, CA 90746	DRIVES
RANTOM	P.O. BOX 5480, AVON, CO 81620	UTILITIES
RCE	536 NE "E" ST., GRANTS PASS, OR 97526	400 KEYBOARD
RESTON PUBLISHING	11480 SUNSET HILLS RD., RESTON, VA 22090	BOOKS
ROMOX	501 VANDELL WAY, CAMPBELL, CA 95008	GAMES
SANTA CRUZ EDU. SOFTWARE	5425 JIGGER, SOQUEL, CA 95073	TUTORIALS
SAR-AN COMPUTER PROD	12 SCAMRIDGE CURVE, BUFFALO, NY 14221	MEMORY/CARTS
SAS ELECTRONICS	3091 NORTH BAY DR., NORTH BEND, OR 97459	
SCI-TOR	710 LAKEWAY, #290, SUNNYVALE, CA 94086	SOFTWARE
SCREEN SONIC	14416 S. OUTER 40, CHESTERFIELD, MS 63017	KEYBOARD
SCREENPLAY	P.O. BOX 3558, CHAPEL HILL, NC 27514	ADV. GAMES
SENTIENT SOFTWARE	P.O. BOX 4929, ASPEN, CO 81612	
SIERRA ON-LINE	3675 MUDGE RANCH RD., COARSEGOLD, CA 93614	SOFTWARE
SIM COMPUTER PRODUCTS	1100 E. HECTOR ST., WHITEMARSH, PA 19428	CALC PRODUCTS
SIRIUS SOFTWARE	10364 ROCKINGHAM DR., SACRAMENTO, CA 95827	GAMES
SOFT SIDE PUBL.	6 SOUTH STREET, MILFORD, NH 03055	MAGAZINE
SOFT UNLIMITED	3546 PILGRIM LN., PLYMOUTH, MN 55451	DISKEDIT
SOFTSYNC, INC.	14 EAST 34TH ST., NEW YORK, NY 10016	SOFTWARE
SOURCE TELECOMPUTING	1616 ANDERSON RD., MCLEAN, VA 22102	
SOUTHERN SOFTWARE	1879 RUFFNER RD, BOX 66398, BIRMINGHAM, AL 35210	THE CHIP
SPECTRAVISION	39 W. 37TH ST., NEW YORK, NY 10018	SURE SHOT STIK
SPINNAKER SOFTWARE	215 FIRST ST., CAMBRIDGE, MA 02142	EDUCATIONAL

FIRM	ADDRESS	PRODUCTS
STAR MICRONICS	#3 OLDFIELD, IRVINE, CA 92714	GEMINI PRINTERS
STARPLEX ELECTRONICS	E. 23301 MISSION, LIBERTY LAKE, WA 99019	
STRATEGIC SIMULATION	883 STIERLIN ROAD #A-200, MT. VW., CA 94043	WAR GAMES
SUBLOGIC	713 EDGEBROOK DR., CHAMPAIGN, IL 61820	GAMES
SUPERWARE	2028 KINGSHOUSE, SILVER SPRINGS, MD 20904	XBASIC EXTENDER
SWIFTY SOFTWARE	64 BROADHOLLOW RD., MELVILLE, NY 11747	SOFTWARE
SWP MICROCOMPUTER PR	2500 E. RANDOL MILL #125, ARLINGTON, TX 76011	ATR8000
SYNAPSE SOFTWARE	5221 CENTRAL AVE. #200, RICHMOND, CA 94804	SOFTWARE
SYNCHRO	742 HAMPSHIRE RD., #C, WESTLAKE VILLAGE, CA 91361	
SYNERGISTICS SOFTWARE	830 N. RIVERSIDE, #201, RENTON, WA 98055	
T.H.E.S.I.S.	P.O. BOX 147, GARDEN CITY, NJ 48135	GAMES
TARA COMPUTER PRODS.	107 DELAWARE, STATLER BLVD., BUFFALO, NY 14202	400 KEYBOARD
TELECOMMUNICATION	1123 OAKFAIR LANE, HARBOR CITY, CA 90710	
1TG PRODUCTS	1105 SUMMIT #110, PLANO, TX 75074	TRACK BALLS
THORN EMI VIDEO	1370 AVENUE OF THE AMERICAS, NY, NY 10019	GAMES
TINY TEK INC.	P.O. BOX 12609, DALLAS, TX 75225	MEMORY
TMQ	82 FOXHILL DR., BUFFALO GROVE, IL 60090	FILE FAX-DBMS
TRAK MICROCOMPUTER	1511 OGDEN AVE., DOWNERS GROVE, IL 60515	DISK DRIVES
TRANSTAR	P.O. BOX C96975, BELLEVUE, WA 98009	PRINTERS
TRONIX PUBLISHING	8925 S. LA CIENEGA, INGLEWOOD, CA 90301	SOFTWARE
UNITED SOFTWARE/AM	750 THIRD AVE., NEW YORK, NY 10017	SOFTWARE
UNIVERSITY SOFTWARE	BOX 4544, STANFORD, CA 94305	CASSETTE SOFTWARE
UTOPIA SOFTWARE	58 MILLAY RD., MORGANVILLE, NJ 07751	PINHEAD
VALPAR INT'L	3801 E. 34TH, TUCSON, AZ 85713	VALFORTH
VERSA COMPUTING	3541 OLD CONEJO RD. #104, NEWBURY PARK, CA 91320	GRAPHICS TABLET
VISICORP	2895 ZANKER ROAD, SAN JOSE, CA 95134	BUSINESS SOFTWARE
VOTRAX	500 STEPHENSON WAY, TROY, MI 48084	VOICE BOX
VOYAGER SOFTWARE	P.O. BOX 1126, BURLINGAME, CA 94010	SOFTWARE
WALLING SOFTWARE	7755 E. EVANS #400, SCOTTSDALE, AZ 85260	APROM CARTRIDGE
WICO	6400 W. GROSSE PT. RD., NILES, IL 60648	GAME CONTROLLER
XLENT SOFTWARE	P.O. BOX 5228, SPRINGFIELD, VA 22150	GRAPHICS DUMP
ZIRCON INTL	475 VANNELL WAY, CAMPBELL, CA 95008	
ZORK USERS GROUP	P.O. BOX 20923, MILWAUKEE, WI 53220	

ABOUT THE AUTHOR

Dave Mentley was president of A.B.A.C.U.S., a very active Atari users group in San Francisco, CA, for over a year and a half. As president of the group, he has had the opportunity to review thousands of Atari newsletter articles and has worked with many major software and publishing companies. Despite warnings that it was impossible, he has written two books using the Atari computer: *Letter Perfect*, and *The Bit 3 80 Column Board*. He hardly ever plays games on his Atari 800. When he is not computing, he does marketing research in the area of personal computers and he will be doing a lot of backpacking in the Sierras.

ABCs of Atari Computers

All the programs listed in ABCs of Atari Computers on one diskette! Save yourself hours of typing and de-bugging.

Only $15.95 with this coupon!

Available only to readers of this book. Send $15.95 plus $2.00 shipping and handling (California residents add 6½% sales tax) to:

DATAMOST™ INC

20660 Nordhoff Street, Chatsworth, CA 91311-6152

Please RUSH me ABCs of Atari Computers diskette.
Please include $15.95 plus $2.00 for shipping and handling (California residents add $1.04 for sales tax).

Name _____

Address _____

City _____ State _____ Zip _____

Phone ( ) _____

☐ Check/Money Order
☐ Master Card ☐ Visa

Credit Card # _____

Expires _____

Signature _____

$14.95

ABCs of Atari Computers

A complete compendium for Atari computerists, from beginners to experts

Arranged in alphabetical order, the ABCs OF ATARI COMPUTERS addresses a broad spectrum of Atari topics. From ACCUMULATORS to CODE CONVERSIONS to SCREEN MEMORY, this fantastic reference book contains the answers to your programming questions.

All definitions are written in clear, concise terms, so everyone from the beginner to the expert can understand. Many of the definitions are accompanied by charts and programs, making this THE complete reference on Atari Home Computers. COLLISION REGISTERS, SECTORY TYPES, TOKENIZING and PADDLE TRIGGERS are just a few of the hundreds of topics covered in detail.

The ABCs OF ATARI COMPUTERS include cable charts for printers and modems, so you can make your own for half the price! The author also lists software packages for the Atari, so you know what each does before spending your hard earned money.

Whether you're the proud owner of a new Atari 800XL, or have been programming since the Atari 800 was released, this comprehensive reference deserves a special place next to your computer!

ISBN 0-88190-367-1

DATAMOST

20660 Nordhoff Street, Chatsworth, CA 91311-6152
(818) 709-1202