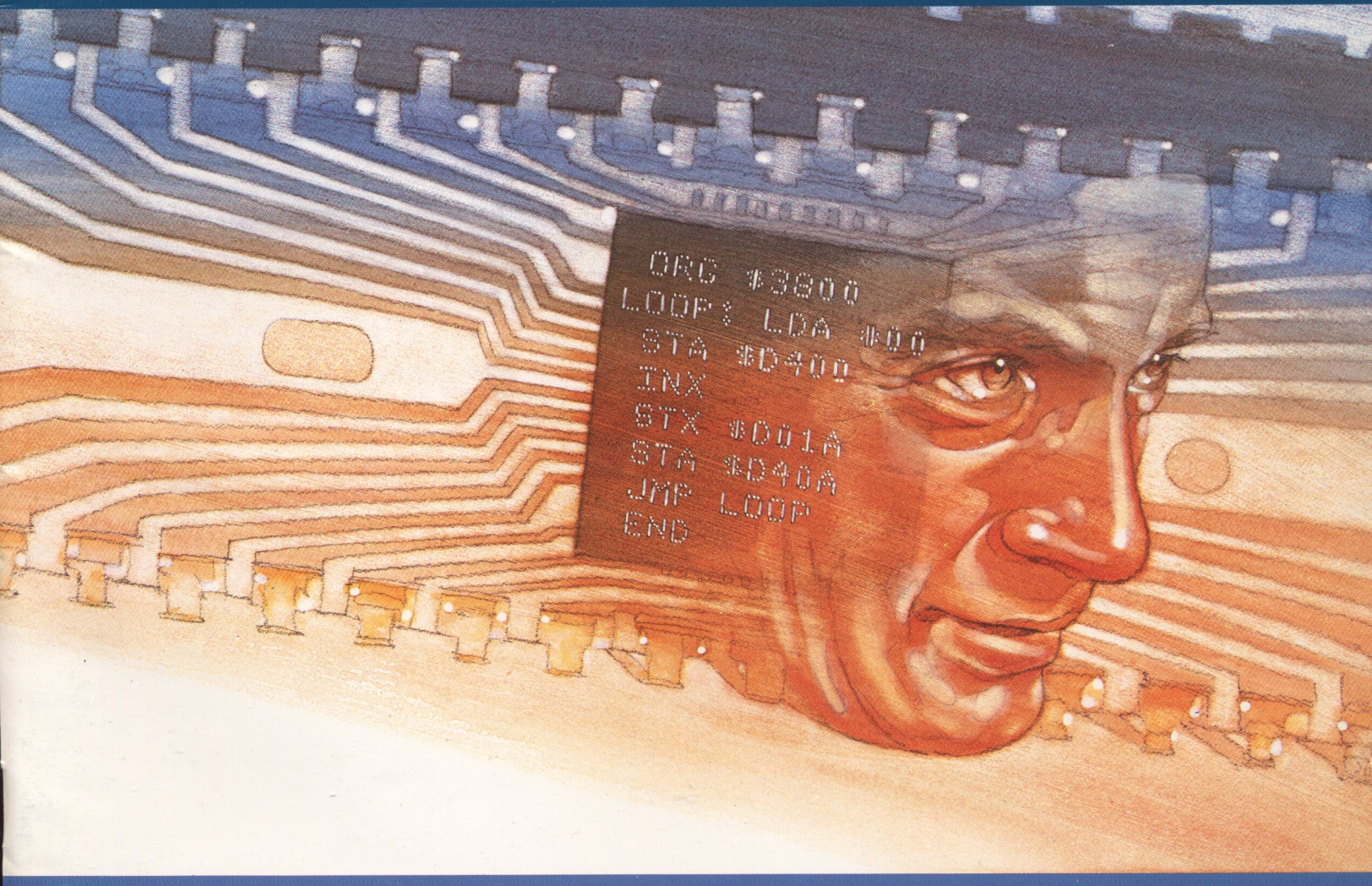
ATARI® 800™

ATARI® MACRO ASSEMBLER



ATARI®
A Warner Communications Company

Use with ATARI 800™
PERSONAL COMPUTER SYSTEM

ATARI®
MACRO ASSEMBLER

A Warner Communications Company

PRINTED IN U.S.A.

CONTENTS

1 INTRODUCTION 1

Features of This Package 1
Macros 1
Conditional Assembly and Code Duplication 1
Systext Files 1
Program Listing Control 1
Cross-Reference Tables 1
Standard ATARI Computer and 6502 Mnemonics 2

Contents of This Software Package 2
Procedures 2
Program Loading Instructions 2
Creating a Source Program 2
Assembling a Source Program 3

Purpose of This Manual 3
References 3

2 ASSEMBLER EXECUTION 5

Command Line Syntax 5
Command Line Options 5
Command Line Examples 6
User Interface 7

3 FILE USAGE 9

Source Input Files 9
System Text Files 9
Object Output File 9
Listing File 9
Source Listing Format 10
Sample Listing 10
Symbol Map Format 14

4 LANGUAGE STRUCTURE 15

Statements 15
Label Field 15
Operation Field 15
Variable Field 15

Statement Termination 16
Comments 16
Definitions 16
Symbols and Names 16
Numbers 17
Character Strings 17
Expressions 18
Operands 18

5 MACRO FACILITY 21

Macro Definition 21
Macro Call 21
Code Duplication 22
Nesting 22

6 PSEUDO-OPERATIONS 23

ASSERT 23
DB 24
DC 24
DS 24
DW 25
ECHO . . . ENDM 25
EJECT 26
END 26
EQU or = 27
ERR 27
IF . . . ENDIF, IF . . . ELSE . . . ENDIF 27
INCLUDE 28
LINK 29
LIST 31
LOC 32
MACRO . . . ENDM 33
ORG 35
PROC . . . EPROC 35
REAL6 36
SET 36
SPACE 36
SUBTTL 37
TITLE 37
USE 38
VFD 38

7	PSEUDO-OP QUICK REFERENCE	41
8	INSTRUCTION MNEMONICS	43
9	USING THE ATARI MACRO ASSEMBLER WITH THE ATARI ASSEMBLER EDITOR SOURCE FILES	49
10	ERROR CODES	51

1

INTRODUCTION

The ATARI® Macro Assembler is a software development tool for writing 6502 assembly language programs for the **ATARI 800™ Home Computer**. The features of this assembler include macros, conditional assembly, code duplication, access to library definitions, program-listing control, and cross-reference tables. It offers fast compilation and uses standard 6502 mnemonics.

FEATURES OF THIS PACKAGE

MACROS

The macro feature allows you to define code words to represent multiple instructions. It makes it easy for you to use a sequence of code many times in a program.

CONDITIONAL ASSEMBLY AND CODE DUPLICATION

Conditional assembly allows the generation of source code based on certain conditions. Combined with macros this offers a powerful and versatile way of coding assembly language programs. An ECHO pseudo-operation enables you to repeat sections of code (similar to the macro feature, but it does not allow parameter passing).

SYSTEXT FILES

Often you will want to create and store symbols and macro definitions on a library file. Once created, the symbols can be referenced by any of your source programs. Such a library file can ease your program development effort.

PROGRAM LISTING CONTROL

The LIST pseudo-op lets you tailor and annotate programs to fit your exact needs. The pseudo-op makes documentation easier by allowing listing control and page headings.

CROSS-REFERENCE TABLES

The Macro Assembler also includes an optional cross-reference table so that you can reference labels and variables in the source program quickly.

STANDARD ATARI COMPUTER AND 6502 MNEMONICS

A file containing the ATARI Home Computer Hardware Register addresses and OS Shadow Register addresses is included on your Macro Assembler diskette. You may reference standard ATARI Computer mnemonics in your programs using this file. See Systext reference in "Command Line Options" in Section 2.

Standard MOS Technology 6502 microprocessor coding format is used in this assembler. The formation of expressions also follows the standard conventions.

CONTENTS OF THIS SOFTWARE PACKAGE

The Macro Assembler includes:

- A diskette containing both the Macro Assembler and Program-Text Editor™ software
- A reference card giving pseudo-ops, error codes, and Program-Text Editor commands and messages
- This reference manual for the ATARI Macro Assembler
- An operators manual for the ATARI Program-Text Editor

PROCEDURES

PROGRAM LOADING INSTRUCTIONS

1. Connect the ATARI 800 Home Computer to a television set and to a wall outlet as instructed in the operators manual.

2. Connect the **ATARI 810™ Disk Drive** to the computer console and to a wall outlet as instructed in the *ATARI 810 Disk Drive Operators Manual*. Verify that the disk drive is set to DRIVE CODE 1 as instructed in the operators manual.

3. Open the cartridge door on the top of the computer console. Remove all cartridges from the top front cartridge slots. Close the cartridge door.

4. Turn on your television set.

5. Turn the disk drive POWER (PWR) switch to ON. Two red lights (the BUSY light and the PWR ON light) will come on.

6. When the BUSY light goes out, open the disk drive door by pressing the door handle release lever.

7. Insert the diskette containing the Macro Assembler and Program-Text Editor programs into disk drive 1.

8. Switch the POWER (PWR) switch on the computer console to ON.

The DOS II Menu will now appear on your screen.

CREATING A SOURCE PROGRAM

To use the editor, refer to the *ATARI Program-Text Editor Manual*.

After you create your source program, exit the Program-Text Editor using the commands that will return you to DOS:

1. Press OPTION.
2. Type **EXIT** and press START. (This returns you to DOS.)

Then, to assemble your source program:

1. Type the letter **L** and press RETURN.
2. Type **AMAC** and press RETURN.

ASSEMBLING A SOURCE PROGRAM

1. Refer to "Command Line Syntax" (in Section 2) for the command line syntax and command line options. Press RETURN after the command line.
2. After the assembly, press the RETURN key to return to DOS. Your DOS directory will now show that you have created an object file with an extension, OBJ.

PURPOSE OF THIS MANUAL

This manual is intended to show you how to use the Macro Assembler. If you plan to use the Program-Text Editor for creating your source program, it is suggested that you read the *ATARI Program-Text Editor Manual*, then practice creating files.

A knowledge of assembly language and ATARI DOS II is also necessary. The texts listed below will assist in your study of assembly language. If you wish to become familiar with the special features of the ATARI Home Computer, a copy of the *ATARI Technical Users Notes* will be needed.

REFERENCES

We recommend the following books:

MOS Programming Manual by MOS Microcomputers
SY6500/MCS6500 Microcomputer Family Programming Manual by SYNERTEK
6502 Assembly Language Programming by Lance Leventhal
6502 Software Design by Leo Scanlon
6502 Software Gourmet Guide and Cookbook by Robert Findley

ATARI publications:

ATARI DOS II Reference Manual
ATARI Technical Users Notes

2

ASSEMBLER EXECUTION

COMMAND LINE SYNTAX

The Macro Assembler is accessed by the ATARI DOS II Menu option L. When DOS asks for a filename to load, type:

AMAC RETURN

Once AMAC is loaded into memory, it will ask you to "Enter source filename and options." The source filename must always be specified. Any options you wish to use should follow the filename, separated by either a comma or space. The command line is terminated by a carriage return. The command line cannot be edited using the cursor control keys.

The general form of the command line is: <filespec> opt1,...optn. Where <filespec> is the source file to be assembled and is of the form <device>:<filename>.<extension>. The above command line could have been typed with any mixture of upper- or lowercase characters. The assembler will convert all command line characters to uppercase before interpretation.

COMMAND LINE OPTIONS

The 'opt1,...optn' are optional parameters (in any order) chosen from this list:

H=Dn: (Default is H=Dn: where n is the same disk drive as the source file)	Generate object output file to the specified disk drive where n may be 1, 2, 3, or 4. If no filename is specified, the object file will be named with the input source filename and the extension, OBJ.
H=<filespec>	Write object code to <filespec>.
H=0	Do **not** generate any object code.
L=P:	List output to printer.
L=Dn:	List output to specified disk drive (n= 1, 2, 3, or 4). List filename has the input source filename and the extension PRN.
L=S:	Output listing to the screen.
L=0 (Default)	Do **not** produce listing for this assembly.
O=n	Preset the value of the run address of the object program. Specifying "O=n" on the command line is exactly like the statement "END n" found at the end of an assembly program.
O=0 (Default)	Set the value of the run address to zero.

PS=n (Default is PS=63)	Set page size to <n> source lines per page. Page size must be less than 127. When page size is less than 10, no title or subtitle lines nor page ejects are printed in the list file, and a full cross-reference is disallowed.
PS=0	Do **not** print title and subtitle lines and page ejects to list file for this assembly.
S=<filespec>	Specify systext file. The S option may be repeated. The user may specify as many systexts as desired, so long as combined number of systexts and link files does not exceed the file limit of 40.
S	Use the default systext D:SYSTEXT.AST.
S=0 (Default)	Specify no systext for this assembly.
R=F	Generate full reference map. List all global symbols and their references on the file specified by the L parameter.
R=S	Generate short reference map. List all global symbols and their values only on the file specified by the L parameter.
R=0 (Default)	Do **not** generate reference map.
SL=n (Default is SL=80 for P: and SL=38 for S:)	Set the line length. Maximum length of the line output to the list file will be <n> characters; the rest of the line is discarded if <n> is greater than the device line length.

All numeric argument values (for O=n, PS=n, and SL=n) may be specified according to the general syntax for numbers. In particular, an explicit radix (decimal, binary, octal, or hexadecimal) can be used. Refer to Section 4, "Numbers," for radix specification.

All lowercase letters on the command line are converted to uppercase before interpretation.

COMMAND LINE EXAMPLES

D:TESTIT.ASM

will read input file D1:TESTIT.ASM (D: implies D1:), no listing will be produced, and the ATARI binary format object file will be D1:TESTIT.OBJ.

D:TESTIT.ASM H=0 R=F L=S:

will assemble D1:TESTIT.ASM, suppress object file generation, and send a listing with full reference map to the screen.

D2:TESTIT.ASM H=D: L=D2: R=F O=$200

The assembler will assemble the file D2:TESTIT.ASM generating the object file D1:TESTIT.OBJ, and will produce a listing and full reference map in D2:TESTIT.PRN. In addition, it will also set the run address to $200.

D2:TESTIT.ASM S S=D2:MSYS.AST L=P: R=F H=D: O=$1700

The assembler will process the two systext files D1:SYSTEXT.AST and D2:MSYS.AST, assemble the file D2:TESTIT.ASM, produce the object file D1:TESTIT.OBJ with a run address of $1700, and print a listing with full reference map on the printer.

USER INTERFACE

The assembler execution may be prematurely terminated by pressing the BREAK key. When output listing is directed to the screen, its execution can be temporarily halted by simultaneously pressing the CTRL key and the **1** key. Pressing those two keys again will restart execution.

If a disk-write error happens (usually disk or directory full), the offending file (object or list file) is erased, an error message is issued to the screen, and further attempts to write to the file are suppressed. Assembly then continues normally.

Assembly time errors are printed to the screen as well as to the list file.

3

FILE USAGE

SOURCE INPUT FILES

You can specify source input files by using the:

- First command line argument
- Systext file argument (S parameter)
- LINK pseudo-instruction
- INCLUDE pseudo-instruction

All input files must be in Program-Text Editor format. They consist of a line or lines of ATASCII characters terminated by ATASCII End-of-Lines <EOL>.

SYSTEM TEXT FILES

A system text file (systext) is an assembly language file of symbols and macro definitions. The programmer can predefine symbols here for many different programs. Some examples are:

- ATASCII control characters (BS, TAB, ESC, EOL,...)
- Addresses (entry points into CIO, SIO, and channel locations)
- Macros

If an assembly error is encountered while scanning a systext file, the assembler aborts with an error message.

OBJECT OUTPUT FILE

The object output file generated by the assembler has a default file extension of OBJ and is in ATARI binary format. Refer to the *ATARI DOS II Reference Manual* for detail specifications of binary format.

LISTING FILE

The output listing of the source program generated by the assembler has a default extension of PRN.

The Macro Assembler has a flexible set of listing control pseudo-ops which allows the user to generate only the desired program content.

Page heading (unless suppressed via PS=0) contains the assembler version and page number as well as optional user-specified title information (see TITLE and SUBTTL pseudo-ops).

The LIST pseudo-op (or L command line argument) controls which source lines are listed. Each code line listed begins with 20 columns of information generated by the assembler.

Column 1 of the listing output is reserved exclusively for errors; a listing free of assembly errors will not have any printing in column 1. An error count is reported at the end of the assembly. (See Section 10, " Error Codes.")

SOURCE LISTING FORMAT

```
          1         2
123456789.123456789.
E addr#  hhhhhhhhhh      Line that generates code.
R addr   = vvvv          EQU, SET, IF, etc.
R -                      Line that is skipped.
O addr   = vvvv #        Location and origin counters are unequal.
R addr
          +hhhhhhhhhh    Macro-generated line.
  addr   hhhh Aaddr      Destination address of PC relative jumps.
```

Column	Description
1	Error flag or blank. See Section 10 for the meaning of error flags.
2	Blank.
3-6	Address location of this instruction (value of the location counter).
6	— sign means line not assembled due to IF...ELSE. Line only listed if LIST F in effect.
7	# sign means the location and origin counters are unequal.
8	+ sign means assembler-generated line. Line listed if LIST M in effect.
9-18	hhhhhhhhhh is the resultant code. Up to five bytes are listed. If LIST G or D is in effect, multiple lines will be listed with up to five bytes on each.
11-14	vvvv = value of expression.
19-20	Always blank.
21-80	Source statement.

SAMPLE LISTING

```
                    I/O EQUATES
=009B    EOL   =    $9B
=0030    IOCB3 =    $30
=0340    ICHID =    $0340
=0341    ICDNO =    ICHID+1
=0342    ICCOM =    ICDNO+1
=0343    ICSTA =    ICCOM+1
=0344    ICBAL =    ICSTA+1
=0345    ICBAH =    ICBAL+1
=0346    ICPTL =    ICBAH+1
=0347    ICPTH =    ICPTL+1
=0348    ICBLL =    ICPTH+1
=0349    ICBLH =    ICBLL+1
=034A    ICAX1 =    ICBLH+1
=034B    ICAX2 =    ICAX1+1
```

```
                =0003           OPEN =      $03
                =0005           GETREC =    $05
                =0009           PUTREC =    $09
                =000C           CLOSE =     $0C
                =0004           OREAD =     $04
                =0008           OWRIT =     $08
                =0088           EOF =       $88
                =E456           CIOV =      $E456
                =0040           IOCB4 =     $40
                                ;
                                ;FIRST INIT THE IOCB FOR OPEN
                                ;
0000#           =5000                       ORG $5000

                                ;DATA REGION
5000            44323A5445      ;NAME1      DB          'D2:TEST1',EOL
                =0050           BUF1SZ      =           80
                =5009           BUF1        =           *
5009            =5059                       ORG         *+BUF1SZ
5059            50323A9B        NAME2       DB          'P2:',EOL
505D            A230            START       LDX         #IOCB3
505F            A900                        LDA         #LOW NAME1
5061            9D4403                      STA         ICBAL,X
5064            A950                        LDA         #HIGH NAME1
5066            9D4503                      STA         ICBAH,X
5069            A900                        LDA         #0
506B            9D4B03                      STA         ICAX2,X
                                ;
                                ;"OPEN" THE DISK
                                ;
506E            A903                        LDA         #OPEN
5070            9D4203                      STA         ICCOM,X
5073            2056E4                      JSR         CIOV
5076            BC4303                      LDY         ICSTA,X
5079            1003 ^507E                  BPL         L1
507B            4CA250                      JMP         ERR2
                                ;
                                ;CHANNEL 4 IS PRINTER
                                ;
507E            A240            L1          LDX         #IOCB4
5080            A959                        LDA         #LOW NAME2
5082            9D4403                      STA         ICBAL,X
5085            A950                        LDA         #HIGH NAME2
5087            9D4503                      STA         ICBAH,X
508A            A908                        LDA         #OWRIT
508C            9D4A03                      STA         ICAX1,X
508F            A900                        LDA         #O
5091            9D4B03                      STA         ICAX2,X
                                ;
                                ;"OPEN" THE PRINTER
                                ;
5094            A903                        LDA         #OPEN
5096            9D4203                      STA         ICCOM,X
5099            2056E4                      JSR         CIOV
509C            BC4303                      LDY         ICSTA,X
509F            1004 ^50A5                  BPL         TP10
```

```
                        ;
                        ;ERROR — JUST BRK
                        ;
50A1    00      ERR1    BRK
50A2    00      ERR2    BRK
50A3    00      ERR3    BRK
50A4    00      ERR4    BRK
                        ;
                        ;SETUP TO READ A RECORD
                        ;
50A5    A230    TP10    LDX     #IOCB3
50A7    A905            LDA     #GETREC
50A9    9D4203          STA     ICCOM,X
50AC    A909            LDA     #LOW BUF1
50AE    9D4403          STA     ICBAL,X
50B1    A950            LDA     #HIGH BUF1
50B3    9D4503          STA     ICBAH,X
                        ;
                        ;READ RECORDS
                        ;
50B6    A950    LOOP    LDA     #LOW BUF1SZ
50B8    9D4803          STA     ICBLL,X
50BB    A900            LDA     #HIGH BUF1SZ
50BD    9D4903          STA     ICBLH,X
50C0    2056E4          JSR     CIOV
50C6    1004 ^50CC      BPL     PRNTR
                        ;
                        ;NEG STATUS ON READ — EOF
                        ;
50C8    C088    TP20    CPY     #EOF
50CA    D0D7 ^50A3      BNE     ERR3
                        ;
                        ;PRINT A RECORD
                        ;
50CC    BD4803  PRNTR   LDA     ICBLL,X
50CF    A240            LDX     #IOCB4
50D1    9D4803          STA     ICBLL,X
50D4    A230            LDX     #IOCB3
50D6    BD4903          STA     ICBLH,X
50D9    A240            LDX     #IOCB4
50DB    9D4903          STA     ICBLH,X
50DE    A909            LDA     #PUTREC
50E0    9D4203          STA     ICCOM,X
50E3    A909            LDA     #LOW BUF1
50E5    9D4403          STA     ICBAL,X
50E8    A950            LDA     #HIGH BUF1
50EA    9D4503          STA     ICBAH,X
50ED    2056E4          JSR     CIOV
50F0    BC4303          LDY     ICSTA,X
50F3    1003 ^50F8      BPL     L3
50F5    4CA450          JMP     ERR4
50F8    A230    L3      LDX     #IOCB3
50FA    BC4303          LDY     ICSTA,X
50FD    C088            CPY     #EOF
50FF    F003 ^5104      BEQ     L2
5101    4CA550          JMP     TP10
```

```
5104  A90C      L2    LDA    #CLOSE
5106  9D4203          STA    ICCOM,X
5109  2056E4          JSR    CIOV
510C  A90C            LDA    #CLOSE
510E  A230            LDX    #IOCB3
5110  9D4203          STA    ICCOM,X
5113  2056E4          JSR    CIOV
5116  00              BRK
5117                  END
```

No ERRORS, 39 labels, $A3E6h free.

```
 BUF1     5009  1#36  2/28  2/30  2/60  3/ 2
 BUF1SZ   0050  1#35  1/37  2/35  2/37
 CIOV     E456  1#25  1/51  2/12  2/39  3/ 4  3/18  3/23
 CLOSE    000C  1#21  3/16  3/20
 EOF      0088  1#24  2/45  3/12
 EOL      009B  1#3   1/34  1/38
nERR1     50A1  2#18
 ERR2     50A2  1/54  2#19
 ERR3     50A3  2#20  2/46
 ERR4     50A4  2#21  3/ 8
 GETREC   0005  1#19  2/26
 ICAX1    034A  1#15  1/16  2/ 4
 ICAX2    034B  1#16  1/45  2/ 6
 ICBAH    0345  1#10  1/11  1/43  2/ 2  2/31  3/ 3
 ICBAL    0344  1# 9  1/10  1/41  1/60  2/29  2/61
 ICBLH    0349  1#14  1/15  2/38  2/54  2/56
 ICBLL    0348  1#13  1/14  2/36  2/50  2/52
 ICCOM    0342  1# 7  1/ 8  1/50  2/11  2/27  2/59  3/17
                3/22
 ICDNO    0341  1# 6  1/ 7
 ICHID    0340  1# 5  1/ 6
 ICPTH    0347  1#12  1/13
 ICPTL    0346  1#11  1/12
 ICSTA    0343  1# 8  1/ 9  1/52  2/13  2/40  3/ 6  3/11
 IOCB3    0030  1# 4  1/39  2/25  2/53  3/10  3/21
 IOCB4    0040  1#26  1/58  2/51  2/55
 L1       507E  1/53  1#58
 L2       5104  3/13  3#16
 L3       50F8  3/ 7  3#10
nLOOP     50B6  2#35
 NAME1    5000  1#34  1/40
 NAME2    5059  1#38  1/59  1/61
 OPEN     0003  1#18  1/49  2/10
nOREAD    0004  1#22
 OWRIT    0008  1#23  2/ 3
 PRNTR    50CC  2/41  2#50
 PUTREC   0009  1#20  2/58
nSTART    505D  1#39
 TP10     50A5  2/14  2#25  3/14
nTP20     50C8  2#45
```

SYMBOL MAP FORMAT

When R=S is selected, the short symbol map is printed at the end of the program listing. For each symbol name in the program, the following is printed:

sa symbol hhhh, where:

<s> is blank or "s" for a name introduced in a systext file.
<a> is either blank or
U = undefined, or
D = doubly defined, or
n = not referenced.
<symbol> is the name of the symbol.

<hhhh> is the symbol value in hexadecimal, or "mac" if the name is a macro.
Four symbols are printed on each line, using the default line length.

When R=F is selected, the full cross-reference map follows the source listing. On each line, in addition to the R=S information above, cross-reference information is listed. Each reference has the form:

ppp/ll

where <ppp> equals page number and <ll> equals line number. For a definition reference, the / is replaced by #.

Names beginning with a : (local symbols) and a ? (usually macro invented) are not included in either type of symbol map output.

Symbols defined in a systext file appear in the cross-reference only if they are used during the assembly; they are flagged with an s.

4

LANGUAGE STRUCTURE

A Macro Assembler source program consists of a sequence of statements, comments, and definitions. Statements are the fundamental units of assembly. Comments do not affect assembler operation or object output. Definitions may be conditionally assembled, saved for later assembly, or repeated.

All characters in a statement are converted to uppercase except those in the comment field.

STATEMENTS

A statement is divided into three fields: a label field, an operation field, and a variable field.

LABEL FIELD

The label field begins with the first character of the statement and is terminated by a blank or an end of statement. If a colon (:) is the last character of the label field, it is discarded. For example:

```
SYMBX:    ADC     MEM,X     ;comment
```

SYMBX is the defined label.

OPERATION FIELD

The operation field begins with the first nonblank character after the label field and terminates with the next blank character. Machine op codes, pseudo-ops, and macro calls all occur in the operation field. If this field is empty, the variable field must be empty also. For example:

```
SYMBX:    ADC     MEM,X     ;comment
```

ADC is the machine op code.

VARIABLE FIELD

The variable field begins with the first character after the operation field and is terminated by an end of statement. Variables, expressions, and other arguments used by the operation field appear in this field. For example:

```
SYMBX:    ADC     MEM,X     ;comment
```

MEM,X is the variable.

STATEMENT TERMINATION

A statement is terminated by:

Beginning of comment (;), or
End-of-Line, or
Logical end of statement mark (!).

```
SYMBX:   ADC   MEM,X     ;comment
SYMBY:   ADC   MEM,X
SYMBZ:   ASL   ! ASL     ! ASL       ! ASL     ; 4 statements
```

In the last example (SYMBZ), one source line contains four statements. Three of them are terminated with an !, the last by a ;. Identical object code would be generated if the ! were replaced by End-of-Line <EOL>. When an ! and a ; occur inside quotation marks, they do not function as separators.

COMMENTS

A comment begins with a ; following the variable field of a statement. A comment affects neither the assembler operation nor the object code generated.

Comments that begin in column 1 are full-line comments; they begin with a ; or an *. (Please note that an * signifies a comment only when found in column 1 — column 1 of input is listed at column 21 on an output listing.) A comment is terminated by EOL.

```
LABEL:   LDA   129        ;This is a "comment."
;This is a full-line comment.
*This is another full-line comment.
FROG:    STA   MEM,X      This is not a legal comment.
                          ;(above comment needs a ;)
```

DEFINITIONS

Definitions begin with specific types of statements (MACRO, ECHO, IF). The end of a definition is dependent on what started the definition, for example, ENDM is used to terminate MACRO and ECHO definitions, while ENDIF terminates an IF range.

SYMBOLS AND NAMES

A symbol is a sequence of characters that identifies a value or a macro. The first character cannot be a digit. Symbols may be any length, but they must be unique in the first six characters. The following characters may be used in a symbol name:

A–Z	The uppercase letters of the alphabet
a–z	The lowercase letters of the alphabet (converted to uppercase by the assembler
:	May only be first character indicating a local symbol
?	If first character, then the symbol is excluded from the reference map
@	Additional alpha extension. Cannot be first character of an identifier, since it is also a prefix for octal numbers.
0–9	Digits

The underline character (__) may occur in a name as written but is discarded. Lowercase letters are mapped into the corresponding uppercase. When a colon occurs as the first character in a name, it denotes a name local to the current PROC (see PROC pseudo-op in Section 6). A colon at the end of a name in the label field is interpreted as a terminator but in any other position, it is ignored.

Examples:

```
ERROR__5:                              ;the assembler ignores __, label is
                                       ERROR5
          JMP       RESTART            ;the assembler uses first 6
                                       characters: 'RESTAR'
TEST      LDA       COUNT
          BNE       Error5             ;'Error5' converted to ERROR5
:LOCAL:   DEC                          ;:local: is a local symbol
```

NUMBERS

A number can be in any one of three forms, depending on the prefix.

Prefix	Base	
%	2	Binary
@	8	Octal
$	16	Hexadecimal

The lack of a prefix implies decimal.

Digits greater than the radix are not allowed. The underline character (__) is ignored.

The Macro Assembler provides constant conversion formatting for 6-byte real numbers as specified in the current ATARI BASIC. Real numbers are not valid expression arguments in variable fields. (See "REAL6," pseudo-op in Section 6).

Examples:

```
BINVAL    EQU       %10 001 010
OCTVAL    EQU       @212
HEXVAL    EQU       $8A
```

CHARACTER STRINGS

The assembler accepts ATASCII characters $20-$7E as valid characters. A character string consists of any sequence of characters surrounded by single quotation marks ('n . . . n'). Within a string, a single quotation mark character is represented by two successive single quotation marks.

Character strings can be used in the TITLE and SUBTTL statements, as a DB or DC subfield, or as operands of relational operators.

The LSTR operator returns the length of a character string (see "Expressions"in this section).

Examples:

```
TITLE   'Sample Expressions'
DB      'This is a STRING.',$9B
DB      'Control characters are illegal in a long string'
DB      $9B
                ;Nonprintable characters may be represented
                ;by using their hexadecimal values,' ,
                ;such as $9B for EOL',
DW      $2766,  'bp', 'BP' ;2-byte values
LDA     #43     ;a decimal number
ADC     #'C'    ;an ATASCII character
CMP     #''''   ;an ATASCII character
```

EXPRESSIONS

An expression consists of operands combined with operators to produce a value. Operators of equal precedence are evaluated left to right. Brackets can be used to override the order of evaluation, since 6502 instructions use parentheses for indirect addressing. Expressions are evaluated using 16-bit twos complement (unsigned) arithmetic. Overflow is ignored.

Real numbers are not valid arguments in expressions.

Examples:

```
DB      'Here are some fancy expressions:'
DB      43 + 22 shl 3 mod 6
DB      'Q' + REF1 xor [99 and REF2]
AND     low ['ZZ' - ['A' xor 'a' + ['A' xor 'a'] shl 8]]
DW      rev [*O - *L]
```

OPERANDS

An operand is either a symbol, an expression enclosed in brackets, a number, a character string, or one of the following special elements:

```
*    = current location counter
*L   = same as *
*O   = current value of origin counter
*P   = current position counter number of defined byte
```

See LOC and ORG pseudo-ops for further discussion of *L and *O. Refer to the VFD pseudo-op for details on *P.

The comparison operators return a value of zero for false and $FFFF for true. Numeric tests treat values as unsigned, so that [-1 < 0] will produce the answer false. Character string tests use the ATASCII collating sequence.

Operators

+		Sum or positive sign
−		Difference or negative sign
*		Multiply
/		Divide
NOT		Bit-by-bit complement
AND		Logical product, conjunction
&		Logical product, conjunction (same as AND)
OR		Logical sum, disjunction, inclusive OR
XOR		Logical difference, inequivalence, exclusive OR
=	EQ	Equality
<>	NE	Inequality
<	LT	Less than
<=	LE	Less than or equal to
>	GT	Greater than
>=	GE	Greater than or equal to
SHL		Shift left n bits
SHR		Shift right n bits
HIGH		Unary, high value to 8-bit field = x / 256
LOW		Unary, low value to 8-bit field = x MOD 256
MOD		Modulus function
REV		Unary Reverse = ((LOW x) left and right SHL 8) + (HIGH x)
DEF		Test symbol previously defined
LSTR		Return the length of a character string

Precedence Levels

Highest	Brackets
	HIGH LOW DEF REV LSTR
	* / MOD SHL SHR
	+ − unary
	+ − binary
	= <> < <= > >= NE EQ LT LE GT GE
	NOT
	& AND
Lowest	OR XOR

5

MACRO FACILITY

A macro is a sequence of source statements that are saved and then assembled through a macro call. A macro call consists of a reference to a macro name in the operation field of a statement. It often includes actual parameters to be substituted for formal parameters in the macro code sequence, so that code generated can vary with each assembly of the definition.

Use of a macro requires two steps: definition of the macro and reference to the macro.

MACRO DEFINITION

A macro definition consists of three parts: heading, body, and terminator.

Heading A macro definition starts with the name of the macro and the substitute parameter names in the variable field.

Body The body begins with the first statement after the heading that is not a comment line. The body consists of a series of instructions. All instructions other than END, including other macro definitions and calls, are legal within the body. However, a definition within a definition is not defined until the outer definition is called. Therefore, an inner definition cannot be called directly. Substitute parameters can occur anywhere in the body. They are prefixed by a percent sign (%):

```
%1 = first parameter
%2 = second
...
%9 = ninth parameter
%K = 4 hex digits, representing the serial number
     of this macro call
%L = the label field of the macro call
%M = the name of the macro
%% = replaced by a single percent
```

Terminator A macro definition is terminated by an ENDM pseudo-instruction. The assembler counts the nesting level of MACRO/ECHO and ENDM pairs occurring in a macro body, so that the definition is terminated only by the corresponding ENDM.

Note: The ENDM pseudo-op must be preceded by a tab (►) character. Press ESC CLR SET TAB to get the tab character.

MACRO CALL

A previously defined macro is called when its name occurs in the operation field of a statement. If actual parameters appear in the call, they are substituted for the corresponding formal parameter in the macro body without evaluation. Only after the entire body has been expanded does assembly resume. Thus the statements generated by the macro may themselves contain further macro calls or definitions, with the nesting limited only by available memory.

Note: When writing recursive macros, take care in the coding of the termination condition(s). A macro that repeatedly calls itself will cause the assembler to terminate (eventually) with the message "Memory Overflow."

CODE DUPLICATION

The ECHO pseudo-instruction is used to repeat a code sequence. It is written similarly to a macro definition but with the following differences: heading is ECHO, not MACRO; no parameters are involved; the variable field of the ECHO statement specifies how many times the body is to be repeated. ENDM is also used to terminate an ECHO sequence (see ECHO pseudo-op).

NESTING

ECHO, MACRO, and IF blocks may be nested in completely arbitrary fashion, subject only to the constraint that it be properly nested; i.e., each block must be contained within the surrounding block.

6

PSEUDO-OPERATIONS

The Macro Assembler provides a comprehensive set of pseudo-operations (pseudo-ops) that permits you to control the assembly process.

For ease of comprehension, the following notations are used in this manual:

iglab	means the label field is ignored by the pseudo-op
<exp>	means that an expression is required
[exp]	means that an expression may appear, at your option
{exp}	means that the item inside the braces { } may appear zero or more times

ASSERT

CHECK ASSEMBLY CONDITION

iglab ASSERT <exp>

where: iglab = ignored label field
exp = any legal expression: Nonzero implies true
Zero implies false

ASSERT allows you to check for and flag illogical assembly conditions such as incorrect parameter values, programs that are too large, and undefined symbols.

The expression is evaluated and a P error will be generated if the expression is false; i.e., if the expression evaluates to zero.

The expression is not examined in Pass 1 of the assembler, so ASSERT can correctly check any condition. Forward references in the expression are evaluated correctly.

Examples:

To check that the location counter in a given piece of code is within bounds, in this case below $2000, add the following line at the end of the assembly:

```
ASSERT *< $2000;test for limit exceeded
```

If the location counter reaches $2000, a P error will generate.

If you are writing a utility subroutine and wish to check that a required symbolic definition has been supplied by the user of the subroutine, you might code:

ASSERT DEF [SYMB1]

If the required symbol SYMB1 is not defined by the user within the assembly, a P error will be generated. Note that the check for symbol definition is postponed until after Pass 1, allowing you to define SYMB1 anywhere in the source code.

DB

DEFINE BYTE

LABEL: DB <exp>...,<exp>

where: <exp> = any legal expression, value, or string

DB allows you to directly specify the content of individual bytes of memory.

A string will generate as many bytes as it has characters; the first character will be the first byte generated. Characters in the string generate their 7-bit ATASCII codes without parity.

DB is used to intersperse code with text strings and for data tables.

The label field is significant; it will address the first byte generated.

Examples:

```
PNCHRS:  DB   ',./;@@<>?+!' '#$%&' '()_*= + (tm):-[]@',0
         DB   $80
         DB   LAB,LAB2,3,$46,$0AF,'xX',17+QVAL*4,'coffee'
```

DC

DEFINE CHARACTER

LABEL: DC <exp>...,<exp>

where: <exp> is any legal expression, value, or string

DC operates like DB, but the high-order bit (parity bit) of the last byte of each expression is set.

DC is used just like DB. The only difference is the parity bit of the last byte of each term.

Examples:

```
TBLHDR   DC   'This is a table of offsets'
ADRLST   DC   128, $36, $15, @21, 159
```

DS

DEFINE SPACE

LABEL: DS <exp16>

where: <exp16> = any legal expression, value, or string

DS allows you to reserve large blocks of memory. The expression <exp16> will be evaluated as an unsigned 16-bit value, and that value will be used to increment the assembler's internal origin and location counters.

Memory allocated is **not** initialized, and will contain unknown values at program execution time. The label field is significant; it will address the first memory byte allocated.

DS reserves space for use at execution time; it can be used to "skip over" an existing piece of ROM or provide for uninitialized data storage.

Example:

```
STORG:    DS    256 ;allocate 256 bytes
```

DW

DEFINE WORD

```
LABEL:        DW      <exp16>...,<exp16>
                      <exp16> = any expression or value or 1 to 2
                                character string
```

where: <exp16> = any expression, value, or string

DW defines the contents of blocks of memory. Values and expressions in the operand field are computed as unsigned 16-bit values and placed in memory as a machine word; the assembler places the Least Significant Byte (LSB) first, followed by the Most Significant Byte (MSB).

The label field is significant; it will address the first byte generated.

DW is intended to build tables of 16-bit values.

Examples:

```
;                          Table of Addresses
DW    PWRON          ;Power on
DW    MSTRST         ;Master reset
DW    SYSCAL         ;System calibrate
DW    RECAL          ;Recalibration
DW    PWRDN          ;Power down
DW    BUTTON         ;Button press
DW    EMERG          ;Emergency shutdown
DW    ACTN1,ACTN2,ACTN3    ;Action numbers 1,2,3
DW    0              ;End of table
```

ECHO...ENDM

ECHO BLOCK

```
LABEL:    ECHO    <exp>
          ...
          ENDM
```

where: <exp> = numeric expression

ECHO . . . ENDM is a simple code-duplication facility. Code between an ECHO and its ENDM will be assembled as many times as specified by the <exp>.

The label field is significant; it addresses the value of *O when the ECHO pseudo-op is encountered.

An ECHO . . . ENDM construct may not exceed 255 repetitions; 0 (zero) repetitions means the ECHO . . . ENDM code is skipped. ECHO . . . ENDM is convenient for repetitious coding problems. An ECHO . . . ENDM sequence is much easier to create and maintain than, say, 127 repetitions of a 6-line procedure.

Note: The ENDM pseudo-op must be preceded by a tab (►) character.

Example:

```
; The following example will create a table
; of 20 entries of 4 bytes each and
; initialize each entry to a value of
; $10 37 00 00.
;
TABLE:  ECHO    20                  ;20 times
        DB      $10, $37, $00, $00
        ENDM                        ;End table
```

EJECT

EJECT PAGE

```
iglab   EJECT
iglab   =  ignored label field
```

EJECT forces a page eject in the assembly listing if the listing is currently turned on.

EJECT can be used anywhere in an assembly source program.

The TITLE pseudo-op sets the internal title string and forces an EJECT.

Example:

```
EJECT
```

END

END PROGRAM

```
LABEL:      END [exp]
```

END tells the assembler where to stop assembly and begin the cross-reference map. The optional address field expression specifies the run address for an object program.

END must be the last statement of the last link file of an assembly.

The label field is significant, and addresses the value of the internal *O counter when the END is processed.

Example:

```
FREESP:     END     ;end of program
```

EQU or =

EQUATE VALUE TO SYMBOL

```
LABEL:    EQU     <exp16>
LABEL:    =       <exp16>
```

where: <exp16> = 16-bit expression or value or
1 to 2 character string

EQU defines the symbol on the left as the value of the 16-bit expression in the operand field.

EQU creates symbols (labels) for use with other assembler instructions. Unlike SET, EQU defines a fixed value to a symbol that cannot be changed during the assembly.

The operand <exp16> must be an absolute value at the time of evaluation; any symbols used in the expression must have been previously defined.

Examples:

```
TSTCHR    EQU  '$'
TS2CHR:   EQU  '@'
ZAP       EQU  $900
ZONK:     =    ZAP * 2
```

ERR

FORCE ERROR FLAG

ERR allows you to force an assembly error. The address field is ignored. When the assembler detects an impossible or undesirable condition at assembly time, ERR allows this to be flagged.

Examples:

```
IF   * > 4000h
ERR       ;Program too long
ENDIF
```

IF...ENDIF, IF...ELSE...ENDIF

```
iglab  IF   <exp>
       <code for special situation>
iglab  ENDIF
```

```
iglab  IF   <exp>
       <assembly code>
iglab  ELSE
       <assembly code>
iglab  ENDIF
```

where: <exp> = expression: nonzero = > true
zero = > false

IF . . . ENDIF and IF . . . ELSE . . . ENDIF control textual input to the assembler. At assembly time, <exp> is evaluated and the result determines where the assembler will resume assembling the input file.

Whenever a single program should be configured as two (or more) distinct versions, IF . . . ENDIF and IF . . . ELSE . . . ENDIF can test assembly-time values and assemble only the appropriate source lines.

Expression <exp> values for an IF must be numeric; strings greater than two characters are not allowed.

IF . . . ENDIF and IF . . . ELSE . . . ENDIF constructs are "nestable"; depth of nesting is limited only by memory space available at assembly time.

Any "label" in the label field is ignored; a descriptive name can be placed here to help associate an IF with its ELSE (if used) and ENDIF.

Examples:

```
        IF      1       ;1 is nonzero, therefore true
        JSR     OUTM
        JMP     BOOT    ;these two lines will be assembled
        ENDIF
LABEL:  IF      DEF X   ;Condition
        JSR     PATH1   ;LABEL is ignored, but
LABEL:  JMP     ELSE    ;assists readability.
        JMP     PATH2
        ENDIF
```

INCLUDE

INCLUDE ANOTHER SOURCE FILE

LABEL: INCLUDE <filespec>

where: <filespec> = <Dn:filename.ext>, n can be 1, 2, 3, or 4

INCLUDE specifies another file to be included in the assembly as if the contents of the referenced file appeared in place of the INCLUDE statement itself. The included file may contain other INCLUDE statements. The listing of code in INCLUDE files is controlled by the I option of the LIST pseudo-op. (See INCLUDE example.)

INCLUDE allows you to divide large programs into manageable pieces for ease of editing, common use of libraries, file manipulations, and so forth.

Example:

The command line

D:INCLDEX.ASM

combined with the following, file setup:

```
<INCLDEX.ASM contents>
        TITLE       'INCLUDE example'
        ORG         $100
        INCLUDE     D:L1
        INCLUDE     D:L2
        INCLUDE     D2:L3.ACD
;*** End INCLDEX.ASM
```

```
<D:L1 contents>
                LDA             L1VAL
;*** End L1. ASM

<D:L2 contents>
                LDA             L2VAL
;*** End L2.ASM

<D2:L3.ACD contents>
L1VAL           DB              '*'
L2VAL           DB              0
                END             ;Stop assembly here.
;*** End L3.ACD
```

This would input to the assembler the following sequence of code:

```
                TITLE           'INCLUDE example'
                ORG             $100
                LDA             L1VAL
;*** End L1.ASM
                LDA             L2VAL
;*** End L2.ASM
L1VAL           DB              '*'
L2VAL           DB              0
                END             ;Stop assembly here.
;*** End L3.ACD
;*** End INCLDEX.ASM
```

LINK

LINK TO ANOTHER SOURCE FILE

iglab LINK <filespec>

where: <filespec> = <Dn:filename.ext>, n can be 1, 2, 3, or 4

The LINK pseudo-op is similar to the INCLUDE facility, except that link files are not assembled until the assembler reaches the end of the current input file. Whenever a LINK pseudo-op is found, it is stored away for processing along with any other LINK statements encountered when the current file is finished processing.

Each source file that contains links to other files will be completely processed, and its links will then be processed in order of occurrence. Any link that contains sublinks will be processed in an identical manner; link files may nest arbitrarily deep, as long as the total number of files does not exceed 40.

If A, Q, S, T, U, and X are assembly-code files, and if A links to Q, S, and X, and S links to T, and T links to U, then the order of assembly will be:

A, Q, S, T, U, X.

If the <filespec> extension is missing, it defaults to the extension used in the current input file; i.e., the file that contains the LINK pseudo-op.

Examples:

```
        Link    D2:PART1        ;Assemble file 'D2:PART1'

                                ;using the same extension as
                                ;the primary file
        LINK    D:UTIL.ACD
BLORP:  LINK    D2:PART2.ASM    ;'BLORP' is ignored
```

LINK allows you to divide large programs into manageable pieces for ease of editing, common use of libraries, file manipulations, and so forth. The LINK facility supports linking across diskettes, so the entire source program does not have to be contained on the same diskette.

Example:

The command line

```
AMAC D:LINKEG.ASM
```

combined with the following link file setup:

```
<LINKEG.ASM contents>
        TITLE   'LINK example'
        ORG     $100
        LINK    D:L1
        LINK    D:L2
        LINK    D2:L3.ACD
;***    Endx    LINKEG.asm

<D:L1 contents>
        LDA     L1VAL
;***    Endx    L1.asm
<D:L2 contents>
        LDA     L2VAL
;***    Endx    L2.asm

<D2:L3.ACD contents>
L1VAL   DB      '*'
L2VAL   DB      0
        END     ;Stop assembly here.
;***    Endx L3.acd
```

would input to the assembler the following sequence of code:

```
        TITLE   'LINK example'
        ORG     $100
;***    Endx    LINKEG.asm
        LDA     L1VAL
;***    Endx    L1.asm
        LDA     L2VAL
;***    Endx    L2.asm
L1VAL   DB      '*'
L2VAL   DB      0
        END     ;Stop assembly here.
;***    Endx    L3.acd
```

LIST

OUTPUT LISTING CONTROL

```
iglab   LIST   *
iglab   LIST   <opt>...,<opt>
```

where: <opt> = optional minus sign followed by one of the following.

C List listing controls: EJECT, PAGE, SPACE, SUBTTL, and TITLE lines. (Default OFF.)

D List **detailed** code: i.e., list every byte generated by DB, DW, VFD, multi-line statements, and so forth.

F List code skipped by IF...ENDIF or IF...ELSE...ENDIF. (Default ON.)

G List **all** generated code: i.e., list every byte placed in the output object file, regardless of origin. Overrides -L. (Default OFF.)

I List code in INCLUDE files. (Default OFF.)

L Master LIST control. When -L option is in effect, nothing is listed except lines with errors, or when -L is overridden by the G option. (Default ON.)

M List all lines generated by macro references. (Default ON.)

R Accumulate cross-references. (Default ON.)

S List code referenced in a systext file. (Default OFF.)

LIST controls the listing produced during an assembly. However when an L=0 command line option is selected, LIST pseudo-op has no effect. The variable-field argument to LIST must be an *, or a set of options.

The LIST pseudo-op operates on a stack: each element of the stack is a set of option flags. The flag on top of the stack controls the content of the listing produced. Each call to the LIST pseudo-op will push, or pop, a flag on or from the stack.

"LIST *" means pop the list-option stack.

"LIST M" means make a copy of the current flag, setting the M-flag to ON, and push the new flag setting onto the stack.

LIST has obvious applications for detailed listing of newly written code, detailed listing of untested macro expansions, and suppressing the listing of library code.

Example:

A common code library may contain a set of routines all having the following IF block at the beginning:

```
IF      ILIST = 0    ;if common code list turned off
LIST    -L, -R       ;no listing, no references
ENDIF
```

Assume that the global symbol ILIST equals zero. A new flag setting is pushed onto the LIST option stack; the options (-L, -R) specify no listing is to be printed, and no cross-reference accumulation is to be done.

Each common code routine also has this IF...ENDIF at its end:

```
IF      ILIST = 0   ;if common code listing was off
LIST    *           ;go back to original list options
ENDIF
```

Now that the common code routine has been assembled, the LIST option stack will be popped. This returns the LIST option stack to its condition before the library was assembled.

LOC

SET LOCATION COUNTER

LABEL: LOC <exp16>

where: <exp16> = 16-bit expression or value

LOC sets the location counter. The expression is evaluated as an unsigned 16-bit value and assigned to the Macro Assembler's internal location counter (*L).

Code generated while the internal LOC counter (*L or *) does not equal the internal ORG counter (*O) will be flagged with # in column 7 of the listing.

The label field is significant; the label defined there will be set to the value of *L **before** *L is changed to <exp16>.

LOC assists you in generating self-overlaying programs. Code generated that way can be positioned anywhere in memory (using ORG), and the code will assemble as if it was located at the address expressed in the LOC statement. Of course, the code must be moved at run time to the address specified in its LOC statement before it can be executed.

Code assembled in one place for execution elsewhere can be especially handy for ROM-resident software, when pieces of code are copied from ROM to RAM before execution.

LOC is also useful for enhancing the readability of data tables for code conversion. The following example is a table of external BCD codes. The location counter is set to the ATASCII value of the first character in the table. In that way, the location field of the assembly listing contains an ATASCII value and the generated code field contains its associated external BCD value.

Examples:

```
                                ;Example of using LOC to enhance readability of
                                ;listings. The location counter will be set to
                                ;the ATASCII value that corresponds to the first
                                ;entry of a table of external BCD values.
                                ;
0000    = 5000                  ORG   $5000
5000    = 0041#                 LOC   'A'
```

```
0041#  61        EBCTBL:  DB    $61    ;The LOC field of the listing
0042#  62                 DB    $62    ;contains the ATASCII value
0043#  63                 DB    $63    ;which corresponds to the
0044#  64                 DB    $64    ;external BCD value in the
0045#  65                 DB    $65    ;generated code field.
                          ...
                          END
```

No ERRORS, 1 labels, $2403 free.

```
nEBCTBL 0041                1# 8

                            ;Example of code to be assembled at $2000
                             to be
                            ;transferred to a ROM at $0F000

        = 0500     COUNT    EQU    $0500  ;RAM working
                                           storage
0000    = 2000          ORG   $2000
2000    = F000#         LOC   $0F000
F000#   A907            LDA   #07
F002#   8D0005          STA   COUNT
F005#   4C0AF0          JMP   L1
F008#   EA              NOP
F009#   EA              NOP
F00A#   CE0005    L1    DEC   COUNT
F00D#   EA              NOP
F00E#                   END
```

No ERRORS, 2 labels, $23F7 free.

```
COUNT  0500   1# 4  1/ 8   1/12
L1     F00A   1/ 9  1#12
```

MACRO...ENDM

MACRO DEFINITION

```
MACNAM:   MACRO     parm1, ..., parmn
          <body>
          ENDM      ;end of MACNAM definition
```

where: <body> = any desired text which may include:

%1..%9	=	parameters number 1 ... 9
%K	=	hexadecimal number of this macro call
%L	=	label field of macro call
%M	=	name of the macro

MACRO . . . ENDM is the macro definition construct.

The symbols in the variable field represent substitutable parameters. The symbol names are for documentation purposes only and may **not** appear in the body of the macro.

Parameters within the macro are represented by %x, where x is replaced with a decimal digit (1-9). %K within the body will be replaced with the serial number of the macro call as four hexadecimal digits. %L within the body will be replaced with the label field of the macro call. %M within the body will be replaced with the macro call.

The label field is significant; it denotes the name of the macro during an assembly.

Note: The ENDM pseudo-op must be preceded by a tab (►) character.

Macros may generate lines which turn out to be macro calls. Thus, a macro may directly or indirectly call itself. Care must be taken so that such a "recursive macro" does not call itself indefinitely.

Macros can be used to generate many copies of a procedure with different internal constants, or in conjunction with VFD to assemble fancy machine op codes (see VFD pseudo-op). There are many other potential uses for macros; these examples are only intended to demonstrate some of these uses.

Example:

One way to find the number of bits needed to contain a value is to compute the logarithm base 2 of the value. To do that at assembly time, we can use recursive macro calls to achieve a looping effect. Note that the condition tested on VAL ensures that the series of nested calls must eventually terminate.

```
;           COMPUTE SYM=Log 2
LOG2:       MACRO SYM,VAL
            IF          [%2] > 1
            LOG2        %1,[%2]/2
%1:         SET         %1+1
            ELSE
%1:         SET         0
            ENDIF
            ENDM
```

Example:

```
;macro to take the high nibble from a memory location
;and the low nibble from the accumulator, storing the
;result in the accumulator

NPACK:      MACRO       ADDR
            EOR         %1
            AND         #0F
            EOR         %1
            ENDM
```

Example:

It is sometimes necessary to be able to create a symbol name that is different for each call of a macro. The %K implicit parameter feature provides the means to do this. In the following macro, a unique jump-target label is created on each call. Note that all the labels begin with the ? character so that they will not clutter up the symbol table map.

```
;           Set accumulator= 0 if sign bit is set.
PARVAL:     MACRO
            BMI         ?%k
            LDA         #0
?%K:
            ENDM
```

ORG

ORIGIN COUNTER

LABEL: ORG <exp16>

where: <exp16> = any absolute, previously defined 16-bit value or expression

ORG sets the address of the first byte of a piece of code (or data) to a physical location in memory.

The label field is significant; it will address the value of *L, **before** <exp16> is evaluated.

The ORG command can be used in a program as often as desired. ORG cannot change the current USE block. (See USE pseudo-op.) ORG changes the block-relative value of the origin and location counters of the current USE block.

ORG is almost always used at the beginning of an assembly to define the starting position in memory of the resultant code. If not explicitly set by ORG (or the O= command-line parameter), the default value of the origin and location counters is zero.

Example:

```
PROG:   ORG     $100    ;Assemble at location $0100
SOCK:   ORG     *O      ;assign *O to *O and *L
```

PROC...EPROC

DEFINE LOCAL SYMBOL RANGE

```
LABEL:  PROC
        <body>
        EPROC
```

PROC tells the assembler that the following code is a procedure that may contain local symbols. A local symbol is a symbol that begins with a colon (:). It does not appear in the cross-reference map and cannot be referenced outside of the PROC range.

The label field is significant; it addresses the value of the *O counter when the PROC statement is processed.

PROC should be the first instruction of any procedure that contains local symbols.

A PROC is terminated by EPROC or the next PROC.

When assembling large programs where symbol table space is at a premium, local symbols can be used whenever appropriate to reduce memory requirements.

Example:

```
INIT:   PROC                ;procedure
        LDA     #0          ;let A=0
        LDY     #0          ;Y indexes through memory
:Loop:  STA     (BEGMEM),Y  ;:Loop: is local symbol
        INY                 ;-won't appear in cross-reference
        BNE     :LOOP       ;Write 256 locations
```

REAL6

DEFINE REAL NUMBER VALUE

LABEL: REAL6 <fpnum>

where: <fpnum> is a floating point number

REAL6 provides constant conversion into 6-byte real numbers as supported by the ATARI operating system.

The label is significant because it denotes the starting location of 6 bytes of the converted number.

Example:

```
PI:         REAL6       3.14159
```

SET

DEFINE VALUE FOR SYMBOL

LABEL: SET <exp>

where: <exp> = numeric expression

The SET pseudo-op defines a symbol to a value representing the 16-bit expression of the operand field. SET works just like EQU, except that LABELs defined with SET may be redefined.

The expression in the variable field must be an absolute value at the time of evaluation. Any symbols used must have been previously defined.

Example:

```
TSTVAL      SET         027h
                        ...
                        DB          TSTVAL
                        ...
TSTVAL      SET         099h
                        ...
TSTVAL      SET         063h
```

SPACE

OUTPUT BLANK LINES TO LISTING

```
iglab   SPACE   <exp1>
iglab   SPACE   <exp1>,<exp2>
```

where: <exp1>,<exp2> = unsigned, numeric expressions

SPACE places blank lines in a listing. If SPACE has one argument, it will output that many blank lines only if doing so will not exceed the length of the current page. If <exp1> lines will not fit on the current page, SPACE will force an EJECT.

If SPACE has two arguments, they are both evaluated and <exp1> blank lines will be placed in the (currently on) listing **only if** the current page will have <exp2> lines left afterwards. If the current page does not have that sufficient room, SPACE will force an EJECT.

SPACE is useful when inserted just before a small procedure if X is the length of the procedure (X lines),

```
SPACE     4,X
<procedure>
```

will output 4 lines to the listing if the procedure will still fit on the current page. If the spacing and the procedure will not fit on the current page, SPACE will force an EJECT.

SUBTTL

DEFINE SECOND LINE OF OUTPUT LISTING

iglab SUBTTL <string>

where: <string> = any string up to 32 characters

SUBTTL allows you to specify secondary title information. SUBTTL without a <string> argument is ignored. To erase the current subtitle, use an empty string.

Example:

```
TITLE       'Section 8 — Pseudo-Ops'
SUBTTL      'SUBTTL syntax and description'
SUBTTL      ' ' ; erase current subtitle
```

TITLE

DEFINE FIRST LINE OF OUTPUT LISTING

iglab TITLE <string>

where: <string> = any string up to 32 characters

TITLE allows you to set/reset the assembler's internal page-heading string. TITLE with a string argument will place that string in the page header (see "Sample Listing," Section 3). If the string contains zero characters, the page header is reset to empty. TITLE without a string argument does not alter the current page header.

The first call to TITLE * will **not** eject a listing page; successive calls will always force an EJECT after any arguments are processed.

TITLE is commonly placed at the beginning of each file used in an assembly. Each linked file will begin assembly on a fresh page, topped with an appropriate header to describe its general contents.

Example:

```
TITLE       'XONC.asm — Interface Subroutines.'
```

USE

DEFINE BLOCK AREA

iglab USE name

USE establishes a new "USE block" or resumes use of a previously established block. The block in use is the block into which code is subsequently assembled. A program may contain up to 60 different USE blocks. The assembler is responsible for computing the length and actual origin of each block. Origins are assigned to each block in the order they are first encountered.

Associated with each USE block are registers to maintain the last values of the origin and position counters (*O and *P). See ORG and VFD for a description of those counters. Initially, the values of these counters default to zero for each USE block. The value of the location counter (*L) is not saved, but set equal to the value of the origin counter. If a LOC had been in effect previously, resetting of the location counter to produce the desired results is the responsibility of the programmer.

USE allows the programmer to specify consecutive pieces of code in discontiguous source segments. It is more convenient than using ORG.

Example:

```
        USE     BTABL       ;(at beginning of program)
BTABL:                      ;define base of jump vector
        USE     *           ;(return to normal org)
        ...
NXLAB:  LDX     Something
        USE     BTABL
        DW      NXLAB       ;add address to jump vector
        USE     *
        STX     Addr        ;more
        ...
        USE     BTABL       ;(at end of program)
        DW      0           ;mark end of vector
        USE     *
        END
```

VFD

VARIABLE FIELD DEFINITION

LABEL: VFD <Fexp>\<exp>, ..., <Fexp>\<exp>

where: 1 <= <Fexp> <= 16
<exp> = any numeric expression

VFD defines variable fields. Each <Fexp> denotes a field width. Each <exp> denotes an expression to be placed into that field; <exp> values that exceed their associate <Fexp> field width values are truncated to match the <Fexp> value.

Negative values are evaluated with unsigned twos-complement arithmetic. For example, -32768 is 32768 and -1 will be represented by 65535. The resultant values are truncated to match the <Fexp> field width.

VFD manipulates the position counter (*P) to keep track of the bits remaining in a byte at the end of a VFD pseudo-op. If the next pseudo-op encountered is another VFD, the next field generated will begin with the unused bits left in the current byte. If the next code-generating pseudo-op is not VFD, the assembler will pad out the unused byte field with zeros.

VFD allows you to specify arbitrarily complex data fields without regard to byte or word boundaries.

Example:

```
MVINST:     VFD 2\01,3\DDD,3\SSS
```

VFD can be used this way inside MACRO-ENDM constructs to assemble code for unusual processors, special peripheral chips, and so forth.

Example:

```
SPEC:       VFD   7\@43,9\I'&&'
            VFD   13\$429
```

SPEC is a label point to a 29-bit field definition. The first 7 bits contain the value 43 octal. The next 9 bits contain the truncated string &&. The next 13 bits contain the value 429 hexadecimal. The *P counter currently points into the fourth byte after SPEC, with 3 bits left in the current byte.

7

PSEUDO-OP QUICK REFERENCE

```
iglab     ASSERT    <exp>              ;Check assembly condition
LABEL:    DB        <exp>,<exp>        ;Define bytes
LABEL:    DB        'ABCDE','f',$0D    ;Define long strings
LABEL:    DC        'ABCDE'            ;DB with 80h added onto the last byte
LABEL:    DS        <exp>              ;Define space
LABEL:    DW        <exp>,<exp>        ;Define words
LABEL:    DW        'Xu',1234,'y'      ;Define 1- or 2-character strings
LABEL:    ECHO      <exp>              ;Duplicate code <exp> times
iglab     EJECT                        ;Page eject
iglab     ELSE                         ;Part of conditional assembly
LABEL:    END       [exp]              ;End of assembly
iglab     ENDIF                        ;Terminate range of IF
iglab     ENDM                         ;Terminate MACRO or ECHO
iglab     EPROC                        ;Terminates local symbol range
LABEL:    EQU       <exp>              ;Define LABEL equals <exp>
iglab     ERR                          ;Force error flag
iglab     IF        <exp>              ;Begin conditional assembly
LABEL:    INCLUDE   <filespec>         ;Include another source file
iglab     LINK      <filespec>         ;Include another source file
                                       at the end of this source file
iglab     LIST      <opt>              ;<opt> = list control option
iglab     LIST      *                  ;Pop list control stack
LABEL:    LOC       <exp>              ;Set location counter
NAME:     MACRO     <parms>            ;Begin macro definition
LABEL:    ORG       <exp>              ;Set origin counter
LABEL:    PROC                         ;Begin local symbol range
LABEL:    REAL6     <exp>              ;6-byte real constant conversion
LABEL:    SET       <exp>              ;Reset LABEL to <exp>
iglab     SPACE     <exp1>,<exp2>      ;Space <exp1> lines if <exp2> lines
                                       left on this page
iglab     SUBTTL    'text'             ;Set listing subtitle
iglab     TITLE     'text'             ;Set listing title
iglab     USE       <name>             ;Use block declaration
LABEL:    VFD       <exp><exp>,..      ;Variable field definition
LABEL:    =         <exp>              ;Synonym for EQU
```

```
<exp>  =  required expression
[exp]  =  optional expression
'text' =  strings
<filespec>  =  <device>:<filename>.<extension>
iglab  =  ignored label
```

8

INSTRUCTION MNEMONICS

The instruction mnemonics provided by the Macro Assembler are identical to the standard mnemonics defined by MOS Technology, with these exceptions:

- Quotation marks denoting character strings must be properly paired. (Some 6502 assemblers allow an unterminated quote for a 1-character string.)
- In this assembler, the symbols < and > are binary operators (less than and greater than). Some 6502 assemblers define these symbols as unary operators (high and low). See Section 4 for operator definitions.

Examples:

AMAC		MOS	
CMP	#'?'	CMP	#'?
LDX	#high EXP	LDX	#>EXP
LDY	#low EXP	LDY	#<EXP

Notation

dd	8-bit signed displacement: -128 <= dd <= +127
mmmm	16-bit address expression
nn	8-bit constant: 0<= nn <= 255
rel	16-bit address within: *-126 <= rel <= *+129
zz	Page 0 location: 0 <= zz <= 255

HEX	OPCODE	ADDRESS	REMARKS
DATA MOVEMENT			
	Register to register transfer.		
AA	TAX		;Transfer A to X
A8	TAY		;Transfer A to Y
BA	TSX		;Transfer S to X
8A	TXA		;Transfer X to A
9A	TXS		;Transfer X to S
98	TYA		;Transfer Y to A
	Load constant into register.		
A9	LDA	#nn	
A2	LDX	#nn	
A0	LDY	#nn	

Load register from memory.

A5	LDA	zz
B5	LDA	zz,X
A1	LDA	(zz,X)
B1	LDA	(zz),Y
AD	LDA	mmmm
BD	LDA	mmmm,X
B9	LDA	mmmm,Y
A6	LDX	zz
B6	LDX	zz,Y
AE	LDX	mmmm
BE	LDX	mmmm,Y
A4	LDY	zz
B4	LDY	zz,X
AC	LDY	mmmm
BC	LDY	mmmm,X

Store register into memory.

85	STA	zz
95	STA	zz,X
81	STA	(zz,X)
91	STA	(zz),Y
8D	STA	mmmm
9D	STA	mmmm,X
99	STA	mmmm,Y
86	STX	zz
96	STX	zz,Y
8E	STX	mmmm
84	STY	zz
94	STY	zz,X
8C	STY	mmmm

Stack load/stores.

48	PHA	;Push accumulator
08	PHP	;Push processor status
68	PLA	;Pop accumulator
28	PLP	;Pop processor status

DYADIC ARITHMETIC

Add operand and carry.

69	ADC	#nn
65	ADC	zz
75	ADC	zz,X
61	ADC	(zz,X)
71	ADC	(zz),Y
6D	ADC	mmmm
7D	ADC	mmmm,X
79	ADC	mmmm,Y

```
                Subtract operand and borrow.
E9      SBC     #nn
E5      SBC     zz
F5      SBC     zz,X
E1      SBC     (zz,X)
F1      SBC     (zz),Y
ED      SBC     mmmm
FD      SBC     mmmm,X
F9      SBC     mmmm,Y

                Compare 8-bit operand with accumulator.
                Set flags as if subtracting, but do not alter accumulator.
C9      CMP     #nn
C5      CMP     zz
D5      CMP     zz,X
C1      CMP     (zz,X)
D1      CMP     (zz),Y
CD      CMP     mmmm
DD      CMP     mmmm,X
D9      CMP     mmmm,Y

                Compare 8-bit operand with index register.
E0      CPX     #nn
E4      CPX     zz
EC      CPX     mmmm
C0      CPY     #nn
C4      CPY     zz
CC      CPY     mmmm

MONADIC ARITHMETIC

                Decrement by 1.
C6      DEC     zz
D6      DEC     zz,X
CE      DEC     mmmm
DE      DEC     mmmm,X
CA      DEX
88      DEY

                Increment by 1.
E6      INC     zz
F6      INC     zz,X
EE      INC     mmmm
FE      INC     mmmm,X
E8      INX
C8      INY

                Arithmetic control.
18      CLC                     ;Clear carry flag
D8      CLD                     ;Clear decimal mode
B8      CLV                     ;Set overflow flag
38      SEC                     ;Set carry flag
F8      SED                     ;Set decimal mode
```

DYADIC LOGICAL/BOOLEAN OPERATIONS

8-bit logical product, conjunction.

29	AND	#nn
25	AND	zz
35	AND	zz,X
21	AND	(zz,X)
31	AND	(zz),Y
2D	AND	mmmm
3D	AND	mmmm,X
39	AND	mmmm,Y

Logical sum, disjunction, inclusive OR.

09	ORA	#nn
05	ORA	zz
15	ORA	zz,X
01	ORA	(zz,X)
11	ORA	(zz),Y
0D	ORA	mmmm
1D	ORA	mmmm,X
19	ORA	mmmm,Y

Logical difference, inequivalence, exclusive OR.

49	EOR	#nn
45	EOR	zz
55	EOR	zz,X
41	EOR	(zz,X)
51	EOR	(zz),Y
4D	EOR	mmmm
5D	EOR	mmmm,X
59	EOR	mmmm,Y

Logical compare.
Set flags as follows:
Z=1 if A AND mem = 0
Z=0 if A AND mem = 1
S=bit 7 of mem
V=bit 6 of mem
(mem = mmmm or zz).

24	BIT	zz
2C	BIT	mmmm

ROTATE AND SHIFT

Arithmetic shift left.

0A	ASL	A
06	ASL	zz
16	ASL	zz,X
0E	ASL	mmmm
1E	ASL	mmmm,X

```
                Logical shift right.
4A              LSR             A
46              LSR             zz
56              LSR             zz,X
4E              LSR             mmmm
5E              LSR             mmmm,X

                Rotate left.
2A              ROL             A
26              ROL             zz
36              ROL             zz,X
2E              ROL             mmmm
3E              ROL             mmmm,X

                Rotate right.
6A              ROR             A
66              ROR             zz
76              ROR             zz,X
6E              ROR             mmmm
7E              ROR             mmmm,X

JUMPS
90              BCC                             ;If carry clear
B0              BCS                             ;If carry set
F0              BEQ                             ;If equal (=0)
30              BMI                             ;If minus
D0              BNE                             ;If not equal (<>0)
10              BPL                             ;If plus
50              BVC                             ;If overflow clear
70              BVS                             ;If overflow set
4C              JMP             mmmm
6C              JMP             (mmmm)

CALL SUBROUTINE
00              BRK                             ;Software interrupt
20              JSR             mmmm            ;Jump subroutine

RETURN FROM SUBROUTINE
40              RTI                             ;Return from interrupt
60              RTS                             ;Return from subroutine

MISCELLANEOUS CPU CONTROL
58              CLI                             ;Clear interrupt mask (EI)
EA              NOP
78              SEI                             ;Set interrupt mask (DI)
```

9

USING THE ATARI MACRO ASSEMBLER WITH THE ATARI ASSEMBLER EDITOR SOURCE FILES

If you have a source program that has been developed using the ATARI Assembler Editor cartridge, and you want to use the Macro Assembler to assemble it, you will have to be aware of the following differences:

- The Macro Assembler does not accept line numbers.
- The = for EQU must be embedded between at least two blanks.
- Comments must be preceded by a semicolon.
- The following pseudo-ops are recognized by the Macro Assembler:

  .BYTE is equivalent to DB
  .END is equivalent to END
  .PAGE is equivalent to TITLE
  .SKIP is equivalent to SPACE
  .WORD is equivalent to DW

- The following are **not** recognized by the Macro Assembler:

  BYTE
  WORD

- The Macro Assembler does not recognize * = for setting the origin counter; use ORG instead.
- All strings must be bracketed by quotation marks (") for the Macro Assembler to interpret them properly.

10

ERROR CODES

Errors are flagged by a single-letter code in column one of the output listing. Lines containing errors are always written to the screen, regardless of the output selection.

A = Address error. Instruction specified does not support the addressing mode specified.

D = Duplicate label error. The last one defined is used.

E = Expression error. An expression on the source line in the address field is unrecognizable.

F = Bad nesting of control statements. Bad nesting of IF . . . ELSE . . . ENDIF statements. When this occurs on the END line, it means an IF was not terminated.

I = Instruction field not recognized. Three NOP bytes are generated.

L = Label field not recognized. Three NOP bytes are generated.

M = MACRO statement error. Improper macro definition.

N = Error in number: digit exceeds radix; value exceeds 16 bits, and so forth.

O = Stack table overflow occurred in evaluating expression; user should simplify expression. Too many LINK files. Too many PROCs. Too many USE blocks.

P = Programmer-forced error. See ASSERT and ERR pseudo-ops.

R = Expression in variable field not computable.

S = Syntax error in statement. Too many or too few address subfields.

U = Reference to an undefined symbol.

V = Expression overflow: resultant value is truncated.

W = Not within VFD field width (1 < = width < = 16).

Y = Misplaced instruction: extraneous ENDM. When this occurs on the END line, it means a MACRO or ECHO was not terminated. Make sure that ENDM is preceded by a tab (►) character.

LIMITED 90-DAY WARRANTY
ON ATARI® HOME COMPUTER PRODUCTS

ATARI, INC ("ATARI") warrants to the original consumer purchaser that this ATARI Home Computer Product (not including computer programs) shall be free from any defects in material or workmanship for a period of 90 days from the date of purchase. If any such defect is discovered within the warranty period, ATARI's sole obligation will be to repair or replace, at its election, the Computer Product free of charge on receipt of the unit (charges prepaid, if mailed or shipped) with proof of date of purchase satisfactory to ATARI at any authorized ATARI Computer Service Center. For the location of an authorized ATARI Computer Service Center nearest you,

call toll-free: In California (800) 672-1430
Continental U.S. (800) 538-8737

or write to: Atari, Inc.
Customer Service/Field Support
1340 Bordeaux Drive
Sunnyvale, CA 94086

YOU MUST RETURN DEFECTIVE COMPUTER PRODUCTS TO AN AUTHORIZED ATARI COMPUTER SERVICE CENTER FOR IN-WARRANTY REPAIR.

This warranty shall not apply if the Computer Product: (i) has been misused or shows signs of excessive wear, (ii) has been damaged by being used with any products not supplied by ATARI, or (iii) has been damaged by being serviced or modified by anyone other than an authorized ATARI Computer Service Center.

ANY APPLICABLE IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE HEREBY LIMITED TO NINETY DAYS FROM THE DATE OF PURCHASE. CONSEQUENTIAL OR INCIDENTAL DAMAGES RESULTING FROM A BREACH OF ANY APPLICABLE EXPRESS OR IMPLIED WARRANTIES ARE HEREBY EXCLUDED. Some states do not allow limitations on how long an implied warranty lasts or do not allow the exclusion or limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

DISCLAIMER OR WARRANTY
ON ATARI COMPUTER PROGRAMS

All ATARI computer programs are distributed on an "as is" basis without warranty of any kind. The entire risk as to the quality and performance of such programs is with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair.

ATARI shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss, or damage caused directly or indirectly by computer programs sold by ATARI. This disclaimer includes but is not limited to any interruption of service, loss of business or anticipatory profits, or consequential damages resulting from the use or operation of such computer programs.

REPAIR SERVICE

If your ATARI Home Computer Product requires repair other than under warranty, please contact your local authorized ATARI Computer Service Center for repair information.

IMPORTANT: If you ship your ATARI Home Computer Product, package it securely and ship it, charges prepaid and insured, by parcel post or United Parcel Service.

PRINTED IN U.S.A.

A Warner Communications Company

CO60028 REV. 1