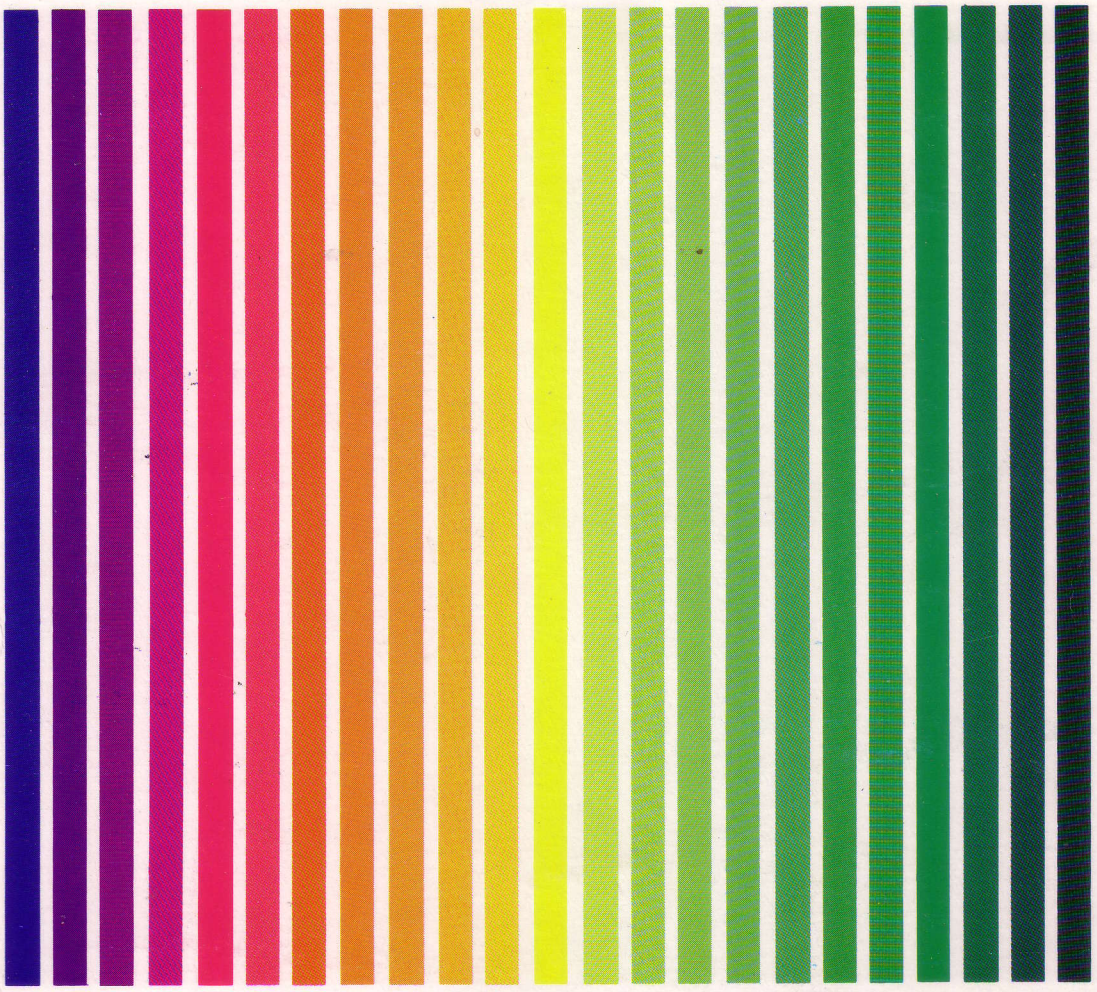
APX

ATARI® PROGRAM EXCHANGE



User-Written Software for ATARI Home Computers

JOHN R. POWERS III JUNE 1982

COSMATIC ATARI DEVELOPMENT PACKAGE

DISKETTE (APX-20051)
REQUIRES: 32K RAM

COSMATIC ATARI DEVELOPMENT PACKAGE

by

John R. Powers, III

TRADEMARKS OF ATARI

The following are trademarks of Atari, Inc.

ATARI®
ATARI 400™ Home Computer
ATARI 800™ Home Computer
ATARI 410™ Program Recorder
ATARI 810™ Disk Drive
ATARI 820™ 40-Column Printer
ATARI 822™ Thermal Printer
ATARI 825™ 80-Column Printer
ATARI 830™ Acoustic Modem
ATARI 850™ Interface Module

****

Distributed by

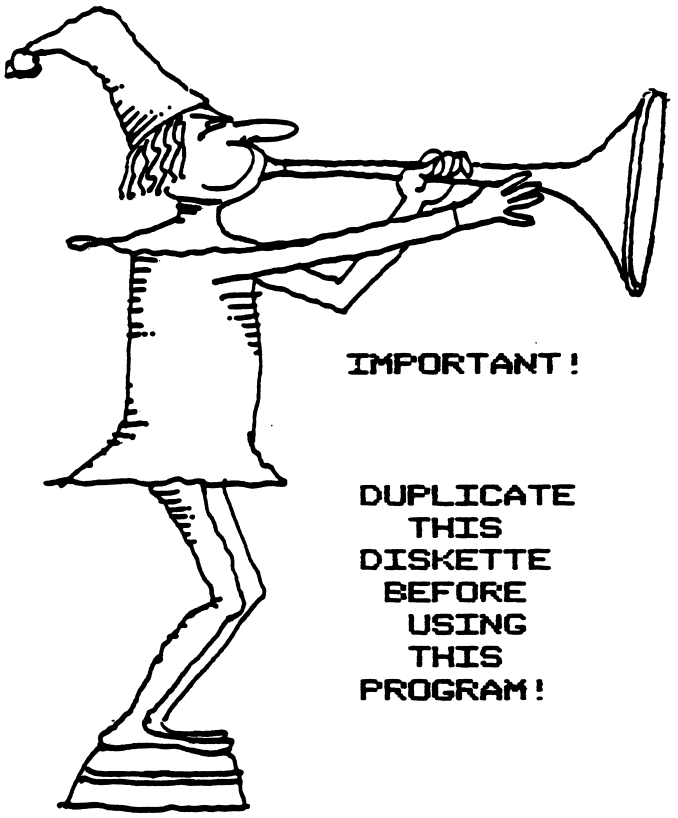
The ATARI Program Exchange
P. O. Box 427
155 Moffett Park Drive, B-1
Sunnyvale, CA 94086

To request an APX Software Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/745-5535.

****



This APX diskette is unnotched to protect the software against accidental erasure. However, this protection also prevents a program from storing information on the diskette. The program you've purchased involves storing information. Therefore, before you can use the program, you must duplicate the contents of the diskette onto a notched diskette that doesn't have a write-protect tab covering the notch.

To duplicate the diskette, call the Disk Operating System (DOS) menu and select option J, Duplicate Disk. You can use this option with a single disk drive by manually swapping source (the APX diskette) and destination (a notched diskette) until the duplication process is complete. You can also use this option with multiple disk drive systems by inserting source and destination diskettes in two separate drives and letting the duplication process proceed automatically. (Note. This option copies sector by sector. Therefore, when the duplication is complete, any files previously stored on the destination diskette will have been destroyed.)

PART I: THE COSMATIC ATARI

An 1802 Cross-Assembler

TABLE OF CONTENTS

INTRODUCTION

OVERVIEW

The COSMATIC ATARI is a cross-assembler for 1802 assembly language. The program takes your 1802 assembly language program and produces hex machine instructions and data. The 1802 microprocessor is used in many popular hobby computers and games because of its low cost and ease of implementation. Example systems include:

RCA COSMAC Development System
VIP Game Computer by RCA
Studio II Programmable Game by RCA
Hughes Development System
ELF II Hobby Computer by Netronics
The Anything Board by Netronics
Super EIF by Quest Electronics
CyberVision Home Computer
Popular Electronics' COSMAC ELF Series

This cross-assembler will let you write 1802 programs in assembly language using all the power of symbolic references, mnemonics, and operand expression processing. You will have the same capabilities of systems costing over $5,000.

FEATURES OF THE COSMATIC ATARI INCLUDE:

- Superset of both the RCA CRA Assembler and the Hughes HMDS Assembler
- The ability to add extensions
- Full operand expression evaluation with nesting to 10 levels
- Source File chaining option
- Sorted label table output
- Assembly at 20-to-30 source statements per minute
- Unique two-step methodology permitting the use of the intermediate file as backup
- Output fully compatible with UT4 and UT20 hex loading utilities
- Creation of source code file with or without line numbers for input to the assembler
- Sample 1802 source files to demonstrate and test features

SYSTEM REQUIREMENTS

The COSMATIC ATARI requires the following:

An ATARI Computer with 32K or more of memory
ATARI BASIC Language Cartridge
One or more ATARI 810 Disk Drives

An ATARI printer is optional.

THE SCOPE OF THIS MANUAL

This manual instructs you in the use of the COSMATIC ATARI program. It is not an instruction guide to 1802 programming.

Information is also provided on how the COSMATIC ATARI works as well as tips on how to add extensions to the implementation. The sample 1802 source files on the diskette are included to give you plenty of examples.

One note on terminology used in this manual - since the primary output of the COSMATIC ATARI is hex machine instructions and data, BASIC programmers may be confused by the use of byte-related terms. Here are some definitions:

Character	A symbol representing a digit or letter. Examples are "0-9" and "A-F".
Byte	An 8-bit quantity represented by two characters. Examples are "00,01,02,..., FF". The terms "byte" and "hex byte" are used interchangeably.
Two Bytes	A 16-bit quantity represented by four characters. Used interchangeably with "a pair of bytes" or "Four-digit hex number".

All input to the COSMATIC ATARI is in the form of characters representing decimal and hexidecimal (hex) digits and alphabetic characters. All assembled output from the cosmatic ATARI is in the form of characters representing hexidecimal digits.

USING THE COSMATIC ATARI

GET READY AND GO

Prepare your 1802 assembly language source program and store it on a diskette file. Source language preparation can be done with the ATARI Assembler-Editor cartridge, the ATARI Program-Text Editor™ (available through APX), or similar text editors. If your 1802 source file has line numbers, such as those produced by the Assembler-Editor cartridge, the COSMATIC ATARI will automatically skip over them. In every case, the last line in the source file must contain the "END" directive.

You may also enter source lines directly to the COSMATIC ATARI. In this case, your source file name is "E:" for the screen editor.

Insert the program diskette and RUN "D:ASM" to load and execute the COSMATIC ATARI. The first screen will show the copyright message and a message indicating that the program is preparing data. This will take a few seconds.

When the program is ready, you will be prompted for the source input file name. Include the device code (e.g., D:for disk drive). You will then be prompted for the hex output file name. Include the device code. Alternatives to disk file output include "S:" for the screen and "P:" for the printer.

Following the first step of reading through the source code, the label table will be displayed in alphabetical sequence. You will then be prompted to press RETURN so that step two can start. Following step two, you will be prompted to press RETURN to restart the program.

FILE USAGE

You will be defining the file names for input to and output from the COSMATIC ATARI. In addition, the program will create a temporary file defined as "D:COSMAC.TMP". Be sure your diskette in drive one does not have the write protect tab in place.

Your APX diskette contains many files for you to use. Only one file is essential, that is "ASM", the program in BASIC SAVEd format. The other files are 1802 source files intended to demonstrate the capabilities of the COSMATIC ATARI. They are:

FILE NAME	DESCRIPTION
TPAR01.LST	Test of Parsing
TOND01.LST	Test of Operand Expressions
TDAT01.LST	Test of Data Statements and END
TDAT02.LST	Evoked by TDAT01.LST
TFWD01.LST	Test of Forward References
TOPC01.LST	Test of all op-codes
TMSG01.LST	Demonstration of program messages
CLICK.LST	Sample Feature mix
FECHR.LST	Simple macro routine

All the 1802 source files have line numbers so that you can ENTER them into the Assembler-Editor Cartridge. Or you may simply COPY the file to your printer (P:) or screen (S:) using the DOS 2.0S copy option.

You are encouraged to look at the sample files to see all the things the COSMATIC ATARI can do for you. It is also an excellent way to see examples of source line syntax. The samples are deliberately ragged to demonstrate the flexibility of the cross-assembler.

An additional file on your diskette is "ASMP". That is the printer version of "ASM", in BASIC SAVEd format. "ASMP" may be used in place of "ASM" if you have an 80-column printer.

MESSAGES

There are only a few messages generated by the COSMATIC ATARI. Extensive error checking would make the assembly uselessly slow. The program will attempt to assemble no matter what. You will need to inspect the hex output if you suspect the accuracy of your source code syntax.

The following messages are possible:

Function?	Expecting function code preceding a left parenthesis.
Re-defined Label	Permissive warning message. The label has been redefined, discarding the previous value.
Unknown Op-Code	The op-code mnemonic was not found in the op-code table.
Error opening file	Error occurred while attempting to open file.
n forward references	Informative message only.
Bad hex	A character other than 0-9 or A-F occurred in a hex number.
n unsatisfied references	References were made to undefined labels. A *; line will be generated in the hex load file for each unsatisfied reference.

A line such as the following may occur in the final, step two output:

```
XXXX      ;* source line
```

Where XXXX is a hex address and the "source line" is a duplicate of a line of your source. This line occurs after the second step when a label in the source line was never defined in the program. This is an unsatisfied reference and the ";*" is the mark for that condition.

The above line may also occur in the intermediate file output of step one. That is okay as it is assumed to be a forward reference which will be satisfied in step two.

THE INTERNAL WORKINGS

A TWO-STEP ASSEMBLER

Most assemblers read the source file twice, once to collect labels and addresses and then to generate the hex output. That is a two-pass approach.

This assembler uses a two-step approach. The first step generates an intermediate file which, in turn, is the input for the second step. Each file is read only once.

The first step collects labels and addresses and attempts to generate hex output. If there are references to labels which have not yet been defined, the generation of hex code for these cases is deferred until the second step. These are called forward references. If the source file contains no forward references then there are no source statements deferred until the second step. In these cases, the intermediate file is almost identical to the second-step output file.

The approach is basically recursive. The source line processor treats each step identically. This means that forward references unsatisfied in the second step are passed on to the output file for imaginary third-step processing. An error message is generated when this occurs.

The net effect is a very efficient assembly algorithm which helps to compensate for the natural slowness of BASIC interpretation.

Within each source line, the operational steps are as follows:

1. Parse line into
   a) label
   b) op-code
   c) operand
   d) comment
   using blanks as delimiters

2. Store label and address, if present

3. Lookup op-code or directive, if present

4. Evaluate operand, if present

5. Store generated hex

6. Go to (4) if comma present, else next source line

The lookup of op-codes and labels uses a binary search in a sorted list. The insertion of newly-defined labels is done so as to maintain the sorted list. A binary search is much faster than a linear search, helping to compensate for the slowness of BASIC interpretation.

OPCODES

The assembly of op-code mnemonics into hex instructions is table-driven making changes and extensions easy. You may, for example, wish to add new op-codes to incorporate the expanded 1800-microprocessor family.

The op-code table is located in lines 30400 thru 30999. Each op-code field consists of seven characters as follows:

Char 1-4	Op-code mnemonic
5-6	Machine-code hex
7	Type

How the machine-code hex is assembled depends on the type identified in character 7. The types are:

0 The machine-code hex is used exactly as shown with no additional hex bytes. Any operand data is ignored.

1 The machine-code hex is used exactly as shown followed by one additional hex byte. An operand is required. Only the lower-order 8-bit byte portion of the evaluated operand is used.

2 The machine-code hex is used exactly as shown followed by two additional hex bytes. An operand is required. The full, 16-bit two-byte evaluated operand is used.

3 Only the higher-order 4-bit nybble (half a byte) portion of the machine-code hex is used. An operand is required. The lower-order 4-bit nybble portion of the machine-code hex is formed from the lowest-order 4-bit nybble portion of the evaluated operand.

4 Directive or special case. The machine-code hex is optional and is entirely a function of the unique directive processing (see below).

All op-codes and directives must be in ascending alphabetic sequence in the table. They must be unique in name, but need not be unique in function. The end of the list must be marked with an asterisk terminator.

The op-code and directive table will be stored in memory by the program, and operations will be automatically adjusted for its size.

DIRECTIVES

Directive mnemonics are located in the same table as the op-codes (see above). The "Type" portion of the directive table data determines processing. The op-code section describes how to add new mnemonics.

Directive processing is located in lines 18000 thru 19999. Specific directive lines are as follows:

Directive	Type	Lines
(not assigned)	4	18100 - 18199
ORG	5	18200 - 18299
PAGE	6	18300 - 18399
INP	7	18400 - 18499
(not assigned)	8	18500 - 18599
END	9	18600 - 18799

Types 4 and 8 are available for your use. The existing directives can serve as examples. The following variables may be useful:

Variable	Description
D	The decimal value of the evaluated operand.
PC	The decimal current program address counter, the value of the asterisk (*) symbol.
TLEN	The decimal count of the number of bytes assembled for this statement.
OPHX$	The op-code hex set by your routine. Length is zero upon entry, may be zero, one, or two characters upon exit. (Remember, it takes two characters to represent one hex byte)
OPHX$	The operand hex set by your routine. Length is zero upon entry, may be from zero to 80 characters upon exit.

The following subroutines may also be helpful:

GOSUB 28800	Converts decimal D to its 4-character, two-hex-byte string representations H$.
GOSUB 28900	Converts the 4-character, two-hex-byte string representation, H$, to its decimal representation, D.

EXPRESSION OPERATORS

You may define your own operand expression operators if you wish. The allocated operators are as follows:

Char.	ASC (char)	WOP	Description
%	37	1	not defined
&	38	2	not defined
'	39	3	not defined
(	40	4	reserved
)	41	5	reserved
*	42	6	reserved
+	43	7	add
,	44	8	reserved
-	45	9	subtract

Operator execution is localized in lines 14220 thru 14240. For example:

```
14230 If WOP= 7 THEN WAC = WAC + D
14232 If WOP= 9 THEN WAC = WAC - D
```

The variable "WAC" is the expression accumulator and "D" is the value of the previously-evaluated portion of the expression. To review, expressions are evaluated from inner-to-outer parentheses levels and from left-to-right at the same level.

It is very easy to add multiplication and division to your program. You simply decide which symbols you want to assign out of those available. The operator table above shows three possibilities: %, &, and ,. The program operator codes are 1,2, and 3 respectively. If you were to use the first two, simply add the following two lines:

```
14234 If WOP= 1 THEN WAC=WAC/D
14236 If WOP= 2 THEN WAC=WAC*D
```

The nesting and evaluation of constants and variables within the expression is handled for you. All you have to do is write your source line expressions using the two new operators as desired.

LABEL TABLE

All labels are dynamically sorted and stored in array LAB$ as they are encountered and evaluated. The dimensioned length of this array determines the maximum number of labels. There is no overflow checking.

The array LAB$ also holds all the op-codes and directives. The partitioning of the array is handled automatically by the program. In the released version of the program, 588 characters are used for op-codes and directives. This leaves 1012 characters for labels. Since the 4-character hex representation of the label's value is stored along with the 6-character label, 10 characters must be allowed for each label. The result is room for 101 labels. Increase the dimension of LAB$ in line 29110 by 10 for each additional label. The dimension can be expanded to about 5000 characters ((5000-588)/10= 441 labels) and still run on a 32K system. There is no performance penalty for dimensioned but unused label storage.

INTERMEDIATE FILE

The file "COSMAC.TMP" is used by the program to store the results of step one. This file may be discarded after step two. However, it may be considered a backup file since its contents are stored in hex load format. The hex load format and file usage are described in other sections.

To change the temporary file name, only line 27080 need be altered. This line also defines the device upon which the temporary file is resident (default is "D:"). The released version contains:

```
27080 FD2$ = "D:COSMAC.TMP"
```

HEX LOAD FILE

The output of the assembler is a hex load file. This is a character file with one record of hex digits per record of source. Source lines which do not generate hex digits (e.g., comments and various directives) have no corresponding output.

The hex load file format is as follows:

Char	Description
1-4	hex load address
5	blank
6-end	hex literals to load terminated by a semicolon (;)

Comments may follow the semicolon up to the end-of-line.

The intermediate file (COSMAC.TMP) is also in hex load format. Source lines deferred for step two are made to look like comments in the intermediate file output.

SPECIFICATIONS

TYPES OF SOURCE STATEMENTS

instructions
directives
comments

INSTRUCTIONS AND DIRECTIVE FORMAT

LABEL OP-CODE OPERAND ..COMMENT

Fields are separated by one or more blanks. The label field must start in column one or two or, if not present, at least two blanks must precede the op-code field. The comment, if present, must be preceded by two consecutive periods (..). A RETURN terminates the source line.

Options: (1) The label field may take the form "LABEL:". In this case the label may start in any column and need not be separated from the op-code field by any blanks.

(2) The comment, if present, may be preceded by a semicolon (;) rather than consecutive periods (..).

LABEL

1-6 alphanumeric characters; first character must be uppercase alphabetic.

OP-CODE

An 1802 instructions mnemonic or directive

OPERAND

Register data

R0, R1, R2, ..., RF
0, 1, 2, ..., 9, 10, ..., 15
#00, #01, ..., #0F

Single-byte data

For immediate instructions and short branches
range 0 to 255, #00 to #FF

Address Data

For long branches
range 0 to 65535, #0000 to #FFFF

I/O device code

Integer from 1 to 7

COMMENTS

Must be preceded by semicolon (;)
or consecutive periods (..).
May begin in any column.

DECIMAL DATA

No prefix

HEX DATA

Must be preceded by pound sign (#).
Must be from one to four hex digits.
Assumed to be right-justified in a four-hex-digit field.
Option: may be preceded by a dollar sign ($) rather than a pound sign.

CURRENT PROGRAM ADDRESS COUNTER

Asterisk (*) is the symbol.
The address of the first byte of the
instruction or data statement in
which it occurs.
Used in operand field only.

LABEL VALUES (EQUATES)

LABEL=OPERAND

Creates the label and stores the value of the operand.
Any valid operand expression may be used.
Evaluated to a two-byte, 16-bit value.
Label may start in any column.
There must be no embedded blanks.

OPERAND EXPRESSIONS

May be used for any operand.
Operators:

+ addition
- subtraction

Functions:

H(expr)	High byte (8-bits)
L(expr)	Low byte (8-bits)
A(expr)	Two bytes (16-bits)
B(expr)	Two bytes, reversed (16-bits)

Rules:

Multiple operators may be used in any sequence.
Parentheses may be nested to ten levels.
Expressions are evaluated first from the innermost parentheses,
   then from left-to-right within a parentheses pair.
The asterisk (*) curent program counter may be mixed in the
   expression.
Decimal and hex values may be mixed.
Labels and constants may be mixed.

Options (1) A.0(expr) may be used for L(expr).
        (2) A.1(expr) may be used for H(expr).

DIRECTIVES

ORG OPERAND

Sets the current program address counter to the value of the operand.
Forward references not supported.
Any valid operand expression may be used.
Evaluated as a 16-bit address.

PAGE

Increments the current program address counter to the beginning of the
   next memory page.
Equivalent to H(L(B(*))+1)
           or H(*+#0100).

LABEL DC OPERAND, OPERAND,...

Defines memory-resident constants.
Label, if used, references first byte.
Each operator is evaluated to one byte unless the outermost function
   is A(exp) or B(exp), which is evaluated to two, 16-bit bytes.
Each one-or two-byte element in an operand list must be separated
   by a comma.
Any valid operand expression may be used.

END chain-file-name

Must terminate a source module.
The chain-file-name, if present,
  defines the next source file to be assembled without operator intervention. All labels remain valid for the chain-file. The current address program counter remains current.
The device id should not be included in the chain-file-name. The COSMATIC ATARI will use the same device that the previous source file was on.

DATA STATEMENTS

Occasionally it is convenient to assemble hex data along with the hex machine instructions. Examples include data tables, table indices, display pixels, parameter passing, and macros. There are two ways to assemble hex data:

1. The DC op-code
2. the comma (,) delimiter

The DC op-code takes the form:

```
LABEL DC OPERAND
```

The label is optional. The operand is a standard operand expression and is evaluated to a 2-byte 16-bit quantity. The length of the stored hex depends on the outermost expression function. The possibilities are:

Statement	Result
DC #12	12
DC #ABCD	CD
DC A(#12)	0012
DC A(#ABCD)	ABCD
DC L(#1234)	34
DC H(#ABCD)	AB
DC B(#1234)	3412

Another method is to use the comma (,) delimiter. Any operand that follows a comma (note, no blanks!) will be assembled as a hex data constant. More than one comma may be used in a single source line. Examples are:

Statement	Result	
TIME ,4	04	
,#12,A(#ABCD)	12ABCD	
SEP CALL,A(SUBR)	D41001	(CALL=4, SUBR=#1001)

Statement	Result
LDI #33,#44	3344
,L(#1234)	34
,A(#1234),A(#5678)	12345678
,#1234,#5678	3478

Only one or two bytes (8 or 16 bits) may be assembled for each comma. Examples of the uses for data statements are as follows:

```
.. TABLE INDEX
DIGS:  DC    A.0(DIG0)
       DC    A.0(DIG1)
       DC    A.0(DIG2)
.. DISPLAY PIXELS
DIGE:  DC    #F0,#80
DIGF:  DC    #F0,#80
DIGF:  DC    #F0,#80,#80,#80
.. PARAMETER PASSING
       SEP   R4,A(SUBR1),#F4
       LDI   A.0(SUBR2)
       PLO   R1
       SEP   R1,#20,#00
.. MACROS
       FECH=#D7
       ,FECH,#01
```

USER NOTES

To assign values to symbolic names use

```
LABEL=OPERAND
```

To set aside memory storage use

```
ORG *+n
```

Where n = number of storage bytes

To mask the lower 8-bits of a 16-bit address use

```
L(#abcd) yields #00cd
```

To mask the higher 8-bits of a 16-bit address use

```
H(#abcd) yields #ab00 if not outermost function
```

H(#abcd) yields #ab if outermost function and used for a single-byte op-code

COSMATIC ATARI 1802 MNEMONICS

Syntax	Machine System	Type	Name
ADC	74	0	Add with carry
ADCI expr	7C	1	Add with carry immediate
ADD	F4	0	Add
ADI expr	FC	1	ADD immediate
AND	F2	0	And
ANI expr	FA	1	And immediate
B1 expr	34	1	Short branch if EF1-1
B2 expr	35	1	Short branch if EF2-1
B3 expr	36	1	Short branch if EF3=1
B4 expr	37	1	Short branch if EF4=1
BDF expr	33	1	Short branch if DF=1
BN1 expr	3C	1	Short branch if EF1=0
BN2 expr	3D	1	Short branch if EF2=0
BN3 expr	3E	1	Short branch if EF3=0
BN4 expr	3F	1	Short branch if EF4=0
BNF expr	3B	1	Short branch if DF=0
BNQ expr	39	1	Short branch if Q=0
BNZ expr	3A	1	Short branch if D not 0
BQ expr	31	1	Short branch if Q=1
BR expr	30	1	Short branch
BZ expr	32	1	Short branch if D=0
DEC reg	20	3	Decrement register
DIS	71	0	Disable
END	n/a	9	Directive, End Source
GHI reg	90	3	Get high register
GLD reg	80	3	Get low register
IDL	00	0	Idle
INC reg	10	3	Increment register
INP expr	60	7	Input
IRX	60	0	Increment register X
KBDF expr	C3	2	Long branch if DF=1
LBNF expr	CB	2	Long branch if DF=0
LBNQ expr	C9	2	Long branch if Q-0
LBNZ expr	CA	2	Long branch if D not 0
LBQ expr	C1	2	Long branch if Q=1
LBR expr	C0	2	Long branch
LBZ expr	C2	2	Long branch if D=0
LDA reg	40	3	Load advance via register
LDI expr	F8	1	Load immediate
LDN reg	00	3	Load via register
LDX	F0	0	Load via X
LDXA	72	0	Load advance via X
LSDF	CF	0	Long skip if DF=1
LSIE	CC	0	Long skip if IE=1
LSKP (expr)	C8	2	Long skip

Syntax	Machine System	Type	Name
LSNF	C7	0	Long skip if DF=0
LSNQ	C5	0	Long skip if Q=0
LSNZ	C6	0	Long skip if D not 0
LSQ	CD	0	Long skip if D=0
MARK	79	0	Push X,P to stack
NBR (expr)	38	1	No short branch
NLBR expr	C8	2	No long branch
NOP	C4	0	No operation
OR	F1	0	Or
ORG	n/a	5	Directive, Define origin
ORI expr	F9	1	Or immediate
OUT expr	60	3	Output
PAGE	n/a	6	Directive, Next memory page
PHI reg	B0	3	Put high register
PLO reg	A0	3	Put low register
REQ	7A	0	Reset Q
RET	70	0	Return
SAV	78	0	Save
SD	F5	0	Subtract D
SDB	75	0	Subtract D with borrow
SDBI expr	7D	1	Subtract D with borrow immediate
SDI expr	FD	1	Subtract D immediate
SEP reg	D0	3	Set P
SEQ	7B	0	Set Q
SEX reg	E0	3	Set X to register
SHL	FE	0	Shift left
SHLC	7E	0	Shift left with carry
SHR	F6	0	Shift right
SHRC	76	0	Shift right with carry
SKP (expr)	38	1	Short skip
SM	F7	0	Subtract memory
SMB	77	0	Subtract memory with borrow
SMBI expr	7F	1	Subtract memory immed with borrow
SMI expr	FF	1	Subtract memory immediate
STR reg	50	3	Store via register
STXD	73	0	Store via X and decrement
XOR	F3	0	Exclusive or
XRI	FB	1	Exclusive or immediate

LEGEND

expr	-	expression evaluated to a 16-bit, 2-byte quantity; assembled into 1 or 2 bytes depending on type
req	-	register expression evaluated to a 16-bit, 2 byte quantity; lowest (least significant) 4-bits masked onto op-code machine stem.

Legend (Continued)

Type 0	-	Machine code stem used as is; no operand expression.
Type 1	-	Additional hex byte assembled from 8-bits of operand expression.
Type 2	-	Two additional hex bytes assembled from operand expression.
Type 3	-	Machine code stem merged with lowest 4-bit nybble of operand expression.
Type 4-9	-	Exception or Directive

Note - The op-code "DC" is treated as a special case. The program translates "DC" to a comma (,) and then evaluates the source line as a data statement.

PART II: THE COSMATIC DEVELOPER

An 1802 Development System

TABLE OF CONTENTS

INTRODUCTION

OVERVIEW

The COSMATIC DEVELOPER is a pair of programs which turn your ATARI Home Computer into an 1802 Development System. The system lets you do the following with your ATARI Home Computer:

Inspect 1802 Memory
Change 1802 Memory
Start 1802 Execution at any Address
Transfer ATARI Diskette Files to the 1802
Transfer 1802 Memory to the ATARI Diskette
Print 1802 Memory Contents
Have an Intelligent Terminal for the 1802

The 1802 microprocessor is used in many popular hobby computers and games because of its low cost and ease of implementation. Example systems include:

RCA COSMAC Development System
VIP Game Computer by RCA
Studio II Programmable Game by RCA
Hughes Development System
ELF II Hobby Computer by Netronics
The Anything Board by Netronics
Super ELF by Quest Electronics
CyberVision Home Computer
Popular Electronics' COSMAC ELF Series

The COSMATIC DEVELOPER will let you extend the capabilities of your ATARI Home Computer to the low cost 1802 system.

SYSTEM REQUIREMENTS

The COSMATIC DEVELOPER requires:

ATARI BASIC Language Cartridge
16K of memory
ATARI 850 Interface Module
ATARI 810 Disk Drive
Printer (optional)

Your 1802 should have an RS-232 electrical interface and the UT4 program in ROM or RAM. An RS-232 interface circuit description and UT4 ROM replacement program are included with the COSMATIC DEVELOPER.

The designation "UT4" refers to a part number in RCA development systems. This part is a 512-byte ROM containing the software for terminal communication with the development system. This is the communication "standard" used in the COSMATIC DEVELOPER. If you already have an RCA development sytem with a UT4 or equivalent ROM, then the COSMATIC DEVELOPER is fully compatible. If not, a new but compatible UT4 program is included with the COSMATIC DEVELOPER for your use.

DISK CONTENTS

The following files are included on your COSMATIC DEVELOPER diskette:

DOS.SYS DUP.SYS	Disk operating system and utility package. The DOS has been modified to work with only a single drive to reduce memory requirements. Refer to Appendix G in your DOS II reference manual to see how it was done. You must use this modified DOS if you have a 16K system. Larger systems can use an unmodified DOS.
AUTORUN.SYS	This the the standard AUTORUN.SYS File provided with DOS II and required for use with the disk and 850 interface module.
UT4E.OBJ	A text file containing 1802 hex machine code. This file was cross-assembled from UT4E.SRC and may be read by the COSMATIC DEVELOPER. This format is in the form created by the COSMATIC ATARI.
UT4E.DMP	A text file containing the same 1802 hex machine code that is in UT4E.OBJ. The format, however, is in the form created by the COSMATIC DEVELOPER when it transfers data from the 1802 to ATARI disk. UT4E.OBJ and UT4E.DMP are functionally identical and both conform to the UT4 standard.
UT4E.SRC	A text file containing the 1802 assembly language source which generated UT4E.OBJ. This source file may be copied to the screen or printer, entered into an assembler-editor cartridge for inspection, or cross-assembled by the COSMATIC ATARI.
DEV	The COSMATIC DEVELOPER in BASIC "LOAD" format.

USING THE COSMATIC DEVELOPER

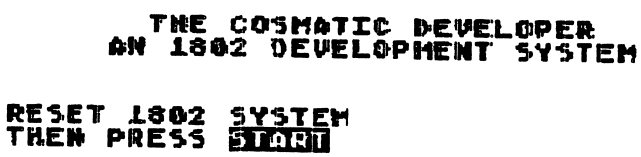
STARTUP

The following steps should be used in sequence:

1. Turn on the disk drive.
2. Turn on the interface module.
3. Insert the COSMATIC DEVELOPER diskette.
4. Insert the BASIC cartridge in the cartridge slot of your computer.
5. Turn on your computer.
6. When you get the "READY" prompt, enter:

```
RUN "D:DEV"
```

The following screen should appear:



Before you press START be sure you have done the following:

1. Connect the 1802 RS-232 interface to port #1 of the ATARI Interface Module.

2. Startup the UT4 program in the 1802.

Pressing START will cause the ATARI Home Computer to send a synchronizing character to the 1802. This character is used by the UT4 program in the 1802 to determine data rate (600 baud) and duplex (full). An acknowledgement is sent back to the ATARI Home Computer. When this occurs, the following screen will appear:

```
        THE COSMATIC DEVELOPER
    AN 1802 DEVELOPMENT SYSTEM

RESET 1802 SYSTEM
THEN PRESS START

SYNC CHAR SENT AND ACKNOWLEDGED

SELECT MODE:

   1. KEYBOARD
   2. 1802 --> FILE
   3. FILE --> 1802
   4. RESTART

?
```

At this point you may select one of three operating modes or restart. The restart will prompt you again for resetting the 1802 system and pressing START. Restart is helpful if a problem occurs either with the ATARI Home Computer or the 1802.

If you do not get the acknowledgement and "SELECT MODE" menu, then the 1802 system is not running or not connected. The COSMATIC DEVELOPER will not work without an 1802 executing a UT4 program.

KEYBOARD MODE

Entering "1" to the "SELECT MODE" choice will place you in keyboard mode. Here your ATARI Home Computer acts as a keyboard and display terminal to the 1802 system. The following screen will appear:

```
THE COSMATIC DEVELOPER
AN 1802 DEVELOPMENT SYSTEM

KEYBOARD MODE

>?
```

If you press RETURN at this point without any other entries, you will be returned to the "SELECT MODE" prompt. No data will be sent to the 1802.

Any other keys will be sent directly to the 1802. In fact, characters entered after the question mark are displayed as they are echoed back from the 1802. This is the full-duplex nature of the system's operation.

You may inspect, change, and execute the 1802 memory from your ATARI Home Computer as follows:

?Maaaa bb	To inspect bb bytes starting at location aaaa
!Maaaa b...b	To enter b...b bytes starting at location aaaa
$Paaaa	To start 1802 execution at location aaaa (PC=R0)

All numeric entries must be in hexidecimal. The COSMATIC DEVELOPER will right-justify and zero-fill hex addresses less than four digits in length.

The following is an example of the inspect command and its output:

```
      THE COSMATIC DEVELOPER
    AN 1802 DEVELOPMENT SYSTEM

KEYBOARD MODE

>??M0000 20
0000 0000 0000 0000 0000
     0000 0000 0000 0000;
0010 0000 0000 0000 0000
     0000 0000 0000 0000
>?
```

Note the two question marks. The first is displayed by the program, the second was entered from the keyboard. Both are required.

The following is an example of the change command:

```
      THE COSMATIC DEVELOPER
    AN 1802 DEVELOPMENT SYSTEM

KEYBOARD MODE

>??M0000 20
0000 0000 0000 0000 0000
     0000 0000 0000 0000;
0010 0000 0000 0000 0000
     0000 0000 0000 0000
>?!M0000 000102030405060?
>?
```

Spaces can be inserted in the data as entered if desired.

The following is an example of the change command for longer strings of hex data:

```
          THE COSMATIC DEVELOPER
        AN 1802 DEVELOPMENT SYSTEM

KEYBOARD MODE

>??M00 10
0000 0F0E 0D0C 0B0A 0908
     0706 0504 0302 0100
>?!M00 000102030405060 7,
>?08090A0B0C0D0E0F

[redacted]>??M00 10
0000 0001 0203 0405 0607
     0809 0A0B 0C0D 0E0F
>?
```

Note the comma at the end of the data to be entered. This signals the UT4 software in the 1802 system that the next line is to follow in memory. The new address need not be entered (nor the "!M"). The continuation can be marked with the concluding comma for as many lines as necessary.

The following is an example of the start execution command:

```
          THE COSMATIC DEVELOPER
        AN 1802 DEVELOPMENT SYSTEM

KEYBOARD MODE

>?$P000
[redacted]>?
```

Starting execution in the 1802 system really means transferring control from the UT4 program in the 1802 to another program in the 1802. This new program can also use subroutines in the UT4 program to display text on your ATARI Home Computer and accept text from your keyboard. Instructions in the UT4E.SRC file describe how to use the UT4 subroutines available to you. Knowledge of 1802 assembly language is assumed.

1802-to-file

Entering "2" to the "SELECT MODE" choices will place you in 1802-to-file mode. This mode transfers 1802 memory contents to a file of your choice. The following is an example:

```
      THE COSMATIC DEVELOPER
    AN 1802 DEVELOPMENT SYSTEM

1802 --> FILE

FD (*=DIRECTORY) ?D:TEST.DMP
MAKE ENTRIES IN HEX

STARTING ADDR.  ?0000
NUMBER OF BYTES ?20

0000 F800 B0B5 F808 A5D5
0008 F801 B3F8 00A3 D3F8
0010 8EA3 D30D D30A D32A
0018 F800 ADBD F83E A3D3

STARTING ADDR.  ?
```

First, the file description ("FD") is entered. This can be a disk file, the screen ("S:"), or the printer ("P:").

Next, enter the 1802 starting address in hex. A null entry ("RETURN" only) will return you to the "SELECT MODE" menu.

Finally, enter the number of bytes desired in hex ("FFFF" maximum). Entering zero will return you to the "SELECT MODE" menu.

The program will then transfer the 1802 memory contents to the file you designate. You will then be prompted for a new starting address. A null entry ("RETURN" only) closes the file and returns you to the "SELECT MODE" menu. The following is an example of a continued transfer:

```
      THE COSMATIC DEVELOPER
    AM 1802 DEVELOPMENT SYSTEM

1802 --> FILE

FD (*=DIRECTORY) ?D:TEST.DMP
MAKE ENTRIES IN HEX

STARTING ADDR.  ?0000
NUMBER OF BYTES ?20

0000 F800 B0B5 F808 A5D5
0008 F801 B3F8 00A3 D3F8
0010 8EA3 D30D D30A D32A
0018 F800 ADBD F83E A3D3

STARTING ADDR.  ?0100
NUMBER OF BYTES ?18

0100 F800 AEAF F800 BCF8
0108 C4AC 370A 3F0C F803
0110 FF01 3A10 8F3A 1A37

STARTING ADDR.  ?
```

This process can be repeated until you have transferred all the memory contents you wish and in the sequence you desire. This is an excellent way to build an 1802 load file.

The file format always starts with a four-hex-character address followed by eight hex-characters of 1802 memory contents.

File-to-1802

Entering "3" to the "SELECT MODE" choice will place you in file-to-1802 mode. This mode transfers the contents of an ATARI Home Computer file to 1802 memory. The following is an example using the UT4E.OBJ file:

```
          THE COSMATIC DEVELOPER
        AN 1802 DEVELOPMENT SYSTEM

FILE --> 1802

FD (*=DIRECTORY) ?D:UT4E.OBJ
!M0000 F800;
!M0002 B0;
!M0003 B5;
!M0004 F808;
!M0006 A5;
!M0007 D5;
!M0008 F801;
!M000A B3;
!M000B F800;
!M000D A3;
!M000E D3;
!M000F F88E;
!M0011 A3;
!M0012 D30D;
!M0014 D30A;
```

The following is an example using the UT4E.DMP file:

```
          THE COSMATIC DEVELOPER
        AN 1802 DEVELOPMENT SYSTEM

FILE --> 1802

FD (*=DIRECTORY) ?D:UT4E.DMP
!M0000 F800 B0B5 F808 A5D5
!M0008 F801 B3F8 00A3 D3F8
!M0010 8EA3 D30D D30A D32A
!M0018 F800 ADBD F83E A3D3
!M0020 FB24 32AB FB05 A1CE
!M0028 FB1E 3A18 D3FB 4D3A
!M0030 A0D3 3B31 D333 34FB
!M0038 203A A09D B08D A081
!M0040 328A F800 ADBD D333
!M0048 46FB 0D3A A0F3 8EA3
!M0050 8DA1 9DB1 D30A 90BF
!M0058 F8A0 A3D3 80BF F8A0
!M0060 A3D3 D320 408F F8A0
!M0068 A3D3 2181 3A71 9132
!M0070 0F80 FA0F 3A7C D33B
```

The program prefixes the file with the "!M" characters and transfers it to the 1802 until an end of file is read.

All you have to enter is the file description ("FD"). The memory addresses and data are stored in the file.

Both file transfer modes permit you to get a disk directory. Entering an asterisk ("*") for the file description will give you a disk directory. The following is an example:

```
        THE COSMATIC DEVELOPER
     AM 1802 DEVELOPMENT SYSTEM

FILE --> 1802

FD (*=DIRECTORY) ?*

* DOS       SYS 039
* DUP       SYS 042
* AUTORUN   SYS 001
  UT4E      OBJ 024
  UT4E      DMP 012
  UT4E      SRC 090
  DEV           047
452 FREE SECTORS

FD (*=DIRECTORY) ?
```

To escape from this mode without a file transfer, press the "RETURN" key without any other entry.

THE 1802 INTERFACE

UT4 Program

The UT4 program runs in the 1802 to provide communication with the COSMATIC DEVELOPER. It takes less than 512 bytes and will also support RS-232 communication to the ATARI Home Computer by an 1802-host program.

RCA development systems include a UT4 ROM for terminal communication. If you do not have this ROM, use the UT4E files included with the COSMATIC DEVELOPER to burn your own EPROM.

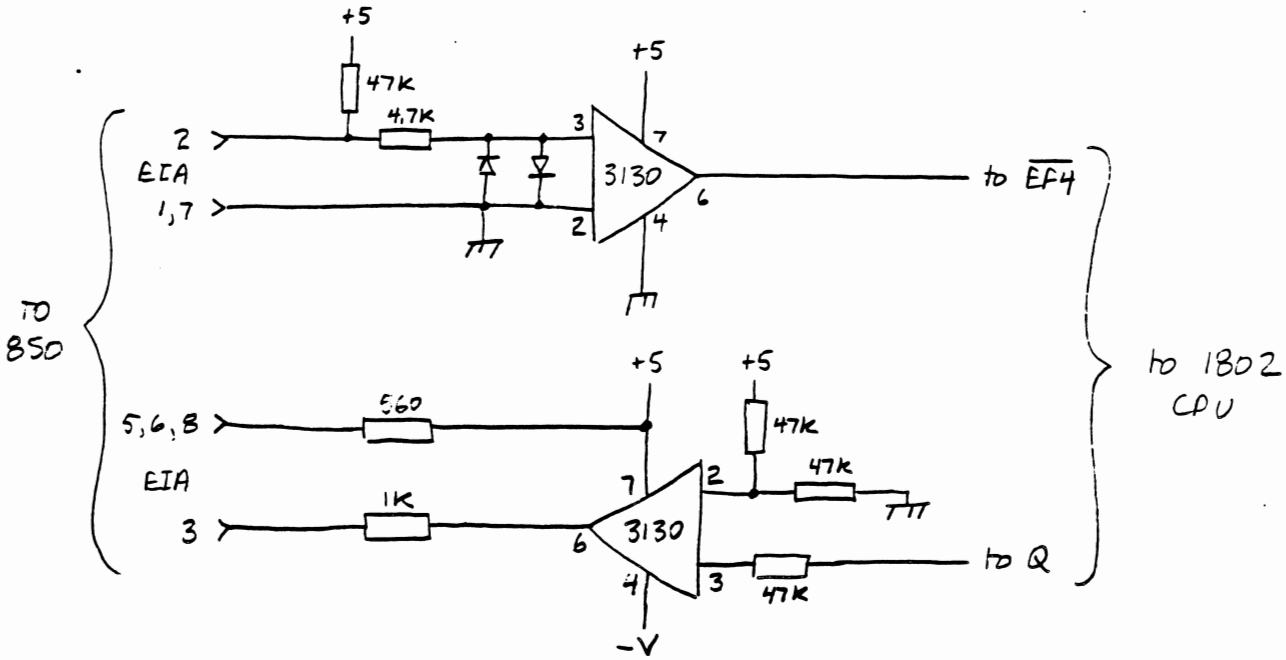
Relocating UT4E is easy. The files provided have an origin of 0 (zero). You can either (1) reassemble UT4E.SRC with a new origin or (2) change two bytes in UT4E.OBJ or UT4E.DMP. The byte changes are:

	contents	
address	current	new
0001	00	origin page
0009	01	origin page + 1

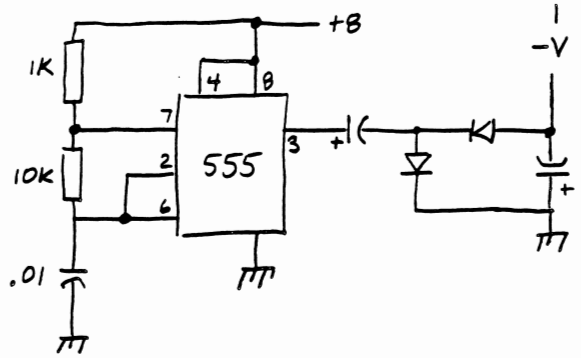
You are encouraged to inspect the UT4E.SRC assembly language source file if you want to see how the program works.

RS-232 INTERFACE CIRCUIT

A number of circuits have been published to permit RS-232 communications with the 1802. The UT4 protocol uses the "Q" line for output from the 1802 and the "EF4" line for input to the 1802. Serial to parallel conversion is all done in software by the UT4; no UART is required. One possible interface circuit is as follows:



RS-232 INTERFACE



VOLTAGE DOUBLER

The circuit was designed by Jim McConnell (Chief Engineer, Astrovision, Inc.).

RCA has a number of excellent publications describing UT4 operation and RS-232 interfaces. One such reference is "Evaluation Kit Manual for the RCA CDP1802 COSMAC Microprocessor", part number MPM-203 from RCA.

LIMITED WARRANTY ON MEDIA AND HARDWARE ACCESSORIES.

We, Atari, Inc., guarantee to you, the original retail purchaser, that the medium on which the APX program is recorded and any hardware accessories sold by APX are free from defects for thirty days from the date of purchase. Any applicable implied warranties, including warranties of merchantability and fitness for a particular purpose, are also limited to thirty days from the date of purchase. Some states don't allow limitations on a warranty's period, so this limitation might not apply to you. If you discover such a defect within the thirty-day period, call APX for a Return Authorization Number, and then return the product along with proof of purchase date to APX. We will repair or replace the product at our option.

You void this warranty if the APX product: (1) has been misused or shows signs of excessive wear; (2) has been damaged by use with non-ATARI products; or (3) has been serviced or modified by anyone other than an Authorized ATARI Service Center. Incidental and consequential damages are not covered by this warranty or by any implied warranty. Some states don't allow exclusion of incidental or consequential damages, so this exclusion might not apply to you.

DISCLAIMER OF WARRANTY AND LIABILITY ON COMPUTER PROGRAMS.

Most APX programs have been written by people not employed by Atari, Inc. The programs we select for APX offer something of value that we want to make available to ATARI Home Computer owners. To offer these programs to the widest number of people economically, we don't put APX products through rigorous testing. Therefore, APX products are sold "as is", and we do not guarantee them in any way. In particular, we make no warranty, express or implied, including warranties of merchantability and fitness for a particular purpose. We are not liable for any losses or damages of any kind that result from use of an APX product.

ATARI PROGRAM EXCHANGE

REVIEW FORM

We're interested in your experiences with APX programs and documentation, both favorable and unfavorable. Many software authors are willing and eager to improve their programs if they know what users want. And, of course, we want to know about any bugs that slipped by us, so that the software author can fix them. We also want to know whether our documentation is meeting your needs. You are our best source for suggesting improvements! Please help us by taking a moment to fill in this review sheet. Fold the sheet in thirds and seal it so that the address on the bottom of the back becomes the envelope front. Thank you for helping us!

1. Name and APX number of program _____

2. If you have problems using the program, please describe them here.

_____

_____

_____

3. What do you especially like about this program?

_____

_____

_____

4. What do you think the program's weaknesses are?

_____

_____

_____

5. How can the catalog description be more accurate and/or comprehensive?

_____

_____

6. On a scale of 1 to 10, 1 being "poor" and 10 being "excellent", please rate the following aspects of this program?

_____ Easy to use
_____ User-oriented (e.g., menus, prompts, clear language)
_____ Enjoyable
_____ Self-instructive
_____ Useful (non-game software)
_____ Imaginative graphics and sound

7. Describe any technical errors you found in the user instructions (please give page numbers).

8. What did you especially like about the user instructions?

9. What revisions or additions would improve these instructions?

10. On a scale of 1 to 10, 1 representing "poor" and 10 representing "excellent", how would you rate the user instructions and why?

11. Other comments about the software or user instructions:

STAMP

ATARI Program Exchange
Attn: Publications Dept.
P.O. Box 50047
60 E. Plumeria Drive
San Jose, CA 95150

[seal here]